PharmaSUG China 2026 - Paper PO-135

Development of an Efficient Automated Tool for Extracting Variables from AnnotatedCase Report Forms (aCRFs) and Mapping to SDTM Specification Files

Ziyu Li, CSPC

ABSTRACT

In the process of clinical data standardization, creating and maintaining Study Data Tabulation Model (SDTM) specification files is a crucial yet often manual task, particularly requiring the accurate extraction of variables and their page information from annotated Case Report Forms (aCRFs).

This paper introduces an open-source, automated tool developed in Python to address this pain point. The tool parses aCRF PDF files containing interactive annotations, identifies standardized and custom variables, and automatically updates the corresponding Excel-based SDTM specification files. This includes supplementing missing variables and domains, and adding precise CRF page references.

Through batch processing and intelligent matching, the tool significantly reduces manual effort—tasks that previously required hours of manual verification and entry can now be completed in approximately 5 minutes for page mapping and variable/domain completion—while improving the accuracy and consistency of specification file creation.

This article details the tool's design architecture, core algorithms, application examples, and the efficiency gains achieved.

INTRODUCTION

In clinical trial data management, SDTM is the standard data model submitted to regulatory agencies. As the core document defining dataset structure, variable attributes, sources, and mapping rules, the quality of SDTM Specification directly affects the efficiency of subsequent programming and data submission. Traditionally, Spec developers manually read annotation boxes in the aCRF PDF and fill in the Spec template one by one. This process is not only time-consuming but also prone to omissions and errors due to carelessness.

As clinical trials become increasingly complex, aCRF pages may number in the hundreds and variables in the thousands, making it nearly impossible to guarantee complete accuracy through manual processing alone. Therefore, developing a tool that can automatically parse aCRF annotations and populate the Spec has significant practical value. The Python tool introduced in this paper is fully open-source, customizable to project needs, and especially suitable for small-to-medium CROs or pharmaceutical companies.

METHODS AND DESIGN

OVERALL ARCHITECTURE

This tool is developed in Python 3, relying on libraries such as PyPDF2, fitz (PyMuPDF), openpyxl, xlwings, and pandas, with a GUI built using Tkinter. The workflow consists of three main phases:

Annotation Extraction: Read all annotation box contents from the aCRF PDF, classify and deduplicate them.

Spec Population: Based on the selected SDTM template version, write extracted variable names and corresponding CRF page numbers into the appropriate sheets of the Spec Excel file.

Post-processing: Includes SUPP variable label generation, missing variable completion, custom domain handling, formatting, etc.

ANNOTATION EXTRACTION ALGORITHM

Annotations in PDFs are typically stored as /Annots objects, each containing a /Contents field. The tool iterates through every page and applies regular expression matching to each annotation content, handling the following cases.

Standard Variables: e.g., LBORRES, directly extract uppercase letter combinations.

SUPP Variables: Formats such as SUPPAEQVAL=Y or QVAL in SUPPAE; need to parse the parent domain and variable name.

Special Annotations: e.g., For Annotations see page X; jump to the specified page to continue extraction.

Filtering Irrelevant Items: Exclude system variables like STUDYID and non-variable text such as NOT DONE.

