PharmaSUG 2022 - Paper AD-069

A Shiny App to Spell Check an ADaM Specification

Simiao Ye, Jeff Xia, Merck & Co., Inc., Rahway, NJ, USA

ABSTRACT

The ADaM define.xml is a required file that describes the structure and content (datasets, variables, value level metadata, and controlled terminologies) of the data in a submission package. To produce a high quality define.xml file, Pinnacle 21 Enterprise (P21E) is often a preferred tool for the compliance check and validation of CDISC standards.

However, the features to automatically check spelling errors, SAS syntax, extra space, and traceability of SDTM variables are not implemented in P21E and may demand excessive attention and efforts from the ADaM define.xml authors. This paper describes a Shiny application that can read through various input files including the SDTM define.xml, ADaM specification spreadsheet, and user-defined dictionaries and perform an automatic spell check to ensure document readability and eliminate manual efforts in identifying the spelling errors. Implementation of the app, relevant package and function details, and examples are provided in this paper.

INTRODUCTION

The ADaM define.xml is a required file that describes the structure and content (datasets, variables, value level metadata, and controlled terminologies) of the data in a submission package. Per PHUSE best practices guide (PHUSE 2019), define.xml should:

- be as detailed as possible, particularly for the derived variables, and be written such that someone unfamiliar with the study can locate data values of interest and understand the source of the value or how it was computed or assigned.
- be in plain language with correct spelling and grammar so that it can be understood by anyone using the document.
- not contain internal information, programming/mapping specifications, and abbreviations that are not known across the industry.

ADaM specification in Microsoft Excel format is commonly used across the industry to document the variable names, variable labels, formats, define derivations, etc. It can be utilized as input to create ADaM datasets. Pinnacle 21 Enterprise (P21E) is often a preferred tool for not only conducting the compliance check and validation of CDISC standards, but also generating define.xml from the ADaM specification spreadsheet. Therefore, with this process flow, a well-written ADaM specification is the key to produce a high quality ADaM define.xml file.

Figure 1 is a screenshot from an ADaM specification template in sponsor format. Within an ADaM specification, the “Define Derivation” column is a crucial component to illustrate how variables are mapped or derived in ADaM datasets, thus it should provide sufficient details to help users and reviewers understand the data. Oftentimes, manual editing is necessary for this “Define Derivation” column and consequently demands excessive attention and efforts from the authors to check spelling and grammar errors, facilitate traceability and improve document readability.

Variable Name	Variable Label	Type	Length	Sig Digits	Format	Codelist / Controlled Terms	Origin	Define Derivation
USUBJID	Unique Subject Identifier	Char	30				Predecessor	DM.USUBJID
SUBJID	Subject Identifier for the Study	Char	10				Predecessor	DM.SUBJID
AGE	Age	integer	8				Predecessor	DM.AGE
AGEGR1	Pooled Age Group 1	Char	20			agegr1	Derived	AGE/AAGE is categorized into descriptive values based on the codelist

Figure 1. Example of ADaM Specification in Sponsor Template

To perform a throughout spell check, Microsoft Office has a built-in Spelling & Grammar tool that can help users to find the misspelled words and offer correction options. However, one can end up with gazillion misspelled results from MS spell checking tool as the ADaM specification spreadsheet contains countless CDISC terminologies and variable names that are not in MS dictionaries. The abundant number of false issues is not helpful for the document reviewers. Instead, it becomes a major headache. Therefore, a customized, automatic spell-checking tool to standardize the process and reduce manual efforts will be much more helpful in this review process.

R is a language and environment for statistical computing and graphics, which is gaining popularity in clinical research area. It allows an easy and effective way for data manipulation, analysis, and graphical presentation. Shiny is an R package that makes it convenient to build interactive web apps straight from R. A Shiny app can be hosted on a server, where users can access it from a web browser and produce outcomes with no requirement of R knowledge.

