phuse·connect
connect·share·advance
10th–13th November
RAI Amsterdam
The Netherlands
EU 2019
The Clinical Data
Science Conference

Paper SI08

Making data mapping process easier and smarter with SAS and R

Andrii Artemchuk, Intego Group, Kharkiv, Ukraine

ABSTRACT

A Case Report Form or CRF is a tool, used in clinical trials to collect the patients' data and deliver it to the regulatory affairs in a standard format (SDTM). The CRF design may vary, thus every trial involves a unique SDTM mapping process.

This paper describes macros that read the data from the PDF template of the CRF. The macros search for the CRF fields with the list of predefined and fixed values, convert them into SAS datasets using PDFtools package in R, and compare to SDTM controlled terminology data, provided by CDISC. This functionality helps to reduce the time of mapping the raw data.

The result of the macros' execution is the SAS code that can be directly used for correct mapping of CRF values to the SDTM's.

The macros can check the SDTM data to make sure they do not contain typos or unexpected values.

INTRODUCTION

The SDTM mapping process, from a CRF annotation to getting final SDTM datasets has already been explored thoroughly. However, there is still no tool that could automatically transform the collected raw CRF data into the SDTM format. This is so due to several reasons. First of all, to understand what raw data values are collected, one would need to parse the corresponding text values, stored in the CRF. Most CRF's have a PDF version, and there is a complexity in parsing the PDF file format to make it further useable in SAS. Secondly, only particular data that are required for the SDTM mapping need to be selected out of the text being read. And finally, the all the CRF data have to be compared to the data from the CDISC SDTM Controlled Terminology to make correspondence. This paper proposes a consequent solution of each of these problems using the means of SAS and R programming languages.

LIMITATIONS

Every SDTM mapping from a statistical programmer point of view begins with the CRF. Therefore, the first thing the proposed algorithm deals with is the CRF processing. The goal is to read the PDF version of the CRF into the SAS dataset(s). Generally, the layout of the CRF can vary, and it imposes some limitations, as not every PDF version of the CRF (hereinafter "CRF document") may be processed.

The first limitation is that a CRF document has to contain the text that can be highlighted by the cursor or can be found by searching through the document using a tool like Adobe Acrobat Reader® or similar one. During the parsing this text will be extracted and saved in the SAS-dataset.

The second limitation is the CRF text field recognition methods. This paper considers the only fields that contain pre-defined text values or tick-boxes, as shown below:

Subject's Best Response	CR (Complete Response) ☐
	PR (Partial Response) ☐
	SD (Stable Disease) ☐
	PD (Progressive Disease) ☐
	NE (Not Evaluable) ☐
	NA (Not Applicable) ☐

The tick-boxes are good as they are the most likely to have the common logic in the physical location on the CRF page. It means that they are the easiest to be found among the other CRF text fields.

The third limitation arises from the second one – the tick-boxes and the related text should lie flat in relation to the left or the right margin of the page. In other words, there should be a regularity in their placement within the CRF document. The recognition algorithm, shown in this paper, allows a small error in the placement of the tick-boxes; however, if they are placed chaotically then the automatic recognition will be impossible. An example of such CRF document can be seen below:

End of treatment

Primary reason for end of treatment:

*Abnormal laboratory values or test results constitute adverse events if they induce clinical signs or symptoms, are considered clinically significant or require therapy."
If this applies, please change reason for end of treatment to adverse event

- ☐ 1 Adverse event(s) *(also specify on the Adverse events page)*
- ☐ 2 Abnormal laboratory value(s) *(please specify below)*
- ☐ 3 Abnormal test procedure result(s) *(please specify below)*
- ☐ 7 Subject withdrew consent *(please specify below)*
- ☐ 8 Lost to follow-up
- ☐ 9 Administrative problems *(please specify below)*
- ☐ 10 Death *(please complete Serious adverse events pages and record adverse events leading to death on the Adverse events page)*

Study evaluation completion

Primary reason for study evaluation completion:

- ☐ 7 Subject withdrew consent *(please specify below)*
- ☐ 8 Lost to follow-up
- ☐ 9 Administrative problems *(please specify below)*
- ☐ 10 Death *(please complete Serious adverse events pages and record adverse events leading to death on the Adverse events page)*