To improve accuracy, the tool prioritizes finding uppercase words to the left of = and excludes commonly misidentified words. For duplicate variables, all page numbers are retained, deduplicated, and sorted.

```
pypdf_doc = PdfFileReader(open(pdfpath+'/'+pdfname, "rb"))
annotslist = []
var_page1 = {}
var_page = {}
allspecvar = []
pages = pypdf_doc.getNumPages()
for i in range(pages):
    pypdf_page = pypdf_doc.getPage(i)
    if '/Annots' in pypdf_page:
        for annot in pypdf_page['/Annots']:
            try:
                page=i+1

                if re.match('SUPP[A-Z]{2}[0-9a-zA-Z\s\_\-\.]+=[0-9a-zA-Z\s\_\-\.]',annot.getObject()['/Contents']) != None:
                    var = annot.getObject()['/Contents'].strip()

                elif re.match('[A-Z0-9]+ in SUPP[A-Z]{2}',annot.getObject()['/Contents']) != None:
                    var = annot.getObject()['/Contents'].strip()

                elif re.findall(r'\b[A-Z]+[\s]*?[A-Z]+\b', annot.getObject()['/Contents']) != None:
                    uppercase_words = re.findall(r'\b[A-Z]+[\s]*?[A-Z]+\b', annot.getObject()['/Contents'])
                    filtered_words = [word for word in uppercase_words if not re.search(r'(?:= |=)' + word, annot.getObject()['/Contents'])]
                    for filtered_word in filtered_words:
                        var = filtered_word
                        if var in var_page1.keys():
                            var_page1[var].append(str(page))
                        else:
                            var_pagea = {var:[str(page)]}
                            var_page1.update(var_pagea)
                        annotslist.append(var)
                else:
                    var = annot.getObject()['/Contents'].strip()

                annotslist.append(var)

                if var in var_page1.keys():
                    var_page1[var].append(str(page))
                else:
                    var_pagea = {var:[str(page)]}
                    var_page1.update(var_pagea)

                if re.match('For[\s]Annotations[\s]see[\s]page[\s]([\d]+)',annot.getObject()['/Contents']) != None:
                    pagerepeat = re.match('For[\s]Annotations[\s]see[\s]page[\s]([\d]+)',annot.getObject()['/Contents']).group(1)

                    pypdf_page = pypdf_doc.getPage(int(pagerepeat)-1)
                    if '/Annots' in pypdf_page:
                        for annot in pypdf_page['/Annots']:
                            try:
                                if re.match('SUPP[A-Z]{2}[0-9a-zA-Z\s\_\-\.]+=[0-9a-zA-Z\s\_\-\.]',annot.getObject()['/Contents']) != None:
                                    var = annot.getObject()['/Contents'].strip()

                                elif re.findall(r'\b[A-Z]+[\s]*?[A-Z]+\b', annot.getObject()['/Contents']) != None:
                                    uppercase_words = re.findall(r'\b[A-Z]+[\s]*?[A-Z]+\b', annot.getObject()['/Contents'])
                                    filtered_words = [word for word in uppercase_words if not re.search(r'(?:= |=)' + word, annot.getObject()['/Contents'])]
                                    for filtered_word in filtered_words:
                                        var = filtered_word
                                        if var in var_page1.keys():
                                            var_page1[var].append(str(page))
                                        else:
                                            var_pagea = {var:[str(page)]}
                                            var_page1.update(var_pagea)
                                        annotslist.append(var)
                                else:
                                    var = annot.getObject()['/Contents'].strip()

                                annotslist.append(var)
                                if var in var_page1.keys():
                                    var_page1[var].append(str(page))
                                else:
                                    var_pagea = {var:[str(page)]}
                                    var_page1.update(var_pagea)

                                var_page.update(var_page1)
                            except Exception as ex:

                                pass
                var_page.update(var_page1)
```

```
            except Exception as ex:

                pass

anduplist = list(set(annotslist))
anduplist.sort(key=annotslist.index)
try:
    anduplist.remove('STUDYID')
    del var_page['STUDYID']
    del var_page1['STUDYID']
except:
    pass
```

SPEC POPULATION STRATEGY

The tool supports three template versions (V3.2/V3.3/V3.4), which mainly differ in how Origin, Source, and Page columns are filled:

For existing variables, the tool checks whether the current values are correct. If correct, they are not overwritten (to avoid damaging existing manual modifications); if incorrect, they are highlighted in green. For coded variables such as XXTESTCD and XXDECOD, even if annotated in the CRF, Origin is forced to Assigned, consistent with industry practice.

```
nondomainlist = []
wb=openpyxl.load_workbook(directorypath+'/A'+excelname)
for domain in anduplist:
    if domain in domainlist:
        try:

            varlist = [cell.value for cell in wb[domain]['A']]
            indexlist = []

            for i in anduplist:
                if i in varlist:
                    indexlist.append(varlist.index(i))
                    allspecvar.append(i)

            sheet=wb[domain]
            fille=PatternFill("solid",fgColor="99CC00")
            for i in range(len(indexlist)):
                sheet.cell(indexlist[i]+1,1).fill=fille
                varpage = list(set(var_page[varlist[indexlist[i]]]))
                varpage = sorted([int(x) for x in varpage])
                varpage = [str(x) for x in varpage]

                if str1.get() == 'V3.2':
                    pattern = r'^[a-zA-Z]{2}(TESTCD|DECOD)$'
                    if re.match(pattern, sheet.cell(indexlist[i]+1,1).value):
                        if sheet.cell(indexlist[i]+1,6).value == "Assigned":
                            pass
                        else:
                            sheet.cell(indexlist[i]+1,6).value = "Assigned"
                            sheet.cell(indexlist[i]+1,6).fill=fille
                    else:
                        if sheet.cell(indexlist[i]+1,6).value == "CRF Page "+" ".join(varpage):
                            pass
                        else:
                            sheet.cell(indexlist[i]+1,6).value = "CRF Page "+" ".join(varpage)
                            sheet.cell(indexlist[i]+1,6).fill=fille

                elif str1.get() == 'V3.3' or str1.get() == 'V3.4':
                    pattern = r'^[a-zA-Z]{2}(TESTCD|DECOD)$'
                    if re.match(pattern, sheet.cell(indexlist[i]+1,1).value):
                        if sheet.cell(indexlist[i]+1,6).value == "Assigned":
                            pass
                        else:
                            sheet.cell(indexlist[i]+1,6).value = "Assigned"
                            sheet.cell(indexlist[i]+1,6).fill=fille
                            sheet.cell(indexlist[i]+1,7).value = "Sponsor"
                            sheet.cell(indexlist[i]+1,8).value = ""
                            sheet.cell(indexlist[i]+1,8).fill=fille
                    else:
                        if sheet.cell(indexlist[i]+1,8).value == " ".join(varpage):
                            sheet.cell(indexlist[i]+1,6).value = "Collected"
                            sheet.cell(indexlist[i]+1,6).alignment = Alignment(vertical="center")
                            sheet.cell(indexlist[i]+1,7).value = "Investigator"
                            sheet.cell(indexlist[i]+1,7).alignment = Alignment(vertical="center")
                        else:
                            sheet.cell(indexlist[i]+1,6).value = "Collected"
                            sheet.cell(indexlist[i]+1,6).alignment = Alignment(vertical="center")
                            sheet.cell(indexlist[i]+1,7).value = "Investigator"
                            sheet.cell(indexlist[i]+1,7).alignment = Alignment(vertical="center")
                            sheet.cell(indexlist[i]+1,8).value = " ".join(varpage)
                            sheet.cell(indexlist[i]+1,8).fill=fille

        except:
            nondomainlist.append(domain)
wb.save(directorypath+'/A'+excelname)
wb.close()
```

SUPP VARIABLE HANDLING

SUPP variables are special SDTM variables used to store supplementary information. The tool identifies them in two ways:

