

Automatic CRF Annotations Using Python

Lijun Chen, Jin Shi, Tong Zhao, LLX Solutions, LLC

ABSTRACT

The annotated CRF is a required component in the submission package sent to FDA. However, the CRF annotation is usually a monotonous and time-consuming manual process, and some existing automatic packages still have limitations. Taking the advantage of all the powerful functions Python can carry, we have developed a Python package to automatically annotate CRF pages. The package is designed to work on a blank CRF that is for a version update or that comes from a new study. The whole process consists of four steps, kicked off with optional information extraction from an old annotated CRF, followed by reading and mapping new CRF to provide input for the next step of annotating CRF, and lastly bookmarks can be optionally added. At a few check points, the Excel spreadsheets are used to allow manual controls to ensure accuracy. The current performance of the package is acceptable, and more explorations have been planned. This paper introduces some key points of the current stage of the package, and some relevant sample codes and examples will be provided in the paper.

INTRODUCTION

The annotation of blank Case Report Form (CRF) pages is the starting point of Standardized Data Tabulation Model (SDTM) production, and the annotated CRF (aCRF) file is required to be bundled with the SDTM datasets in the clinical data submission package to the Food and Drug Administration (FDA).

CRF annotation is usually a manual and time-consuming task performed using the Adobe Acrobat comment tool. Each annotation is created individually with text entered manually and settings adjusted specifically.

Statisticians and programmers have been struggling with this process and have come up with several ways to automate the work. The currently available packages or methods mostly use multiple software, and output Forms Data Format (FDF) files to be imported into blank CRFs. One of the published methods is to use the combination of Excel, SAS, SAS XML Mapper, and PDF editor (Adobe Acrobat) (Gu n.d.) (Noory Kim, Bhagyashree Shivakumar n.d.). However, Base SAS or SAS XML Mapper cannot extract the text contents with corresponding positions of the questions on each page, and therefore the annotation performance could be very poor for a brand new CRF or when there are dramatic changes in the new version of CRF from the previous one. Another method shared with public involves the use of Python besides Excel and PDF editor (Hema Muthukumar, Kobie O'Brian, Julie Stofel n.d.). One major limitation we see in this method is the requirement of Study Design Specification (SDS), which is not always available to statistical and programming teams. Therefore, it is of great importance to look for better ways to solve these problems.

Taking the advantage of all the powerful functions Python can carry, we have developed a Python package to automatically annotate CRF pages. Compared to the traditional ways or other automatic tools, our package has the following essential features:

- The package is developed to accommodate both scenarios, either when the blank CRF is for a CRF version update with an existing old aCRF of the study, or when the blank CRF is for a new study.
- The package utilizes only Python and Excel. The Python code can be run using batch files with a simple click, which is convenient for users without Python experience. The Excel files are generated from Python code, which provides manual check points for all team members.
- The input of the package is CRF only, and SDS is not a mandatory input.
- The package covers the whole process, including extracting annotations from the old aCRF, reading new questions from the new CRF, mapping annotations for the new questions, adding annotations to the new CRF, and generating bookmarks. Most parts are entirely automatic, with manual works required only at a few necessary check points.
- The package can capture questions and their corresponding mappings from the annotated CRF pages.

Using this function, we have established a growing mapping database in Excel format that stores the questions and their annotations in pairs. Each time the package reads in an aCRF, it expands the database with the new mappings. In addition, we have applied a string-matching method in the package. By comparing the strings of the new questions and the old questions, the package can find the best-matched old questions for the new questions and assign the annotations of the old questions to the new questions. These approaches make it possible to annotate a brand-new question automatically, which is the biggest breakthrough of our package.

WORKFLOW AND PROCESS

The whole process consists of the following four steps, and we have created four Python programs for each of the step.

1. Read and extract an old aCRF
2. Read and map a new CRF
3. Annotate CRF pages
4. Add bookmarks

As Figure 1 shows, the only essential input of the whole program is CRF files in PDF format. Excel spreadsheets will be generated after some programmatical executions for manual checks. Process marked in gray indicates that the encompassed steps are optional.