The most convenient CRF layout for reading and recognition will be the one with all page margins being the same within the document, all tick-boxes being right-aligned, and the text corresponding to each tick-box not wrapped to the line below. All the tick-boxes related to the same CRF section or question are on the same page as shown in the picture below:

Histological Grade of Cancer at Study Entry	Well differentiated ☐
	Moderately differentiated ☐
	Poorly differentiated ☐
	Undifferentiated ☐
	Unknown ☐
AJCC TNM Stage at Study Entry	Stage II ☐
	Stage IIA ☐
	Stage IIIB ☐
	Stage IIIC ☐
	Stage IVA ☐
	Stage IVB ☐

Summarizing the limitations:

- The CRF document must have text that can be highlighted or found;
- Only CRF fields that contain tick-boxes can be analyzed;
- The tick-boxes and their corresponding text must be left or right aligned.

The desired but not required CRF document layout, in addition to limitations:

- The text corresponding to each tick-box should not be wrapped;
- All tick-boxes from one section or question should be on a single page;
- The page margins should be the same within the whole document.

PREPARATION

A PDF document might be read and processed by such powerful tools as Acrobat Reader® or Fine Reader®. However they are not necessary and this paper describes the way how to process a PDF file using the software that would very likely be installed for any SAS programmer.

REQUIRED SOFTWARE

For the successful import and processing of the data, you will need to have installed:

- SAS® version 9.2 or higher;
- R version 3.0.1 or higher;
- R studio® version 1.2.5001 or higher for convenient work with the R code;

SET-UP R AND READ THE CRF DATA

This topic describes the path from the CRF document to SAS datasets, each of them contains CRF section headers with all the corresponding tick-boxes. The first step is to read the CRF document page by page, processing one page at a time. This option has not been implemented in SAS yet, but the *pdftools* R package comes to the rescue. The main R function used in the package is *pdf_data*. It returns R data frame with all the recognized text values in a page, including their physical position: i.e. width, height, and (x, y) position, starting from the top left corner of the page.
All the R code that is necessary for parsing the CRF document by page and converting each page to the CSV format is relatively simple and is shown below:

```
install.packages("callr");
install.packages("pdftools");

library(pdftools);
library(callr);

pdf_path <- "<full path to CRF in PDF format>";
total_pages <- <number of CRF pages>;

for (i in 2:total_pages){
      pdfdata <- r(function(x) pdftools::pdf_data(x), args = list(pdf_path))[[i]];
      path <- paste("<path to output CRF pages in CSV format>", "page", i-1, ".csv",
      sep = "");
      print(paste("Page", i, "of", total_pages, "was output into file:", path, sep =
      " "));
      write.csv(pdfdata, path, row.names = FALSE);
}
```

All is needed to set up is the number of pages to be read from the CRF document, the path to the document and the path where the CRF pages in the CSV format will be output. A note should be made that folders in the path variable definitions must be split with // symbol to run correctly. The CSV file format was chosen due to the ease of transferring data in this format to SAS using PROC IMPORT.

From now on begins the most interesting and the most difficult part – searching for tick-boxes and finding them a pair among the SDTM CDISC Submission values (hereinafter “SDTM terms”).

PROCESSING OF THE CRF DATA

The CRF pages are processed in SAS one by one. The algorithm contains a set of macros that are responsible for each step of the processing. The main macro call includes all the necessary parameters to control the whole processing starting from import of the CRF pages into the CSV format, and ending with the SAS code creation. The

main macro parameters are shown below:

```
%auto_mapping(func_outlib_name        = <area.ds_name.user_name for function>,
              sdtm_term_xlsfile_path = "<SDTM Controlled Terminology file path>",
              sdtm_term_sheet_name   = "<SDTM Controlled Terminology Excel sheet>",
              crf_page_path          = "<Path to CRF pages in CSV>",
              print_test_pagenum     = <enter any number to print test page>,
              keep_or_drop_text      = "<word pattern for keep or drop>",
              keep_or_drop_mode      = <KEEP | DROP: mode for a pattern>,
              start_pagenum          = <start page number>,
              end_pagenum            = <end page number>,
              tbox_line_end          = <tick-box word end position>,
              corr_tbox_end          = <tick-box word end pos. correction>,
              del_prev_page_fl       = <Y | N: delete prev. file with code?>,
              output_code_path       = "<Path to output files with code>",
              show_temp_datasets     = <Y | N: show/hide temp. datasets?> );
```