1. Directly matching the format SUPP<DOMAIN><VAR>=<VALUE>.

2. Matching the format <VAR> in SUPP<DOMAIN>.

After extraction, the tool inserts a new row in the corresponding SUPP sheet, filling in the variable name, data type (Char), length (200), and CRF page numbers. If the user chooses to generate labels, the tool uses the fitz library to extract text near the SUPP variable in the PDF as the label description, highlighted in yellow.

```
    suppdomlist = []
    wb_supp=openpyxl.load_workbook(directorypath+'/A'+excelname)
    for supp in anduplist:
        try:
            if re.match('SUPP[A-Z]{2}',supp) or re.match('[A-Z0-9]+ in SUPP[A-Z]{2}',supp) != None:
                if re.match('[A-Z0-9]+ in SUPP[A-Z]{2}',supp) != None:
                    suppdom = re.match('([A-Z0-9]+) in SUPP([A-Z]{2})',supp).group(2)
                    suppvar = re.match('([A-Z0-9]+) in SUPP([A-Z]{2})',supp).group(1)
                elif re.match('SUPP[A-Z]{2}',supp) != None:
                    suppdom = re.match('SUPP([A-Z]{2})([0-9a-zA-Z\s\.\=]+)',supp).group(1)
                    suppvar = re.match('([0-9a-zA-Z\s\.]+)=[\s]?([0-9a-zA-Z]+)',supp).group(2)
                suppdomlist.append(suppdom)
                allspecvar.append(supp)
                allspecvar.append(suppvar)

                sheet_supp=wb_supp[suppdom]

                suppvarlist = [cell.value for cell in sheet_supp['A']]

                if suppvar in suppvarlist:
                    suppindex =suppvarlist.index(suppvar)
                    sheet_supp_raw =suppindex+2
                else:
                    sheet_supp_max_raw = sheet_supp.max_row
                    for i in range(sheet_supp_max_raw,1,-1):
                        if sheet_supp.cell(i,1).value == "CONTENT":
                            sheet_supp_raw = i
                            break
                    sheet_supp.insert_rows(sheet_supp_raw-1)

                side1 = Side(style="thin",color="000000")
                border = Border(left=side1,right=side1,top=side1,bottom=side1)
                fille=PatternFill("solid",fgColor="99CC00")

                sheet_supp.cell(sheet_supp_raw-1,1).value = suppvar
                sheet_supp.cell(sheet_supp_raw-1,1).fill=fille

                sheet_supp.cell(sheet_supp_raw-1,3).value = 'Char'
                sheet_supp.cell(sheet_supp_raw-1,4).value = 200
                sheet_supp.cell(sheet_supp_raw-1,4).number_format = '"$"#,##0_);("$"#,##'
                sheet_supp.cell(sheet_supp_raw-1,4).alignment = Alignment(horizontal='left')

                varpage = list(set(var_page[supp]))
                varpage = sorted([int(x) for x in varpage])
                varpage = [str(x) for x in varpage]

                if str1.get() == 'V3.2':
                    if sheet_supp.cell(sheet_supp_raw-1,6).value == "CRF Page "+" ".join(varpage):
                        pass
                    else:
                        sheet_supp.cell(sheet_supp_raw-1,6).value = "CRF Page "+" ".join(varpage)
                        sheet_supp.cell(sheet_supp_raw-1,6).fill=fille
                        sheet_supp.cell(sheet_supp_raw-1,9).value = 'SUPP'
                        sheet_supp.cell(sheet_supp_raw-1,10).value = "=J"+str(sheet_supp_raw-2)+"+1"
                        if suppvar in suppvarlist:
                            pass
                        else:
                            sheet_supp.cell(sheet_supp_raw,1).value = ''
                            sheet_supp.cell(sheet_supp_raw+1,1).hyperlink = ('#CONTENT!A1')
                        for j in range(65,75):
                            sheet_supp[chr(j)+str(sheet_supp_raw-1)].font=openpyxl.styles.Font(name="Arial",size=8,color="000000")
                            sheet_supp[chr(j)+str(sheet_supp_raw-1)].border = border
                elif str1.get() == 'V3.3' or str1.get() == 'V3.4':
                    if sheet_supp.cell(sheet_supp_raw-1,8).value == " ".join(varpage):
                        pass
                    else:
                        sheet_supp.cell(sheet_supp_raw-1,6).value = "Colledted"
                        sheet_supp.cell(sheet_supp_raw-1,6).alignment = Alignment(vertical="center")
                        sheet_supp.cell(sheet_supp_raw-1,7).value = "Investigator"