This paper describes a Shiny app that can read through various input files including the SDTM define.xml, ADaM specification spreadsheet, and user-defined dictionaries, and perform an automatic spell check to ensure document readability and eliminate manual efforts in identifying spelling errors. Implementation of the app, relevant package and function details, and examples are provided in this paper.

PROCESS OVERVIEW AND IMPLEMENTATION

The process and implementation of the app is presented in three main steps [**Figure 2**], and brief details for each step are described below:

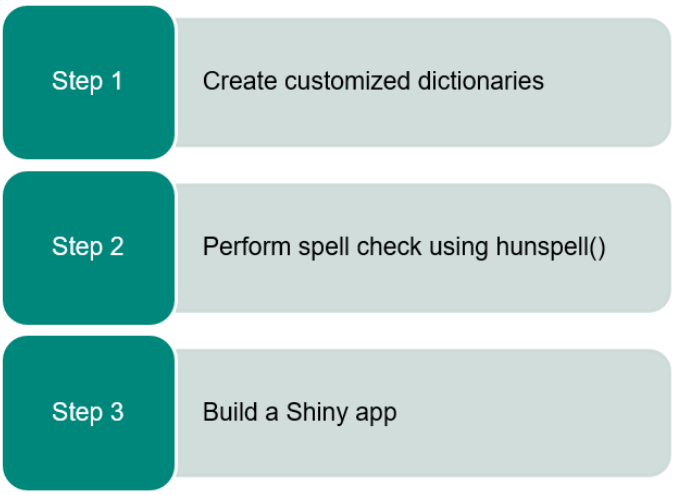


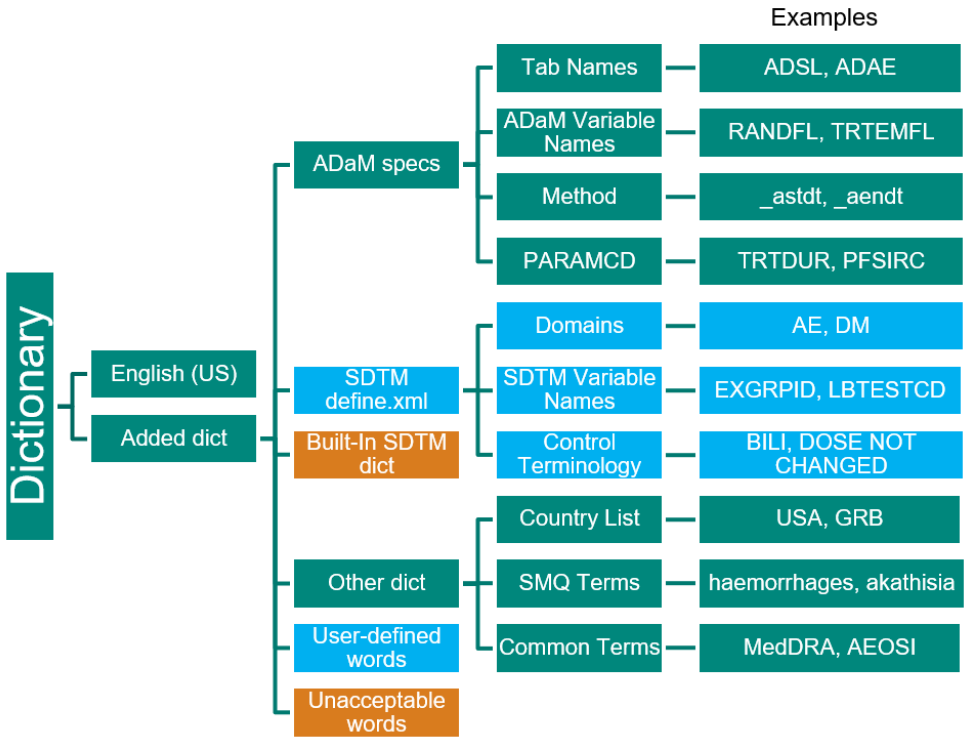
Figure 2. Process and Implementation

Step 1: Create customized dictionaries

A composite dictionary is fundamental to the app. It is built by reading through various input files and combining them together, which includes ADaM specification spreadsheet (either sponsor format or P21E format), SDTM define.xml (optional), CDISC common terminologies, and user-defined words on top of American dictionary from LibreOffice extensions.