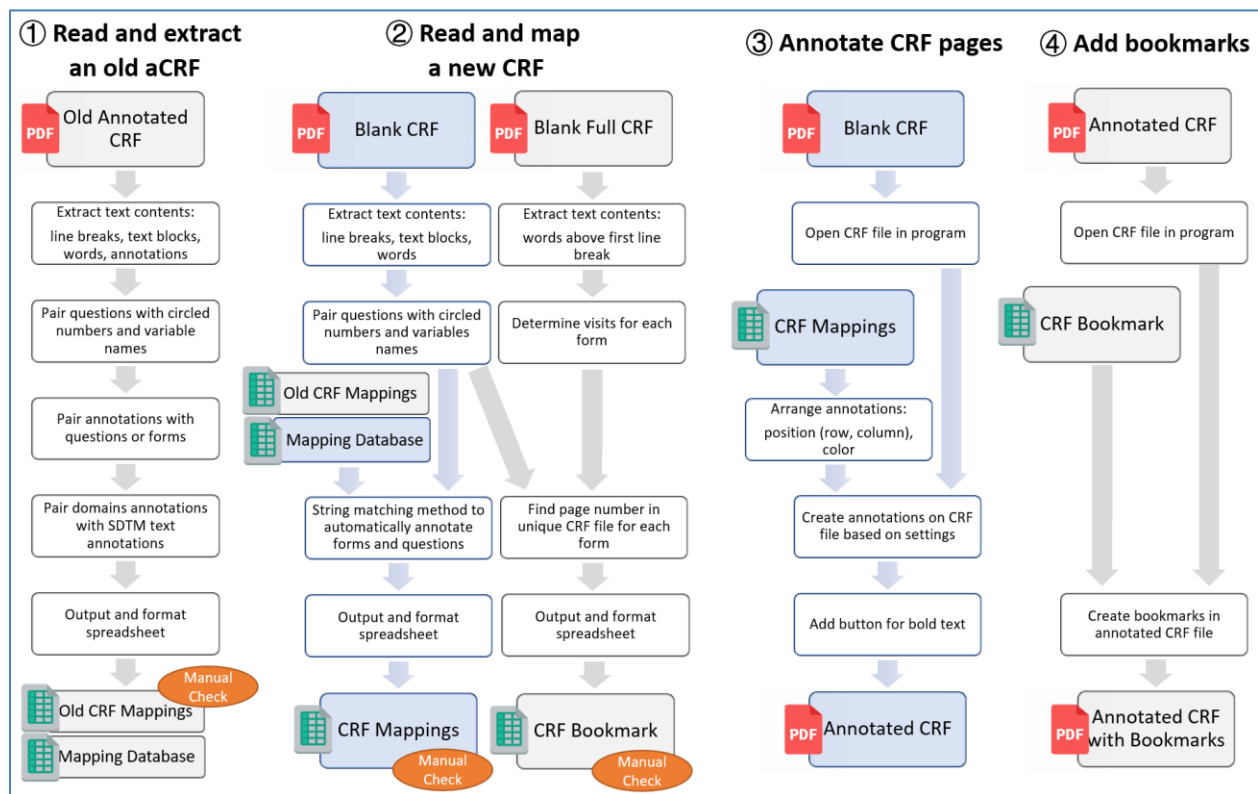


Figure 1 Workflow overview

1. READ AND EXTRACT AN OLD ACRF

When there is an old aCRF of the study, the first program can be used to read in the information of the old aCRF, including the texts, positions, and annotations of the questions. All the captured information will be reorganized by merging the form names and questions with circled numbers, variable names, and annotations, and will be exported as a well-formatted Excel spreadsheet (Figure 2) including variables of *page*, *form*, *question*, *circled_num*, *var_num*, *SDTM_Domain* and sequential *SDTM_Text*. We can manually check the spreadsheet to make sure the questions and annotations are stored correctly. Team members will review the spreadsheet together, and the completed file will be appended to the mapping database for future use.

When the CRF comes from a new study, and there is no available old mapping for the study, the first step can be skipped.

	A	B	C	D	E	J	K	L
1	page	form	question	circled_num	var_name	SDTM_Domain	SDTM_Text1	SDTM_Text2
2	1	VISIT	Form: VISIT					
3	1	VISIT	Visit Date	1	VISDAT	SV=Subject Visits	SVENDTC	SVSTDT
4	1	VISIT	Visit Not Done	2	VISSTAT	SV=Subject Visits	[NOT SUBMITTED]	
5	1	VISIT	Specify Reason if Visit Not Done	3	VISREAS	SV=Subject Visits	[NOT SUBMITTED]	
6	3	Demographics	Form: Demographics					
7	3	Demographics	Date that subject started the OLE study	1	RFICDAT	DS=Disposition	DSCAT=OTHER EVENT	
8	3	Demographics	Date subject took the first dose of [] in the OLE	2	EXSDAT	DS=Disposition	DSSDTC	DSTERM/DSDECOD=START OPEN LABEL STUDY
9	3	Demographics	Sex	3	SEX	DM=Demographics	TREATMENT	DSSDTC
10	3	Demographics	If Female, Childbearing Status	4	CHDBER	DM=Demographics	SEX	
11	5	Physical Examination YN?	Form: Physical Examination YN?			PE=Physical Examination	CBP in SUPPDM	
12	5	Physical Examination YN?	Was the Physical Examination Performed?	1	PEYN		[NOT SUBMITTED]	
13	7	Physical Examination	Form: Physical Examination			PE=Physical Examination	PECAT	
14	7	Physical Examination	Describe only the Abnormalities or indicate Not Done for each of the Body Systems					
15	7	Physical Examination	Check if All Body Systems are Normal	2	PESTAT	PE=Physical Examination	PEORRES=NORMAL where PETESTCD=INTP	
16	7	Physical Examination	Body System examined with Abnormal Results	3	PETEST	PE=Physical Examination	PETEST	
17	7	Physical Examination	Specify abnormality	4	PEDESC	PE=Physical Examination	PEORRES	
18	7	Physical Examination	Record any clinically significant abnormal findings on the Medical History or Adverse Events form as appropriate					
19	9	Neurological Examination YN?	Form: Neurological Examination YN?			NV=Nervous System Findings	[NOT SUBMITTED]	
20	9	Neurological Examination YN?	Was the Neurological Examination Performed?	1	PEYN			
21	11	Neurological Examination	Form: Neurological Examination					
22	11	Neurological Examination	Describe only the Abnormalities or indicate Not Done for each of the Neurological Systems					
23	11	Neurological Examination	Check if All Neurological Systems are Normal	2	PESTAT	NV=Nervous System Findings	NVORRES=NORMAL where NVTESTCD=INTP	
24	11	Neurological Examination	Neurological System examined with Abnormal Results	3	PETEST1	NV=Nervous System Findings	NVTEST	
25	11	Neurological Examination	Specify abnormality	4	PEDESC	NV=Nervous System Findings	NVORRES	
26	11	Neurological Examination	Record any clinically significant abnormal findings on the Medical History or Adverse Events form as appropriate					
27	13	Vital Signs YN?	Form: Vital Signs YN?			VS=Vital Signs	[NOT SUBMITTED]	
28	13	Vital Signs YN?	Were vital signs measured?	1	VSPERF			
29	15	Vital Signs	Form: Vital Signs					
30	15	Vital Signs	Section - Vital Signs (Semi-Supine) : Record average of vital signs collected in semi-supine position after patient has rested comfortably for at least 5			VS=Vital Signs	VSPOS	
31	15	Vital Signs	Systolic Blood Pressure	2	VSTESTSB	VS=Vital Signs	VSORRES=VSORRESU when VSTESTCD=SYBP	VSCLSIG in SUPPV
32	15	Vital Signs	Specify if abnormal	3	VSSBAIN	VS=Vital Signs	VSINTP in SUPPV	VSCLSIG in SUPPV
33	15	Vital Signs	Diastolic Blood Pressure	4	VSTESTDB	VS=Vital Signs	VSORRES=VSORRESU when VSTESTCD=DIABP	
34	15	Vital Signs	Specify if abnormal	5	VSSBAIN	VS=Vital Signs	VSINTP in SUPPV	VSCLSIG in SUPPV