IMPORTING THE CRF DATA INTO SAS

The import of the CRF pages into the CSV format is performed by a simple PROC IMPORT call. The only thing needed is to set the path where the CSV files are stored:

```
%auto_mapping(…
              crf_page_path          = "<Path to CRF pages in CSV>",
              … );
```

EXTRACTING CRF TICK-BOXES

The structure of the imported SAS dataset, that corresponds to a CRF page is simple. Each observation includes a word that can be found in the CRF page and 4 numeric variables that are responsible for the physical position of this word. Two variables contain the distance from the top left corner of the page vertically and horizontally, and the remaining two contain the height and width of the word.

These parameters are sufficient to determine the start and end position of each word on the page vertically and horizontally, as well as to estimate the length of the space between words.

	width	height	x	y	text
21	16	9	90	169	Age
22	16	9	90	184	Age
23	15	9	109	184	unit
24	15	9	90	199	Sex
25	32	9	489	184	YEARS
26	20	9	485	199	Male
27	29	9	476	215	Female
28	36	9	90	234	Ethnicity
29	35	9	431	234	Hispanic
30	8	9	469	234	or
31	26	9	479	234	Latino
32	15	9	413	250	Not
33	35	9	431	250	Hispanic
34	8	9	469	250	or
35	26	9	479	250	Latino
36	15	9	455	266	Not
37	33	9	472	266	reported
38	39	9	466	282	Unknown

(1) (2) (3) (4)

Day month and year of birth

Age (x;y) word start coordinates (1)

Age unit text (2) YEARS

Sex height (3) Male

Female

Ethnicity Hispanic or Latino

width (4) Not Hispanic or Latino

Not reported

Unknown

Estimating the space length is a pretty important step in the text processing. Why an estimate, and not an exact calculation? The fact is that words in the CRF document might lie in different distances from each other. This can be seen if one would compare the end numeric position of a word with the start numeric position of the next word. The numeric value of the distance between words that occurs most frequently will be selected as the space estimation, i.e. the mode of all the distances. The space estimation is automatic and is performed separately for each CRF page. In the example shown below the mode of all the distances is equal to 3.

	text	v_startpos	h_endpos	h_startpos	lh_endpos	diff
18	Enter	137	126	103	235	.
19	Protocol	137	166	130	126	4
20	version	137	201	169	166	3
21	4	137	209	204	201	3
22	details	137	241	213	209	4
23	only	137	262	244	241	3
24	after	137	285	265	262	3
25	approval	137	326	288	285	3
26	is	137	336	329	326	3
27	received	137	376	339	336	3
28	Did	156	118	103	376	.
29	the	156	134	121	118	3
30	subject	156	169	138	134	4
31	consent	156	207	172	169	3
32	to	156	218	210	207	3
33	protocol	156	256	221	218	3
34	version	156	291	259	256	3
35	4?	156	305	294	291	3

Space estimation is followed by a more complex task. How to pick the tick-box text values and their section headers from the rest of the data on the page? The point is to find out if the location of the tick-boxes on the CRF page has any regularity. This regularity can easily be seen visually on the page: the tick-box values must end precisely one under another. Therefore, it makes sense to expect that if one takes all numeric end positions of the tick-box text values, they should be equal. This numeric end position is expected to be unique for each CRF document. It will be used in the algorithm for all CRF pages to correctly identify and read the tick-box text values.