Here is an itemized breakdown of the composite dictionary components [**Figure 3**]:

- English (US): American dictionary from LibreOffice extensions (https://extensions.libreoffice.org/extensions/english-dictionaries) is a built-in dictionary from hunspell package and can be referenced directly.
- ADaM specification spreadsheet: R function readxl::read_xlsx() is used to read in the spreadsheet and convert each tab into a data frame. After the conversion, ADaM dataset names, variable names, derivation method names, and PARAMCD from BDS datasets are pulled from the data frames to a consolidated vector.
- SDTM define.xml file (optional): to obtain the list of text including SDTM domain names, variable names, and control terminologies from a project specific SDTM define.xml, similar method is used as illustrated in Shiny diagram paper (Ye 2021): The app parses the XML file and generates an R structure representing the XML tree using function XML::xmlTreeParse(), then extracts XML nodes and attributes by calling functions XML::getNodeSet() and XML::xmlGetAttr(), and finally returns to a data frame of the target text. The SDTM define.xml file is optional for the app, however, if users provide the project specific SDTM define.xml as an input, it will tailor the dictionary, lower the false positive rate, and increase the accuracy of the check.
- Built-in SDTM dictionary: in the scenario that no project specific SDTM define.xml file is provided by users, a pre-defined SDTM dictionary from an oncology study will kick in and be utilized.
- Other dictionaries: some terms that are commonly used in the industry, i.e., country list, SMQ terms, and MedDRA terms are included in the app by default.
- User-defined words: users can add additional words such as project specific drug names or symptoms, this is extremely helpful to reduce the number of false positive issues.
- Unacceptable words/formats: users have the option to check for SAS syntax and extra spaces between words.



Note: boxes in blue are optional, boxes in orange are triggered by user actions.

Figure 3. Composite Dictionary of the App

Step 2: Perform spelling check using hunspell()

hunspell (https://hunspell.github.io/) is the spell checker of LibreOffice, OpenOffice.org, Mozilla Firefox 3 & Thunderbird, Google Chrome, and it is also used by proprietary software packages, like macOS, InDesign, memoQ, Opera and SDL Trados. For its popularity and powerfulness, R package hunspell is also available in CRAN (https://cran.r-project.org/web/packages/hunspell/index.html), which aims in analyzing or checking individual words. Within the package, the hunspell() function scans the text, extracts each word, compares each word with a known list of correctly spelled words (i.e., a customized or composite dictionary), and ultimately returns to a list of misspelled words. It is noteworthy that the function is case sensitive to help users to identify the inappropriate usage of letter casing.

Below is a simple example of how hunspell() function works. The execution of hunspell(text)will return to a list of two words: “ADaM” and “specifiction”, as the checker is case sensitive and “ADaM” is not a typical word in American English dictionary, and “specifiction” is a typographical of “specification”.

```
# load the package
library(hunspell)
# create input text and check spelling
text <- "I want to check ADaM specifiction."
hunspell(text)
```

For the same sentence, if we add the word “ADaM” to the function parameter add_words, then hunspell(text, dict = new_dict) will return to the only misspelled word “specifiction”.

```
# add new word to dictionary and check spelling
new_dict <- hunspell::dictionary(lang = "en_US", add_words = "ADaM")
hunspell(text, dict = new_dict)
```

This works similarly to all the other spell checkers we are familiar with, but the app focuses on consolidating the list of contexts added to the parameter add_words so as to reduce the manual maintenance of the dictionaries.