Figure 2 Excel file of old CRF mapping

2. READ AND MAP A NEW CRF

The second program is the main part of the package. It reads in the blank CRF and maps the SDTM variables to the questions.

The program will extract the text contents from the blank CRF and arrange the information by forms and questions with their corresponding positions, circled numbers, and variables.

In the case of a CRF version update where there is an existing mapping file finalized in the first step, the program will first apply mappings from the old aCRF to questions that remain the same in the new CRF and then automatically search for appropriate mappings from the database for newly detected questions.

In the other scenario, when it is a new study, only the mappings from the database will be used.

Our current method to automatically impute annotations for the questions is to find the old questions that have the highest similarities with each of the new questions using a built-in string-matching package in Python. Annotations of these old questions will be merged with new questions as the annotations for the new questions. The mappings generated by the program will be output in another formatted Excel spreadsheet (Figure 3). Here is the second manual check point. All the group members can simultaneously work on the Excel file to adjust the mappings as needed.

#	A	B	C	D	E	J	K	L	M
1	page	form	question	circled_num	var_name	NEW	SDTM_Domain	SDTM_Text1	SDTM_Text2
2	1	Subject	Form: Subject			X			
3	1	Subject	Enter Double Blind Study Unique Subject ID	1	PUSUBIID	X			
4	3	VISIT	Form: VISIT						
5	3	VISIT	Visit Date	1	VISDAT	SV=Subject Visits		SVENDTC	SVSTDTCT
6	3	VISIT	Visit Not Done	2	VISSTAT	SV=Subject Visits		[NOT SUBMITTED]	
7	3	VISIT	Specify Reason if Visit Not Done	3	VISREAS	SV=Subject Visits		[NOT SUBMITTED]	
8	5	Demographics	Form: Demographics						
9	5	Demographics	Date that subject started the OLE study	1	RFICDAT	DS=Disposition		DSCAT=OTHER EVENT	
10	5	Demographics	Date subject took the first dose of [] in the OLE	2	EXSTDAT	DS=Disposition		DSSTDTCT	DSTERM/DSDECOD=START OPEN LABEL STUDY
11	5	Demographics	Sex	3	SEX	DM=Demographics		TREATMENT	DSSTDTCT
12	5	Demographics	If female, Childbearing Status	4	CHDBER	DM=Demographics		CBP in SUPPDM	
13	7	Physical Examination YN?	Form: Physical Examination YN?				PE=Physical Examination		
14	7	Physical Examination YN?	Was the Physical Examination Performed?	1	PEYN	PE=Physical Examination		[NOT SUBMITTED]	
15	9	Physical Examination	Form: Physical Examination				PE=Physical Examination	PECAT	
16	9	Physical Examination	Describe only the Abnormalities or indicate Not Done for each of the Body Systems						
17	9	Physical Examination	Check if All Body Systems are Normal	2	PESTAT	PE=Physical Examination		PEORRES=NORMAL where PETESTCD=INTP	
18	9	Physical Examination	Body System examined with Abnormal Results	3	PETEST	PE=Physical Examination		PETEST	
19	9	Physical Examination	Specify abnormality	4	PEDESC	PE=Physical Examination		PEORRES	
20	9	Physical Examination	Record any clinically significant abnormal findings on the Medical History or Adverse Events form as appropriate						
21	11	Neurological Examination YN?	Form: Neurological Examination YN?				NV=Nervous System Findings	[NOT SUBMITTED]	
22	11	Neurological Examination YN?	Was the Neurological Examination Performed?	1	PEYN	NV=Nervous System Findings		[NOT SUBMITTED]	
23	13	Neurological Examination	Form: Neurological Examination				NV=Nervous System Findings	[NOT SUBMITTED]	
24	13	Neurological Examination	Describe only the Abnormalities or indicate Not Done for each of the Neurological Systems						
25	13	Neurological Examination	Check if All Neurological Systems are Normal	2	PESTAT	NV=Nervous System Findings		NVORRES=NORMAL where NVTESTCD=INTP	
26	13	Neurological Examination	Neurological System examined with Abnormal Results	3	PETEST1	NV=Nervous System Findings		NVTEST	
27	13	Neurological Examination	Specify abnormality	4	PEDESC	NV=Nervous System Findings		NVORRES	
28	13	Neurological Examination	Record any clinically significant abnormal findings on the Medical History or Adverse Events form as appropriate						
29	15	Vital Signs YN?	Form: Vital Signs YN?				VS=Vital Signs	[NOT SUBMITTED]	
30	15	Vital Signs YN?	Were vital signs measured?	1	VSPERF	VS=Vital Signs		[NOT SUBMITTED]	
31	17	Vital Signs	Form: Vital Signs				VS=Vital Signs		
32	17	Vital Signs	Section - Vital Signs (Semi-Supine): Record average of vital signs collected in semi-supine position after patient has rested comfortably for at least 5				VS=Vital Signs	VSPOS	
33	17	Vital Signs	Systolic Blood Pressure	2	VSTESTSB	VS=Vital Signs		VSORRES/VSORRESU when VSTESTCD=SYSBP	
34	17	Vital Signs	Specify if abnormal	3	VSKBRBM	VS=Vital Signs		VSKMTR in SUPPDM	VSKCLSIG in SUPPDM