                        sheet_supp.cell(sheet_supp_raw-1,7).alignment = Alignment(vertical="center")
                        sheet_supp.cell(sheet_supp_raw-1,8).value = " ".join(varpage)
                        sheet_supp.cell(sheet_supp_raw-1,8).fill=fille
                        sheet_supp.cell(sheet_supp_raw-1,11).value = 'SUPP'
                        sheet_supp.cell(sheet_supp_raw-1,12).value = "=L"+str(sheet_supp_raw-2)+"+1"
                        if suppvar in suppvarlist:
                            pass
                        else:
                            sheet_supp.cell(sheet_supp_raw,1).value = ''
                            sheet_supp.cell(sheet_supp_raw+1,1).hyperlink = ('#CONTENT!A1')
                        for j in range(65,79):
                            sheet_supp[chr(j)+str(sheet_supp_raw-1)].font=openpyxl.styles.Font(name="Arial",size=8,color="000000")
                            sheet_supp[chr(j)+str(sheet_supp_raw-1)].border = border


        except Exception as ex:
            pass

    wb_supp.save(directorypath+'/A'+excelname)
    wb_supp.close()

    '''添加 supp 变量标签'''
    if str2.get() != 'Y':
        pass
    elif str2.get() == 'Y':
        doc=fitz.open(pdfpath+'/'+pdfname)
```

```
supp_text = {}
for x in range(pages):
    location = []
    hash_loc = []

    case = []
    page = doc[x]
    words=page.get_text_words()
    for w in words:

        location.append(list(w[1:4:2]))
        hash_loc.append(tuple(w[1:4:2]))
        case.append(w[4])

    loc_case={}
    for i in range(len(hash_loc)):
        if hash_loc[i] in loc_case:
            loc_case[hash_loc[i]]+=[case[i]]
        else:
            loc_case[hash_loc[i]]=[case[i]]

    loc_list1 = list(loc_case.keys())

    for i in range(len(loc_case)):
        loc_case[loc_list1[i]]=' '.join(loc_case[loc_list1[i]])
    k1=[k for k,v in loc_case.items() if re.search('SUPP[A-Z]{2}',v) != None or re.match('SUPP([A-Z]{2})([0-9a-zA-Z\s\.\=]+)',v) != None]
    v1=[v for k,v in loc_case.items() if re.search('SUPP[A-Z]{2}',v) != None or re.match('SUPP([A-Z]{2})([0-9a-zA-Z\s\.\=]+)',v) != None]

    supp_case_loc={}
    for i in range(len(v1)):
        if v1[i] in supp_case_loc:
            supp_case_loc[v1[i]]+=[k1[i]]
        else:
            supp_case_loc[v1[i]]=[k1[i]]

    supp_case_list1 = list(supp_case_loc.keys())

    for i in range(len(supp_case_list1)):
        matchloc=[]
        matchcase=[]
        for j in range(len(supp_case_loc[supp_case_list1[i]])):
            a = supp_case_loc[supp_case_list1[i]][j][0]
            b = supp_case_loc[supp_case_list1[i]][j][1]
            for k in range(len(loc_list1)):
                c = loc_list1[k][0]
                d = loc_list1[k][1]
                e = (c+d)/2
                if (a <= c <= b) or (a <= d <= b) or (a <= e <= b):
                    matchloc.append(loc_list1[k])
            matchcase=[]
            for l in range(len(matchloc)):
                matchcase.append(loc_case[matchloc[l]])
        matchcase=list(set(matchcase))
        matchcase.remove(supp_case_list1[i])
        matchcase='/'.join(matchcase)
        suppcase=supp_case_list1[i]
        supp_text[suppcase]=matchcase

supp_key_list=supp_text.keys()

wb=openpyxl.load_workbook(directorypath+'/A'+excelname)
for supp in supp_key_list:
    try:
        if re.match('[A-Z0-9]+ in SUPP[A-Z]{2}',supp) != None:
            suppdom1=re.match(r'[A-Z0-9]+ in SUPP([A-Z]{2})',supp).group(1)
            suppvar1=re.match(r'([A-Z0-9]+) in SUPP[A-Z]{2}',supp).group(1)
        elif re.match('SUPP([A-Z]{2})([0-9a-zA-Z\s\.\=]+)',supp) != None:
            suppdom1=re.match(r'SUPP([A-Z]{2})([0-9a-zA-Z\s\.\=]+)',supp).group(1)
            suppvar1=re.match(r'[0-9a-zA-Z\s\.]+=[\s]?([0-9a-zA-Z]+)',supp).group(1)

        varlist = [cell.value for cell in wb[suppdom1]['A']]

        if suppvar1 in varlist:
            supp_index=varlist.index(suppvar1)

        sheet=wb[suppdom1]
        fille=PatternFill("solid",fgColor="FFFF00")

        sheet.cell(supp_index+1,2).value = supp_text[supp]
        sheet.cell(supp_index+1,2).fill=fille
    except Exception as ex:
        print("出现如下异常%s"%ex)

wb.save(directorypath+'/A'+excelname)
wb.close()
```