Here is another example using the DCSCOVFL variable from ADSL [**Figure 4**], by running hunspell() with the customized dictionary, the highlighted words are captured in the app due to the following improper usages:

- discontiued: typo of discontinued
- NULL/null: unacceptable word, should be replaced by ‘missing’ in plain language
- suppds.qnam, suppds.qval: inappropriate letter casing
- ‘Y: unbalanced quotation mark
- strip(): unacceptable word which is a SAS syntax
- DTHCOFL: typo of DTHCOVFL

Example with extraneous info, spelling errors and SAS syntax:

Variable Name	Variable Label	Type	Length	Origin	Define Derivation
DCSCOVFL	Reas for DC from Study Asso COVID Flag	Char	1	Derived	Set to ‘Y’, if subjects discontiued from study (ADSL.DCSREAS not NULL) and the discontinuation reason is associated with COVID, e.g., (1) ADSL.DCSREAS not null and the corresponding suppds.qnam='DSEPRELI‘ and suppds.qval='Y (2) strip(ADSL.DCSREAS) = "Death" and DTHCOFL = ‘Y’ (Added per suggestion from COVID19 A&R SME Jane Doe due to lack of study DEG update on DS01)

Improved version:

Variable Name	Variable Label	Type	Length	Origin	Define Derivation
DCSCOVFL	Reas for DC from Study Asso COVID Flag	Char	1	Derived	Set to ‘Y’, if subjects discontinued from study (ADSL.DCSREAS is not missing) and the discontinuation reason is associated with COVID, e.g., (1) ADSL.DCSREAS not missing and the corresponding SUPPDS.QNAM='DSEPRELI‘ and SUPPDS.QVAL='Y' (2) ADSL.DCSREAS = "Death" and DTHCOVFL = ‘Y’

Figure 4. Derivation Examples of DCSCOVFL Variable from ADSL

Step 3: Build a Shiny app

After a Shiny app is deployed centrally and hosted on a server, it provides a web-based interface to users requiring no knowledge of R. The proposed app comes with comprehensive instructions, supports upload and download activities, and display dynamic results based on user actions. Here is a screenshot of the user interface of the app [**Figure 5**].