Figure 3 Excel file of new CRF mapping

To generate bookmarks automatically, the full casebook of blank CRF that contains all applicable forms at each of the visits will be used. This full CRF is also read in at this step, and only the titles of each form will be extracted. The program will find all the scheduled visits for each form as they appear in the title of the full CRF file as wells as the page number of the form in the unique CRF file, and will output the information to an Excel file (Figure 4) for manual check and potential adjustment. Considering the full CRF may not be available, this file is optional to run the program. This section within step 2 will be automatically skipped if no such file is provided.

#	A	B	L	M	N	O	P	Q	R	S	T	U
1	page	form	Visit 1 - Screening	Visit 2 - Baseline	Visit 3	Visit 4	Visit 5	Visit 6	Visit 7	Visit 8	Visit F1a	Visit F2a
2	1	Subject	X									
3	5	Demographics	X									
4	7	Physical Examination YN?	X	X	X	X	X	X	X	X	X	X
5	9	Physical Examination	X	X	X	X	X	X	X	X	X	X
6	11	Neurological Examination YN?	X	X	X	X	X	X	X	X	X	X
7	13	Neurological Examination	X	X	X	X	X	X	X	X	X	X
8	15	Vital Signs YN?	X	X	X	X	X	X	X	X	X	X
9	17	Vital Signs	X	X	X	X	X	X	X	X	X	X
10	20	Urine Pregnancy Test	X	X	X			X		X		X
11	22	Serum Pregnancy Test	X	X	X			X		X		X
12	24	12-Lead ECG YN?	X	X	X	X	X	X		X	X	X
13	26	12-Lead ECG	X	X	X	X	X	X		X	X	X
14	28	Concomitant Medications										
15	30	Adverse Events YN?										
16	32	Adverse Events										
17	35	Central Lab Sample Collection	X	X	X	X		X		X	X	X
18	37	ALIA Score	X	X	X					X		
19	40	PK Blood Samples			X	X	X	X		X	X	X
20	42	YN (Since Last Visit)		X	X	X	X	X	X	X	X	X
21	44	Seizure Rescue Medications per use		X	X	X	X	X	X	X	X	X
22	46	Eligibility Criteria	X		X	X	X	X	X	X	X	X
23	48	Disposition - Study Completion										
24	55	Quality of Life in Epilepsy			X					X		X
25	66	CSSRS Since Last Visit		X	X	X	X	X	X	X	X	X
26	77	Drug Dispensation										
27	81	Supply Deactivation				X	X	X	X	X	X	X
28	83	Unscheduled Dispensation										
29	85	Patient Supply Accountability Log										
30	93	Chemistry (Local Laboratory)										
31	97	Coagulation (Local Laboratory)										
32	99	Haematology (Local Laboratory)										
33	103	Urinalysis (Local Laboratory)										
34	106	Non-medication Treatment Log	X									
35	111	Dilated Ophthalmic Examination YN?										
36	113	Dilated Ophthalmic Examination										
37	115	Exposure - Dose Reduction				X	X	X	X	X	X	

Figure 4 Excel file for bookmarks

3. ANNOTATE CRF PAGES

After the Excel spreadsheet of CRF mapping is finalized, the third step is to add the annotations to the CRF pages, which is another crucial part that determines the success of the whole process. The program will read in the CRF to be annotated again, together with the Excel spreadsheet that contains the mappings.

We use strategies based on the space of the field to arrange the location of the annotations. It then places each of the domain annotations and variable annotations onto the CRF at a proper place, following the pre-assigned settings of font, color, and size. The annotated CRF (Figure 5) will be saved as a new PDF file.

DS=Disposition	DM=Demographics
Form: Demographics DSCAT=OTHER EVENT : Unique (PDF Generation Only)	
Date that subject started the OLE study DSSTDTC ①	
DSTERM/DSDECOD=START OPEN LABEL STUDY	
Date subject took the first dose of DSSTDTC in the OLE ②	
DSTERM/DSDECOD=START OPEN LABEL TREATMENT	
Sex SEX	Male <input type="radio"/> ③ Female <input type="radio"/>
If Female, Childbearing Status	
CBP in SUPPDM	Post Menopausal <input type="radio"/> ④ Child-Bearing Potential <input type="radio"/> Surgically Sterile <input type="radio"/>

Figure 5 Annotated CRF page

4. ADD BOOKMARKS

The last step is to add bookmarks to the annotated CRF, and it is optional. Using the completed Excel file generated for bookmarks, the program will create bookmarks at corresponding pages in the annotated CRF file (Figure 6) following the Clinical Data Interchange Standards Consortium (CDISC) guidelines.