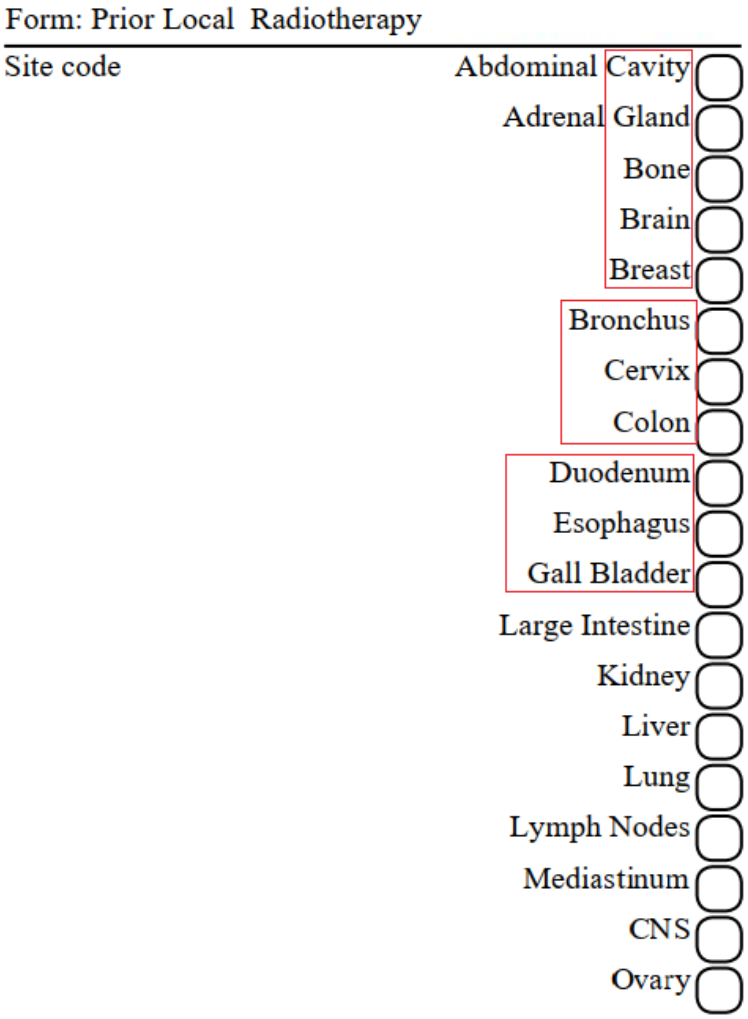
At this step the programmer's contribution is needed. A test run, involving a single CRF page, is required to find the end position before processing of all the pages. Simply select the page number and pass into the macro:

```
%auto_mapping(…
              print_test_pagenum     = <enter any number to print test page>,
              … );
```

The result contains the words, found on a selected page and the corresponding end positions. The programmer needs to look at the CRF page, and then to find out which words are the last within every tick-box. The end position value, that is required for the algorithm to work correctly is the one that is common for the last words found. In the picture shown below, the last words within every tick-box text value have the end position value equal to 505. This value will be used in the `tbox_line_end` parameter of the macro.

Text fields that lie on the left 2/3rd part of the page 62

CRF text value	Horizontal end position of text	Horizontal start position of text
Large	468	445
Small	468	445
Gall	471	454
Urinary	471	440
Abdominal	475	431
Soft	476	460
Lymph	477	449
Adrenal	478	446
Salpinx/Fallopian	482	411
62	493	483
of	504	496
Cavity	505	479
Gland	505	481
Bone	505	484
Brain	505	483
Breast	505	480
Bronchus	505	467
Cervix	505	478
Colon	505	481
Duodenum	505	461
Esophagus	505	462
Bladder	505	474



There is a certain freedom in choosing a page, but I would recommend to select a page with many tick-boxes - this will provide a more accurate selection of the coordinate. The other recommendation would be to make several test runs for different pages.

The test runs show that end positions of the words may differ by 1-3 units within the same page. The macro can apply correction of these values. If this positive number (usually small) is specified, then all the values of the interval (end position ± correction value) will be treated as a valid end position of a word. Here are the macro parameters that set these options:

```
%auto_mapping(…
              tbox_line_end          = <tick-box word end position>,
              corr_tbox_end          = <tick-box word end pos. correction>,
              … );
```

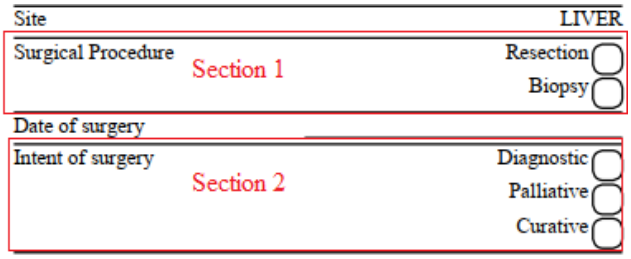
This explains the limitation that all the CRF tick-boxes must be left or right aligned. Please note that in this paper only right alignment of the tick-boxes is considered. However, it is not so important as the main idea is to find the regularity which does not depend on the direction of alignment.