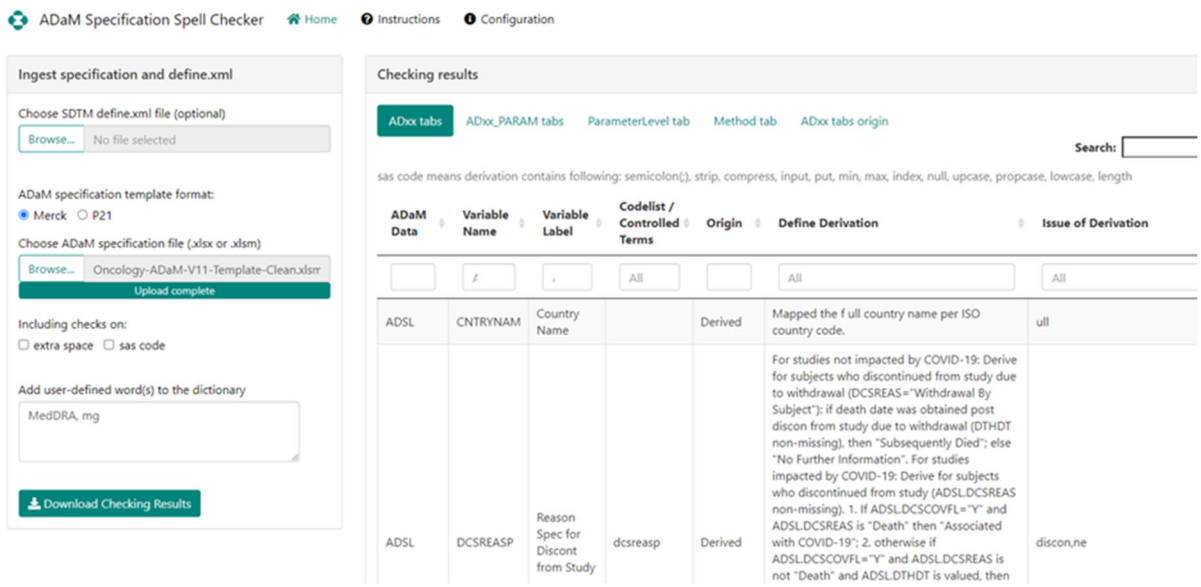


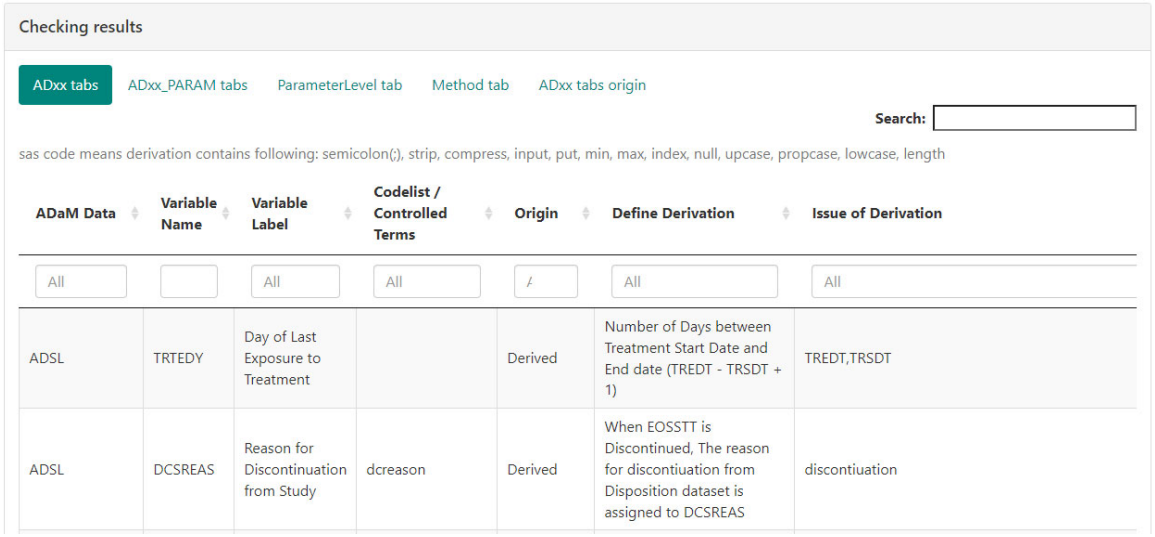
Figure 5. User Interface of the App

The **Home** tab hosts two panels: the input panel (“Ingest specification and define.xml”) and the output panel (“Checking results”).

Input panel (“Ingest specification and define.xml”) locates on the left-hand side of the app, where users can decide:

- whether to upload project specific SDTM define.xml file
- which ADaM specification format to be used (either sponsor format or P21E format)
- whether to further check for extra space and/or SAS syntax
- whether to add additional user-defined project specific words to the dictionary
- whether to download the results in .xlsx format

Output panel (“Checking results”) locates on the right-hand side of the app, by using a sponsor format ADaM specification, the results are to be printed under 5 sub-tabs (ADxx tabs, ADxx_PARAM tabs, ParameterLevel tab, Method tab, and ADxx tabs origin) [**Figure 6**]. The individual checking results for each variable are captured and illustrated in the last column (i.e., “Issue of Derivation”) of the table.