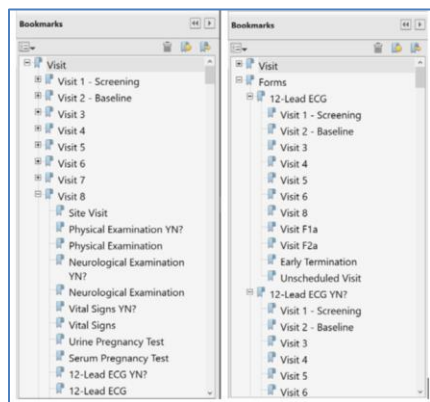


Figure 6 Bookmarks added to the annotated CRF

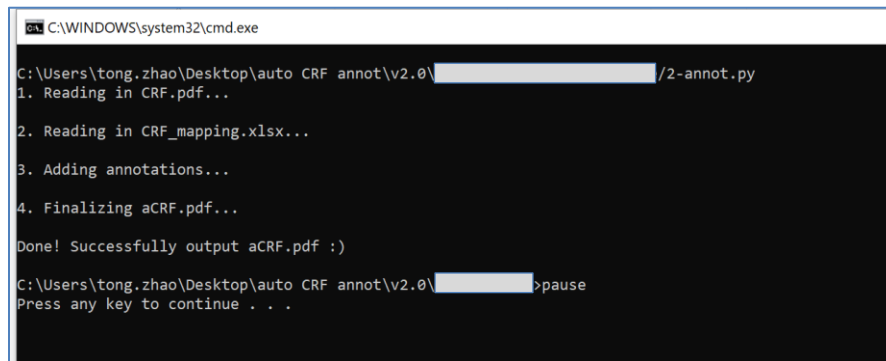
APPLICATION AND USAGE

SOFTWARE AND KEY PACKAGES

- Excel
- Python 3
 - PyMuPDF
 - PyPDF2

PROGRAM EXECUTION

All the Python programs are saved as Python (.py) files and can be executed using prepared batch files (.bat). The batch files contain commands to run the Python scripts through command prompt, and there will be messages showing the running progress (Figure 7).



```
C:\WINDOWS\system32\cmd.exe
C:\Users\tong.zhao\Desktop\auto CRF annot\v2.0\ /2-annot.py
1. Reading in CRF.pdf...
2. Reading in CRF_mapping.xlsx...
3. Adding annotations...
4. Finalizing aCRF.pdf...
Done! Successfully output aCRF.pdf :)
C:\Users\tong.zhao\Desktop\auto CRF annot\v2.0\ >pause
Press any key to continue . . .
```

Figure 7 Program running through command prompt

LIMITATIONS AND FUTURE WORKS

- Mapping performance can be improved.

The most challenging part of this automatic annotation package development is to find the accurate annotations for the questions using program. With our current string-matching method, the current package achieves an accuracy of roughly around 20% to 99% depending on the differences between the strings of the new questions and the existing questions from the database. We are still working on increasing the accuracy by exploring some other methods and expanding the database using more aCRF from various studies. The running time of the programs is acceptable, with an estimated **range of 10s to 30s for each of the step**.

- Tweaks in the programs may be required for every single study, and the current programs may be restricted to the use on CRF pages from specific database.

More comprehensive programs can be developed to accommodate CRF pages of various appearances.

- Most of the settings cannot be customized.

In the cases when there are special needs for attributes like text font and color, or file names and paths, these can only be modified in Python scripts now. It is preferable to create user interface in batch files to allow for more personal controls.

- There will always be new questions that cannot be mapped even though with a large database and a proper method.

The current idea to get annotations merely relies on existing mappings and previous experiences, but there can be new cases beyond recognition. Another way to solve this problem might include training a computer model to learn the SDTM implementation guide (SDTMIG) and controlled terminology (CT) documents and to provide appropriate annotations, just like the way humans handle the CRF annotation work.

CONCLUSION

Our packages are developed to fill the gap between the expectation of automatic CRF annotation tool and the existing packages in various aspects taking the advantage of the powerful functions Python can carry. Some breakthroughs have been made and more future works are planned to improve the performance and to accommodate more scenarios.

REFERENCES

Foundation Python Software. n.d. "Python 3.10.4 Documentation." <https://docs.python.org/3/index.html>.

Gu, Yating. n.d. "Auto-Annotate Case Report Forms with SAS® and SAS XML Mapper." PharmaSUG 2019.

Hema Muthukumar, Kobie O'Brian, Julie Stofel. n.d. "Automating CRF Annotation using Python." PharmaSUG 2020.

McKie, Jorj X. n.d. "PyMuPDF Documentation." <https://pymupdf.readthedocs.io/en/latest/index.html>.

Noory Kim, Bhagyashree Shivakumar. n.d. "Annotating CRFs More Efficiently." PharmaSUG 2021.

Phaseit, Inc. and Mathieu Fenniak. n.d. "PyPDF2 Documentation." <https://pythonhosted.org/PyPDF2/>.

The Pandas Development Team. n.d. "Pandas Documentation." <https://pandas.pydata.org/docs/>.

CONTACT INFORMATION

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

Lijun Chen
lijun.chen@lxsolutions.com

Tong Zhao
tong.zhao@lxsolutions.com

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

APPENDIX

SAMPLE CODES AND DETAILED LOGICS

Extract Text Contents

The code below shows how we capture the page number (*Page_Num*), text content (*text*), and position of text blocks (*x_left*, *x_right*, *y_top*, *y_bottom*) from the old aCRF file (*aCRF_old.pdf*) using package PyMuPDF (McKie n.d.) and convert the information into a pandas (The Pandas Development Team n.d.) DataFrame of “df”. Using the same logic, we repeat it for the line breaks, words, and annotations.

```
doc = fitz.open(path+'aCRF_old.pdf')
q_list= []
for i in range(doc.pageCount):
    page = doc[i]
    for x in page.get_text('blocks'):
        q_list.append({'Page_Num': i, 'text':x[4], "x_left":x[0], "x_right":x[2], "y_top":x[1], "y_bottom":x[3]})

df = pd.DataFrame(q_list)
```

Organize Extracted Information

The figure displays a screenshot of a CRF form with several fields and a table of extracted data. The form includes fields for 'Folder: Default', 'Form: Eligibility Criteria', 'DSSTDTC when DSDECOD=INFORMED CONSENT OBTAINED', 'Initial informed consent date (dd MMM yyyy)', 'RFICDTC', 'IEDTC', 'Select Initial Protocol Version', and 'PROTV in SUPPDS'. The table on the right lists the extracted data for each field, including the field name, data type, units, values, pre-filled values, and include field OID.