It is very likely that a CRF page will not have any tick-boxes at all. Such pages will be read, but will not be analyzed. The algorithm contains a macro that stops the processing of a page and goes to the next one if the specific text is found. It can work vice versa i.e. can process only those pages that contain the specified text.

```
%auto_mapping(…
              keep_or_drop_text      = "<word pattern for keep or drop>",
              keep_or_drop_mode      = <KEEP | DROP: mode for a pattern>,
              … );
```

The result of the macro execution at this step will be the following dataset:

	section	linenum	crf_header	crf_term
1	1	1	Surgical Procedure	Resection
2	1	2	Surgical Procedure	Biopsy
3	2	1	Intent of surgery	Diagnostic
4	2	2	Intent of surgery	Palliative
5	2	3	Intent of surgery	Curative



MATCHING CRF AND SDTM DATA

The next task is more complex than the previous one. Now, every tick-box text value has to match a corresponding value in the CDISC SDTM Controlled Terminology (hereinafter "SDTM CT"). The SDTM CT is imported into the SAS dataset from an Excel file using the following parameters:

```
%auto_mapping(…
              sdtm_term_xlsfile_path = "<SDTM Controlled Terminology file path>",
              sdtm_term_sheet_name   = "<SDTM Controlled Terminology Excel sheet>",
              … );
```

It is regularly updated, so the corresponding changes might be done in the code to use the most up-to-date version.
Next, every CRF section must be associated with an SDTM Codelist Name value (hereinafter "SDTM section"). This requires some text search and matching process. In the algorithm described in the paper, this is implemented using a user-defined function, created by PROC FCMP. It is required to specify the name of an output data set to which compiled subroutines and functions will be output (for example, work.userfuncs.char).

```
%auto_mapping(func_outlib_name       = <area.ds_name.user_name for function>,
              … );
```

This function can determine whether a string is part of another string or vice versa, whether the strings are equal or not. In order to increase the accuracy of the matching results, a decision was made not to compare the strings character by character, as this would lead to a large number of false string matches.
At the end of this step there will be one-to-one correspondence between the CRF and SDTM sections. Some CRF sections will have no pair, as many of them are study specific. Such sections will be processed in a certain way, but those ones who got a pair will be processed initially. Now the task is to pair each CRF tick-box text value with a unique CDISC Submission value or CDISC Synonym(s) value. Visually, it is quite easy to do, however, the algorithm that performs the matching includes 4 steps. Let us take closer look at them.

Step 1: Find full matches among all of the combinations of CRF tick-box text values and CDISC Submission value or CDISC Synonym(s) value. These matches will be considered *trusted matches*, and will be output into a separate dataset.

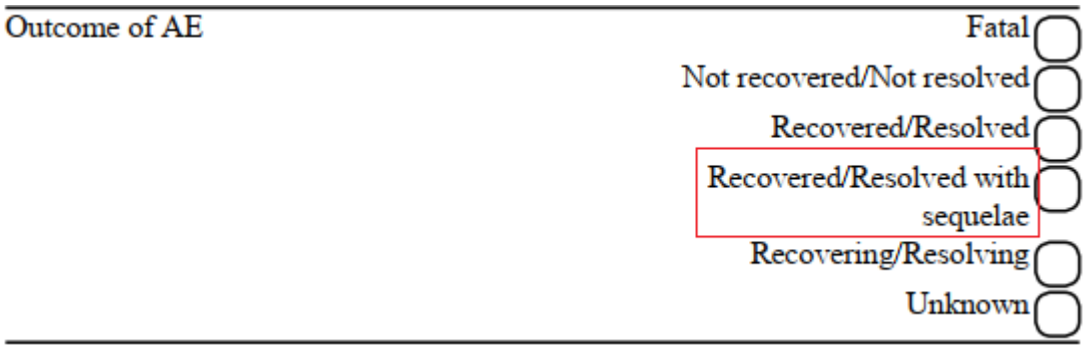
Step 2: There is another type of matches, when **both** the SDTM CDISC Submission value **and** the CDISC Synonym(s) value are the part of the CRF tick-box text value, and they are not equal. These matches are also considered *trusted matches.* They will be added to the matches already found during the previous step.