Checking results

ADxx tabs | ADxx_PARAM tabs | ParameterLevel tab | Method tab | ADxx tabs origin

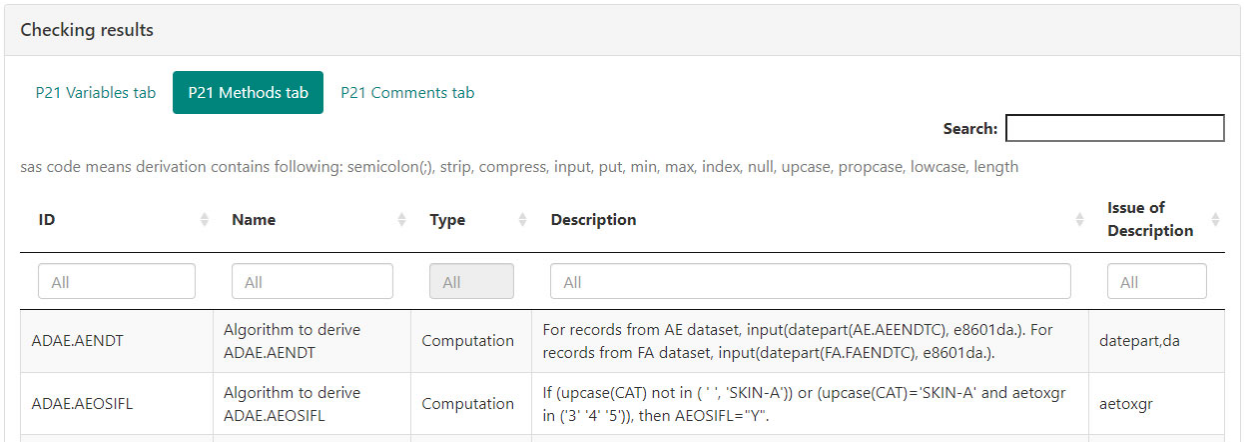
Search:

sas code means derivation contains following: semicolon(;), strip, compress, input, put, min, max, index, null, upcase, propcase, lowcase, length

ADaM Data	Variable Name	Variable Label	Codelist / Controlled Terms	Origin	Define Derivation	Issue of Derivation
All		All	All	/	All	All
ADSL	TRTEDY	Day of Last Exposure to Treatment		Derived	Number of Days between Treatment Start Date and End date (TREDT - TRSDT + 1)	TREDT,TRSDT
ADSL	DCSREAS	Reason for Discontinuation from Study	dcreason	Derived	When EOSSTT is Discontinued, The reason for discontiuation from Disposition dataset is assigned to DCSREAS	discontiuation

Figure 6. Example of Checking Results from ADaM Specification in Sponsor Format

By using a P21E format ADaM specification, the results are to be printed under 3 sub-tabs (P21 Variables tab, P21 Methods tab, and P21 Comments tab) [**Figure 7**]. The individual checking results for each variable are captured and illustrated in the last column (i.e., “Issue of Description”) of the table.



Checking results

P21 Variables tab | P21 Methods tab | P21 Comments tab

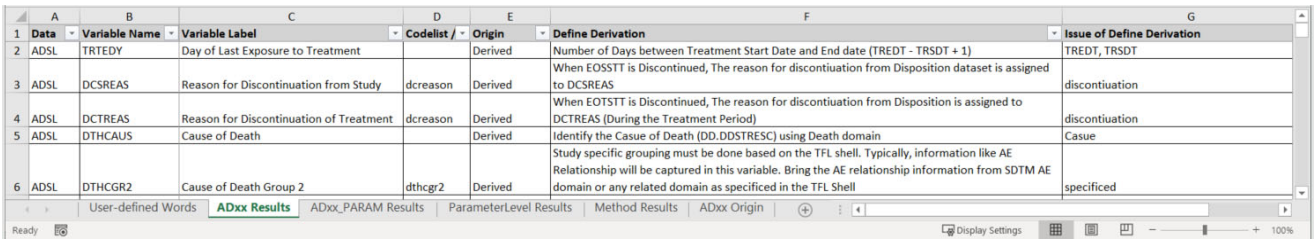
Search:

sas code means derivation contains following: semicolon(;), strip, compress, input, put, min, max, index, null, upcase, propcase, lowcase, length