Field Name	Data Type	Units	Values	Pre-Filled Values	Include Field OID
① ICDAT	dd MMM yyyy				ICDAT
② IEICVER	\$15		2 = Version 2.0 3 = Version 3.0 4 = Version 4.0 6 = Version 6.0 7 = Version 7.0 8 = Version 8.0 9 = Version 9.0		IEICVER

Figure 8 Example from old annotated CRF page

Pair Questions with Circled Numbers and Variable Names

Vertical positions of line breaks can be used to bundle questions with their corresponding circled number. For instance, question “Initial informed consent date (dd MMM yyyy)” in Figure 8 falls within top line and second line from the top, and therefore will be linked to circled number ①. To identify if a certain digit text is attributable to a circled number, it is noted that while all questions are horizontally limited within ranges defined by line breaks, circled numbers have larger *x_right* value than that of line breaks. Next by looking at the annotations page, circled number ① has the field name “ICDAT”. Thus question “Initial informed consent date (dd MMM yyyy)” is linked to *var_name* “ICDAT”.

Pair Annotations with Questions or Forms

Domain annotations always show above the top line break and top text blocks, starting with “XX=”. With these characteristics, domain annotations can be easily extracted as *SDTM_Domain*. Each SDTM variable annotation, based on their midpoint vertical position, can be assigned to each question. For example, the midpoint of *y_top* and *y_bottom* of “DSSTDTC when DSDECOD=INFORMED CONSENT OBTAINED” is within the top two lines, and we then know it is the annotation of the first question.

Pair Domain Annotations with SDTM Text Annotations

As you may have noticed, there are three annotations (denoted as *SDTM_Text*) for “ICDAT”. How are we going to assign each SDTM text annotation correctly to each domain? Here we define the color distance. The RGB color representations can be obtained using the following codes.


```

li = []
for i in range(doc.pageCount):
    page = doc[i]
    for annot in page.annots(types=[fitz.PDF_ANNOT_FREE_TEXT]):
        li.append({'page': i, 'text':annot.info['content'],
                    "x_left":annot.rect[0] if annot.rect else np.nan,
                    "x_right":annot.rect[2] if annot.rect else np.nan,
                    "y_top":annot.rect[1] if annot.rect else np.nan,
                    "y_bottom":annot.rect[3] if annot.rect else np.nan,
                    "color_r":annot.colors['stroke'][0] if annot.colors['stroke'] else np.nan,
                    "color_g":annot.colors['stroke'][1] if annot.colors['stroke'] else np.nan,
                    "color_b":annot.colors['stroke'][2] if annot.colors['stroke'] else np.nan})

df_anno = pd.DataFrame(li)

```

As discussed above, we have been able to separate out domain annotations from SDTM text annotations. And color distance is then defined as:

$$d = (color_r_{domain} - color_r_{SDTM\ text})^2 + (color_g_{domain} - color_g_{SDTM\ text})^2 + (color_b_{domain} - color_b_{SDTM\ text})^2$$

For each SDTM text annotation, we will assign it to the domain with the smallest color distance.

String Matching Method

The following line of code shows how we do the automatic mapping after reading in CRF file and the training files to get the information into DataFrame of *CRF_map_df* and *train_df*. The *get_close_matches* (Foundation Python Software n.d.) function takes a single string as input, together with a list of reference strings to be compared with. The cutoff argument can be set by users to control the desired similarity between the string and the reference strings to be selected as a match. For the questions that have been included and annotated in the old CRF file, the cutoff is set to 1, so that we can guarantee that all the old mappings are retained 100% accurately in the new aCRF file. For the new questions, the cutoff is set to 0.7 in our program. This number is selected based on the test for the trade-off of relevance and return rate using our current mapping database. We are doing the string-matching using a concatenation of form name and question in case the questions with the same description appear in different forms and are for different domains. Then the program gets the corresponding annotations for each of the best matched strings from the reference files as the mappings for the questions.

```

match = difflib.get_close_matches(CRF_map_df.loc[qnum, 'form_question'], train_df['form_question'], n=1,
cutoff=cutoffset)

```

Create Annotations

The codes here illustrate how we use package PyMuPDF (McKie n.d.) to create annotations on the CRF file. The key function is *add_fretext_annot* from the package, and it requires input of the text content (*t*), text box (*r*) on selected location (*r1*, *r2*, *r3*, *r4*), and settings including color and alignment. Some specific controls of the appearance are only allowed using other functions of *set_border* and *update*.

```

t=(page_df.iloc[field_item, 9])
r=fitz.Rect(r1, r2, r3, r4)
annot = page.add_fretext_annot(r,t,fontsize=fontsize_text,rotate=0,fontname=fontname_text,text_color=black,
fill_color=fill_color_list[int(page_df.iloc[field_item,17])], align=fitz.TEXT_ALIGN_LEFT,)
annot.set_border(width=0.8)
annot.update(text_color=black,fill_color=fill_color_list[int(page_df.iloc[field_item,17])], border_color=black)