MISSING VARIABLE COMPLETION

Since not all variables annotated in the aCRF exist in the Spec template (e.g., certain custom domains or newly added SDTM variables), the tool automatically detects these missing variables and inserts new rows at the end of the corresponding Domain sheet (before the CONTENT row), filling in page numbers and formatting, marked in yellow for user review.

```
adddomainlist = []
addvarlist = []
wb_add=openpyxl.load_workbook(directorypath+'/A'+excelname)
addsdtmvarlist = list(set(annotslist) ^ set(allspecvar))
for var in addsdtmvarlist:
    if re.match(r'[A-Z]{2}[A-Z]+[0-9]*',var) != None:
        try:
            adddomain = re.match(r'([A-Z]{2})[A-Z]+[0-9]*',var).group(1)

            sheet_add=wb_add[adddomain]
```

```
            sheet_add_max_raw = sheet_add.max_row
            column = sheet_add['A']
            varlist = [column[x].value for x in range(len(column))]

            if var not in varlist:
                adddomainlist.append(adddomain)
                addvarlist.append(var)

                varpage = list(set(var_page[var]))
                varpage = sorted([int(x) for x in varpage])
                varpage = [str(x) for x in varpage]

                if varpage != None:
                    for i in range(sheet_add_max_raw,1,-1):
                        if sheet_add.cell(i,1).value == "CONTENT":
                            sheet_add_raw = i
                            break
                    side1 = Side(style="thin",color="000000")
                    border = Border(left=side1,right=side1,top=side1,bottom=side1)
                    fille=PatternFill("solid",fgColor="FFFF00")

                    sheet_add.insert_rows(sheet_add_raw-1)
                    sheet_add.cell(sheet_add_raw-1,1).value = var
                    sheet_add.cell(sheet_add_raw-1,1).fill=fille

                    if str1.get() == 'V3.2':
                        sheet_add.cell(sheet_add_raw-1,6).value = "CRF Page "+" ".join(varpage)
                        sheet_add.cell(sheet_add_raw-1,6).fill=fille
                        sheet_add.cell(sheet_add_raw,1).value = ''
                        sheet_add.cell(sheet_add_raw+1,1).hyperlink = ('#CONTENT!A1')
                        for j in range(65,75):
                            sheet_add[chr(j)+str(sheet_add_raw-1)].font=openpyxl.styles.Font(name="Arial",size=8,color="000000")
                            sheet_add[chr(j)+str(sheet_add_raw-1)].border = border
                    elif str1.get() == 'V3.3' or str1.get() == 'V3.4':
                        sheet_add.cell(sheet_add_raw-1,8).value = " ".join(varpage)
                        sheet_add.cell(sheet_add_raw-1,8).fill=fille
                        sheet_add.cell(sheet_add_raw,1).value = ''
                        sheet_add.cell(sheet_add_raw+1,1).hyperlink = ('#CONTENT!A1')
                        for j in range(65,79):
                            sheet_add[chr(j)+str(sheet_add_raw-1)].font=openpyxl.styles.Font(name="Arial",size=8,color="000000")
                            sheet_add[chr(j)+str(sheet_add_raw-1)].border = border


        except Exception as ex:

            pass
wb_add.save(directorypath+'/A'+excelname)
wb_add.close()
```