ID	Name	Type	Description	Issue of Description
All	All	All	All	All
ADAE.AENDT	Algorithm to derive ADAE.AENDT	Computation	For records from AE dataset, input(datepart(AE.AEENDTC), e8601da.). For records from FA dataset, input(datepart(FA.FAENDTC), e8601da.).	datepart,da
ADAE.AEOSIFL	Algorithm to derive ADAE.AEOSIFL	Computation	If (upcase(CAT) not in (' ', 'SKIN-A')) or (upcase(CAT)='SKIN-A' and aetoxgr in ('3' '4' '5')), then AEOSIFL="Y".	aetoxgr

Figure 7. Example of Checking Results from ADaM Specification in P21E Format

Once the check is done on the website, a formatted Excel spreadsheet can be downloaded by clicking on the “Download Checking Results” button at the bottom of the input panel. As shown in **Figure 8**, the spreadsheet is filtered by default and well-formatted with colored header and freeze panes. User can read through the issue column and decide if modifications are needed in the ADaM specification. After all applicable issues are addressed, one can repeat the checking process and upload the updated ADaM specification for the next rounds of checking until a satisfactory version is achieved.



	A	B	C	D	E	F	G
1	Data	Variable Name	Variable Label	Codelist /	Origin	Define Derivation	Issue of Define Derivation
2	ADSL	TRTEDY	Day of Last Exposure to Treatment		Derived	Number of Days between Treatment Start Date and End date (TREDT - TRSDT + 1)	TREDT, TRSDT
3	ADSL	DCSREAS	Reason for Discontinuation from Study	dcreason	Derived	When EOSSTT is Discontinued, The reason for discontiuation from Disposition dataset is assigned to DCSREAS	discontiuation
4	ADSL	DCTREAS	Reason for Discontinuation of Treatment	dcreason	Derived	When EOTSTT is Discontinued, The reason for discontiuation from Disposition is assigned to DCTREAS (During the Treatment Period)	discontiuation
5	ADSL	DTHCAUS	Cause of Death		Derived	Identify the Casue of Death (DD.DDSTRESC) using Death domain	Casue
6	ADSL	DTHCGR2	Cause of Death Group 2	dthcgr2	Derived	Study specific grouping must be done based on the TFL shell. Typically, information like AE Relationship will be captured in this variable. Bring the AE relationship information from SDTM AE domain or any related domain as specificed in the TFL Shell	specificed

Figure 8: Example of the Downloaded Spreadsheet

CONCLUSIONS

The proposed Shiny app in this paper greatly simplifies the process to manually identify the spelling errors and improper formats in Define Derivation, saving a lot of efforts and time for the ADaM spec reviewers and define.xml authors. It not only provides a more accurate and efficient checking than manual scanning, but also can ensure proper plain language and symbols are used.

REFERENCES

PHUSE, 2019. “Best Practices for Documenting Dataset Metadata: Define-XML Versus Reviewer’s Guide”. https://phuse.s3.eu-central-1.amazonaws.com/Deliverables/Optimizing+the+Use+of+Data+Standards/Best+Practices+for+Documenting+Dataset+Metadata-Define-XML+Versus+Reviewers+Guide.pdf

Jeroen Ooms. 2018. “hunspell”. R package version 3.0 available at https://github.com/ropensci/hunspell

https://hunspell.github.io/

Simiao Ye, Yilong Zhang. 2021. “A Shiny App for Generating Data Dependency Flowchart in ADRG”. Proceedings of PharmaSUG 2021.

ACKNOWLEDGMENTS

We would like to thank Merck late-stage oncology programming team for the discussion and testing on the features and documentations of the Shiny app, which helped us develop and maintain a neat and stable app. And special thanks to Jamuna Purma and Yizhuo Zhong for their review and inputs on the paper.

CONTACT INFORMATION

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

Simiao Ye
Merck & Co., Inc., Rahway, NJ, USA
simiao.ye1@merck.com

Jeff Xia
Merck & Co., Inc., Rahway, NJ, USA
jeff.xia@merck.com