```

Usage of DataFrames and Loops

In the program, we store all the information in DataFrames or lists and access each of the contents using loops to add the annotations one by one. For example, the filling color of the text box is pre-assigned and stored as a list in Python, and the color to be used by each of the annotation is determined using the information in the DataFrame.

```

white=(1,1,1)
first_light_blue=(0.75,1,1)
second_light_yellow=(1,1,0.66)
third_light_green=(0.75,1,0.75)
fourth_purple=(0.66,0.75,1)
fifth_light_orange=(1,0.75,0.66)
fill_color_list=[white,first_light_blue,second_light_yellow,third_light_green,fourth_purple,fifth_light_orange]

```

Annotation Position

The most challenging part of creating annotations might be determining the position of the annotations. Each of the annotation text boxes is settled by four numbers that represent the minimum and maximum values on horizontal and vertical directions ($r1$, $r2$, $r3$, $r4$). Adding the domain annotations is easier, because they are always placed at the top of the page. For question annotations, each set of the numbers is set based on the sizes and the positions of other items.

When there are multiple annotations for a single question, we first left align the annotations vertically based on the allowed space between the top line and bottom line of the question field, and then throw additional annotations to the right of the added annotations, as if columns are added. In most cases, the question field can accommodate two rows of annotations (Setting 1 in Figure 9).

The figure below shows the logic we use to determine the position for annotation #4 in Setting 1.

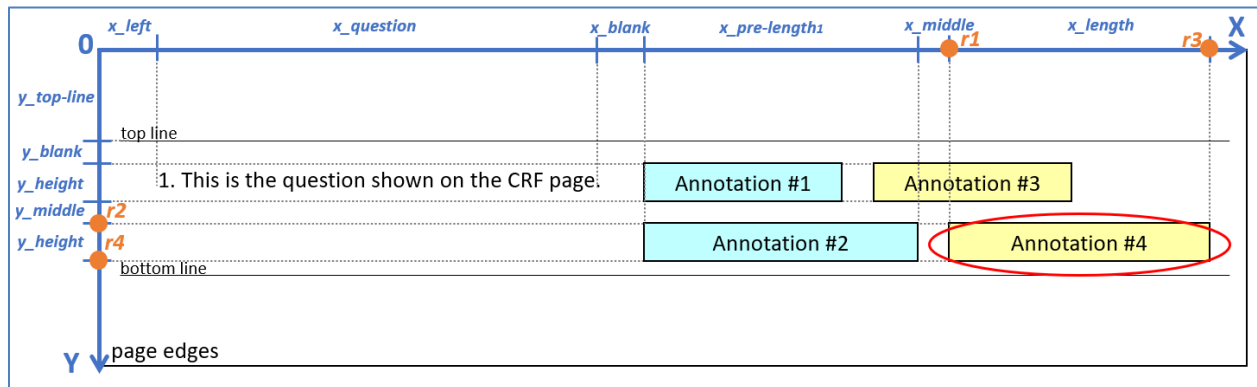


Figure 9 Annotation position example for Setting 1

In the case when the question field expands for more rows, we place the annotation right below the question to allow for more space in the horizontal direction, as shown in the last question annotation of Figure 5 (Setting 2 in Figure 10).

The following figure shows the logic how we set the position for annotation #1 in Setting 2.

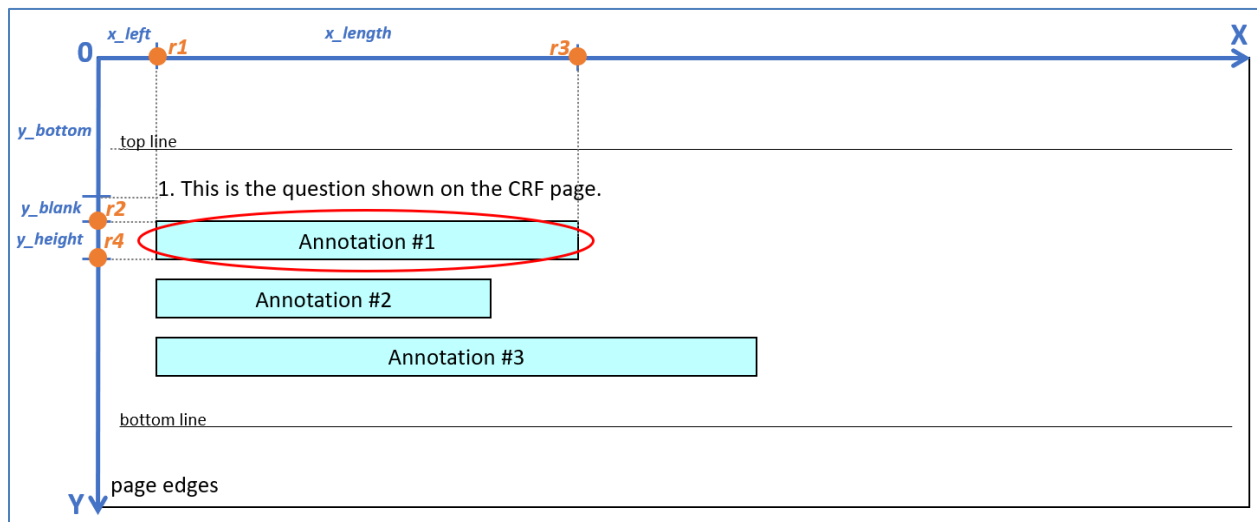


Figure 10 Annotation position example for Setting 2

The Minimum Horizontal Value ($r1$)

Setting 1:

$$r1 = x_{left} + x_{question} + x_{blank} + x_{prelength_1} + x_{prelength_2} + \dots + x_{prelength_{n-1}} + (n - 1) \times x_{middle}$$

Setting 2:

$$r1 = x_{left} + x_{prelength_1} + x_{prelength_2} + \dots + x_{prelength_{n-1}} + (n - 1) \times x_{middle}$$

In Setting 1, we consider the minimum horizontal value ($r1$) to be primarily affected by the start position of the question (x_{left}), the space that the question takes ($x_{question}$), and the gap we would like to have between the end of the question and annotations (x_{blank}). The question length ($x_{question}$) is influenced by the number of characters the question contains and if the question is in bold mode. The gap value (x_{blank}) can be set freely. In Setting 2, it basically starts at the left margin of the question (x_{left}).