Step 3: The situations when **either** the SDTM CDISC Submission value **or** the CDISC Synonym(s) value is contained in the CRF tick-box text value are considered *partially-trusted matches*. They will be added to the dataset with trusted matches, created at the end of the previous step. As this type of matches is not 100% exact, a warning message will be shown next to them at the stage of the SAS code generation:

```
/*Warning: check this line in CRF and SDTM for consistency*/
```

This is done for the programmer to pay special attention to these matches at the step of SDTM mapping.

The last step is step 4: the type of match here is the opposite to the one in the 3rd step. Here the cases are processed when the CRF tick-box text value is a part of **either** SDTM CDISC Submission value **or** CDISC Synonym(s) value. Generally speaking, this type of match is considered the most unreliable, so it is not possible to use it in the mapping process. However, a situation may arise when the a CRF tick-box text value is wrapped to the next line:



In this case, an attempt is made to join the wrapped lines and compare the result to the SDTM CDISC Submission value or CDISC Synonym(s) value. If with any luck full matches appear, they will also be considered as *trusted matches*.

At the end of this process a dataset with CRF tick-box text value/SDTM CDISC Submission value pairs is generated. Some pairs will have a warning message to check the specific pair manually. Some CRF tick-box text values will not have a pair at all. In this case a CRF tick-box text value will be paired with itself in the upper case and a warning message will be shown next to this line at the stage of the SAS code generation:

```
/*Warning: corresponding value was not found in SDTM!*/
```

SAS CODE GENERATION

Finally, everything is ready to generate a SAS code, that can be directly copied to the program, and can be used during the SDTM mapping process. For each CRF page where tick-boxes were found, a separate RTF file is created in a folder, specified by the following parameter:

```
%auto_mapping(…
              output_code_path       = "<Path to output files with code>",
              … );
```

The generated code consists of a standard PROC FORMAT call. All the programmer needs to do at this stage is to specify the format name for each CRF section.

```
/*Code generated from page 122*/
proc format;
  value <insert_user_value_name>; /* Anatomical location of biopsy */
     "Adrenal gland" = "ADRENAL GLAND"
     "Ascites" = "ASCITES"/*Warning: corresponding value was not found in SDTM!*/
     "Bone" = "BONE"
     "Brain" = "BRAIN"
     "Breast" = "BREAST"
     "Bronchus" = "BRONCHUS"
     "Cervix" = "CERVIX"/*Warning: corresponding value was not found in SDTM!*/
     "Colon" = "COLON"
     "Duodenum" = "DUODENUM"
     "Esophagus" = "ESOPHAGUS"
     "Gall bladder" = "BLADDER"/*Warning: check this line in CRF and SDTM for
consistency*/;
run;
```

Special mention should be made of the case when a CRF section occupies two pages. This usually happens with long sections, like Dose Frequency. In this case the code will be generated for both the previous page and the combined previous and next pages. The macro allows to either keep or remove the file with the code for the previous page using the following parameter:

```
%auto_mapping(…
              del_prev_page_fl       = <Y | N: delete prev. file with code?>,
              … );
```

The RTF file with the SAS code for combined pages will contain the following message:
`/*Code generated from page <i-1> appended to page <i> */.`

The considered steps are repeated consequently for each CRF page. The macro allows to select the start page number and the end page number (usually it is 1 and the last CRF page number). It might also be useful to have a look at intermediate datasets for testing purposes. The corresponding macro parameters are shown below:

```
%auto_mapping(…
              start_pagenum          = <start page number>,
              end_pagenum            = <end page number>,
              …
              show_temp_datasets     = <Y | N: show/hide temp. datasets?>);
```