CUSTOM DOMAIN SUPPORT

For user-defined domains (e.g., XZ, YA), the tool looks up the template for a generic sheet based on the first letter of the domain name (e.g., X-), copies and renames it to the target domain name, and automatically corrects the variable prefix (e.g., changes X-VAR to XZVAR).

```
user_defined_list = []
for i in domainlist:
    if re.match('[X-Z][A-Z]', i) != None:
        user_defined_list.append(i)
for i in user_defined_list:
    if i not in sheet_names:
        try:
            j = i[0]+'-'
            if j not in sheet_names:
                addsheet(templatepath+'/'+templatename, excelpath+'/A'+excelname, j)
        except:
            pass

aexcel=openpyxl.load_workbook(excelpath+'/A'+excelname)
for i in user_defined_list:
    if i not in sheet_names:
        user_defined = i[0]+'-'

        sheet=aexcel[user_defined]
        copy_sheet=aexcel.copy_worksheet(sheet)
        copy_sheet.title = i
        copy_sheet_raw = copy_sheet.max_row
        copy_sheet.cell(1,1).value = i+' Domain Mapping Specifications'
        copy_sheet.cell(15,8).value = i
        for j in range(17,copy_sheet_raw-2):
            if str(copy_sheet.cell(j,1).value)[1] == '-':
                copy_sheet.cell(j,1).value = i + copy_sheet.cell(j,1).value[2:]
for j in user_defined_list:
    if j not in sheet_names:
        try:
            user_defined = j[0]+'-'
            del aexcel[user_defined]
        except:
            pass
        try:
            user_defined = j[0]+'- (2)'
            del aexcel[user_defined]
        except:
            pass
        try:
            user_defined = j[0]+'- (3)'
            del aexcel[user_defined]
        except:
            pass

aexcel.save(directorypath+'/A'+excelname)
aexcel.close()
```

GRAPHICAL INTERFACE

The tool uses Tkinter to build a simple file selection interface. Users simply select the aCRF PDF, Spec Excel, template Excel, and output directory in sequence, then click "Run" to execute. Additionally, radio buttons for template version and a toggle for SUPP label generation are provided for flexible configuration.

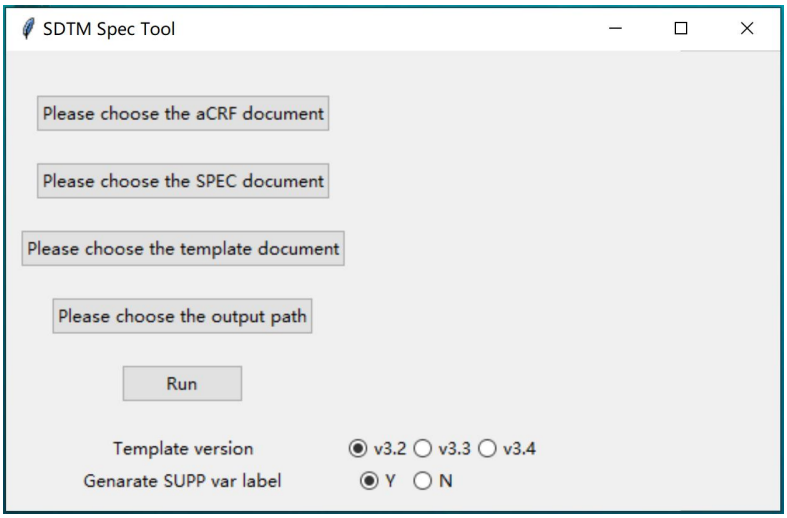

Figure 1. Graphical Interface

RESULTS AND DISCUSSION

PERFORMANCE

For a project with a 200-page aCRF and approximately 1000 annotations, the tool runs in about 3-5 minutes (depending on PDF parsing complexity), whereas manual completion of the same workload would typically take 1–2 days. The generated Spec file can be used directly for subsequent programming with minimal manual correction required.

ACCURACY VERIFICATION

Through retrospective testing on 10 projects, the tool achieved over 99% accuracy for standard variable recognition and 99% for SUPP variable recognition.

VERSION COMPATIBILITY

The tool has been adapted for three mainstream SDTM templates (V3.2, V3.3, V3.4) and includes extension interfaces to accommodate future version changes. Users only need to provide the corresponding template Excel file.

LIMITATIONS

Relies on the standardization of PDF annotation objects; does not work with scanned aCRFs (which lack real annotation layers).

CONCLUSION

This paper presents a Python-based automated SDTM Specification population tool that achieves full automation from aCRF annotation extraction to Spec filling. The tool offers the following advantages:

High Efficiency: Reduces days of work to minutes.

High Accuracy: Reduces errors through multi-layer filtering and intelligent matching.

High Flexibility: Supports multiple template versions, custom domains, and user configurations.

Open-Source and Free: Easy to customize and extend.

The tool has been successfully applied in several clinical projects, significantly improving the quality and efficiency of SDTM Spec development.

CONTACT INFORMATION

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

Ziyu Li
1020348735@qq.com