When the annotation is not in the first column of the annotations, but in column n , to avoid overlapping with the annotations on the left, we need to further add the length of the previous annotations ($x_{pre-length}$) and the pre-set space in between (x_{middle}). The $x_{pre-length}$ here is also affected by the number of characters in the annotation text, and x_{middle} is a user-defined number.

As the example shown in Figure 9, in Setting 1, annotation #4 is in the second column ($n=2$), and the minimum horizontal value ($r1$) is:

$$r1 = x_{left} + x_{question} + x_{blank} + x_{prelength_1} + x_{middle}$$

As the example shown in Figure 10, in Setting 2, annotation #1 has the minimum horizontal value ($r1$) of:

$$r1 = x_{left}$$

The Minimum Vertical Value ($r2$)

Setting 1:

$$r2 = y_{topline} + y_{blank} + (m - 1) \times y_{height} + (m - 1) \times y_{middle}$$

Setting 2:

$$r2 = y_{bottom} + y_{blank} + (m - 1) \times y_{height} + (m - 1) \times y_{middle}$$

The minimum vertical value ($r2$) of the text box is primarily determined by the start position in the vertical direction and the preferred gap we choose from the start point to the annotation text box (y_{blank}). In Setting 1, the start point is the top line of the question filed ($y_{top-line}$), and in Setting 2, it is the bottom of the question (y_{bottom}).

If the annotation is not in the first row, but in row m , it starts under all the annotations above it, and we need to further append the height of all the annotations on top of it (y_{height}) and the extra space we would like to add between the annotations in the vertical direction (y_{middle}).

For the annotation #4 in Setting 1 shown in Figure 9, the minimum vertical value ($r2$) is:

$$r2 = y_{topline} + y_{blank} + y_{height} + y_{middle}$$

For annotation #1 in Setting 2 shown in Figure 10, the minimum vertical value ($r2$) is:

$$r2 = y_{bottom} + y_{blank}$$

The Maximum Horizontal Value ($r3$)

$$r3 = r1 + x_{length}$$

The maximum horizontal value ($r3$) is the minimum horizontal value ($r1$) plus the length of the annotation itself that is influenced by the number of characters of the text.

The Maximum Vertical Value (*r4*)

$$r4 = r2 + y_height$$

The maximum vertical value (*r4*) is the minimum vertical value (*r2*) plus the height of the annotation that is set by the user.

Bold Text in Annotations

In order to set the annotations for the domain names as bold, because this function is no longer supported by the package, a JavaScript can be executed in Python to add a button for the change in style from normal to bold. The code below particularly works on the annotations with certain attributes of text size of 13 and contents contains “=” and sets the button to clear after the click.

```
jscript = """
var annt = this.getAnnots();
annt.forEach(function (item, index) {
    try {
        var span = item.richContents;
        var size = item.textSize;
        var cont = item.contents;
        if (size == 13 & cont.indexOf("=") != -1) {
            span.forEach(function (it, dx) {
                it.fontWeight = 800;
            })
        }
        item.richContents = span;
    } catch (err) {}
});
var butt = this.getField("Bold");
butt.display = display.hidden;
this.flattenPages(nStart = 0, nNonPrint = 2);
"""
```

Create Bookmarks

We use Python package PyPDF2 (Phaseit, Inc. and Mathieu Fenniak n.d.) to add bookmarks to the PDF file. As it is shown in the following code, the *addBookmark* function takes the input of bookmark title and page number. The information is stored in DataFrame and is accessed by a loop. The parent argument allows a multi-level bookmark structure.

```
bm2 = file.addBookmark(title = str(bm_df.columns[visitnum]), pagenum = int(bm_visit_df.iloc[0, 0]), parent = bml)
```

Format Excel Files

ExcelWriter in Pandas (The Pandas Development Team n.d.) is a powerful tool to adjust the format of the excel file while writing DataFrame objects into excel sheets. The below sample code shows all the format settings we are applying for the CRF mapping spreadsheet generated in step 2.

```
unhidden_col_fm = workbook.add_format({'text_wrap': True})
form_row_fm = workbook.add_format({'bg_color': '#E2EFDA'})
title_row_fm = workbook.add_format({'top': True, 'bg_color': '#E2EFDA'})
header_fm = workbook.add_format({'bold': True, 'valign': 'center', 'bg_color': '#A9D08E', 'text_wrap': True})
footer_fm = workbook.add_format({'bottom': True})
alert_header_fm = workbook.add_format({'bold': True, 'valign': 'center', 'bg_color': '#FABF8F', 'text_wrap': True})
```

Run Programs using Batch Files

The command in each of the batch files is simply two lines. The code below is the command in the first batch file to run the first python program. The *python* command executes Python application and run the program, and the *pause* command prevents the window from closing automatically. All the messages are generated by *print* command written within the Python programs.

```
python source/0-extractCRF.py
pause
```