SDTM CHECKER

This is a simple macro that shares the logic with the algorithm described above. The purpose is to check if a variable was mapped correctly according to the SDTM CT:

```
%sdtm_check(func_outlib_name       = <lib.ds_name.user_name for function>,
            dsin_sdtm              = <SDTM dataset lib.name>,
            variable_list          = <space-delimited list of variables>,
            sdtm_term_xlsfile_path = "<SDTM Controlled Terminology file path>",
            sdtm_term_sheet_name   = "<SDTM Controlled Terminology Excel sheet>",
            report_file_path       = "<Path to output files with report>");
```

Some of the parameters should already be familiar, and the rest are quite easy to use. Simply specify the SDTM dataset name and the already mapped variable names – one or several. The macro compares each of the variable values to the SDTM CDISC Submission values in the SDTM CT and generates the report. The report files are created separately for each checked variable in the RTF format. Checking the mapping can produce three types of results:

- a match;
- no match while the list of SDTM CDISC Submission values is extensible;
- no match while the list of SDTM CDISC Submission values is **not** extensible;

An example of the report for SDTM.AE AEACN variable is shown below:

```
Results of checking for variable AEOUT (Outcome of Adverse Event) mapped
values against CDISC SDTM

Value:                            Check result:
 FATAL                             OK - mapping is correct
 NOT RECOVERED/NOT RESOLVED        OK - mapping is correct
 RECOVERED/RESOLVED                OK - mapping is correct
 RECOVERED/RESOLVED WITH SEQUELAE  OK - mapping is correct
 RECOVERING/RESOLVING              OK - mapping is correct
 UNKNOWN                           OK - mapping is correct
```

WHAT IS THE BENEFIT?

Comparison of the programming methods consist of two parts: speed of delivery and accuracy of the results. A typical SDTM mapping process contains looking at the CRF and finding the tick-boxes with pre-defined text values. The next step is to find out if these values have the corresponding standard terms in the CDISC SDTM Controlled Terminology. If they have, then the programmer needs to pair each of the CRF terms with an SDTM CDISC Submission value and then create a code to implement this pairing. Usually it is either a PROC FORMAT call or IF-THEN-ELSE blocks, so a number of formats have to be created manually. The described algorithm can carry out this pairing for all tick-boxes found in the CRF within an hour. This is an approximate time estimation, and it includes the setting all folders up, running R and SAS code, and several test runs.

Accuracy is definitely not the strongest part of the algorithm, though the test runs show that no data are being lost during the process. Obviously, the SDTM mapping performed manually is more precise in terms of pairing the CRF

and SDTM data. Though having the ready code even for the situations when the automatic pairing was unsuccessful will still make the manual matching faster and more convenient.

THE IMPLEMENTATION OF THE ALGORYTHM

The algorithm described in the paper is implemented via the set of macros. They are stored in the programs that can be found and downloaded using the following link:
https://github.com/aartemchuk/SDTM_MAPPING .
The program that controls the input parameters, shown in the `%auto_mapping` macro is main_macro.sas. The program for checking the SDTM mapping is called the sdtm_checker.sas. Both of these programs require the import_macros.sas program where all macros needed for the processing are stored. The programs are updated regularly to improve the speed and stability.

CONCLUSION

It might seem that the algorithm described in the paper is rather unreliable and has a number of assumptions. This is partly true: when it comes to text recognition and matching, it is difficult to achieve 100% reliability using the same strict approach. Accuracy can be gradually improved by complicating the algorithm, and processing more and more cases.
This task is not set by this paper: the true goal is to show that reading and processing PDF data is quite possible by the combined efforts of SAS and R, but the most important thing is to show by a particular example that it is possible to significantly speed up the SDTM mapping process. This idea, however, can be developed in the direction of the machine learning. Then the algorithm will be designed in such a way that CRF recognition and matching it to the SDTM data will become more accurate with each new CRF being read, but this is a completely different story.

REFERENCES

SAS Institute Inc. 2013. SAS/STAT® 13.1 User's Guide. Cary, NC: SAS Institute Inc.

Jeroen Ooms. Package 'pdftools'. March 10, 2019

CONTACT INFORMATION

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

Author Name: Andrii Artemchuk
Company: Intego Group LLC
Address: 43/2 Gagarin Avenue
City / Postcode: Kharkov 61001, Ukraine Work
Phone: +38 057 755 7020 ext. 2425
Fax: +38 057 728 30 35
Email: andrii.artemchuk@intego-group.com
Web: https://intego-group.com