PharmaSUG 2021 - Paper SS-058

A Shiny App for Generating Data Dependency Flowchart in ADRG

Simiao Ye, Yilong Zhang, Merck & Co., Inc., Kenilworth, NJ, USA

ABSTRACT

Analysis Data Reviewer's Guide (ADRG), which provides agency reviewers with context for analysis datasets and terminology, is an important part of a standards-compliant analysis data submission for clinical trials. Within the ADRG, the data dependencies section is used to describe any dependencies among analysis datasets, where a flowchart (diagram) is recommended to demonstrate the relationship concisely and intuitively. Shiny is an open-source R package for building web applications straight from R. This paper discusses how to utilize Shiny to automatically generate a data dependency flowchart through reading in ADaM define.xml and adapting user-input dataflows. The Shiny app simplifies manual steps to create the analysis datasets flowchart and ensures the flowchart matches with define.xml. Implementation of the app, relevant package/function details, and examples are provided in this paper.

INTRODUCTION

The ADRG is intended to describe analysis data submitted for individual or multiple studies in the Module 5 clinical section of the Electronic Common Technical Document (eCTD) (FDA 2018). Within the required section of analysis data creation and processing issues in the ADRG, data dependencies are used to describe any dependency between analysis datasets. A flowchart is recommended when there are dependencies between analysis datasets beyond a dependency on ADSL. **Figure 1** is an example flowchart that gives us the intuitive idea of the relationship among datasets, i.e., ADEX is derived based on ADSL, and then used in derivation of ADEXSUM.

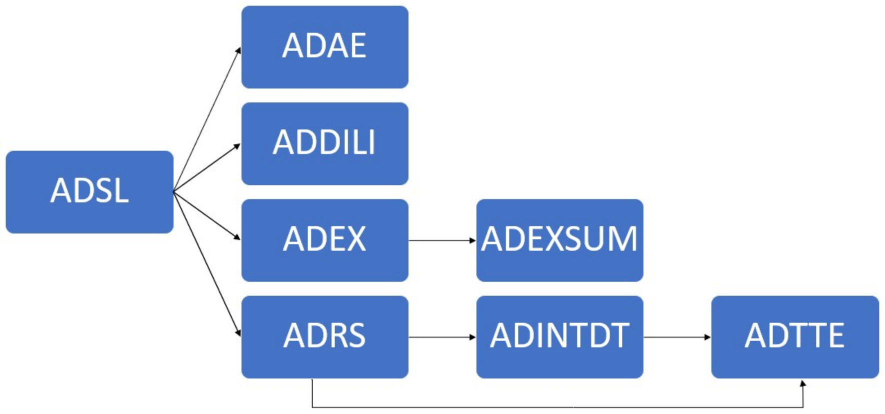


Figure 1. Flowchart Example

In order to obtain the data dependency flowchart, Microsoft office is commonly used because it offers powerful and rigid diagram templates in SmartArt graphics. However, current SmartArt flowchart/diagram types do not support what we tend to have for the data dependencies, many edits are often needed such as adding text, adding/removing shapes, formatting the arrow styles, adjusting the positions and connection lines. Additionally, it is subject to human errors due to typing in dataset names and labels manually or including a dataset not submitted for the study. This can cause confusion for the reviewers. Moreover, it cannot provide flexibility if we want to add/remove dataset labels, and it may impact the structure that results in rework of the whole flowchart. Lastly, it is difficult to maintain the consistency of flowchart styles among studies since different authors have distinguished preferences on layouts and tools. A method to standardize the process and reduce manual efforts is much more efficient. The automation idea is inspired by Zhao et.al. (2020).

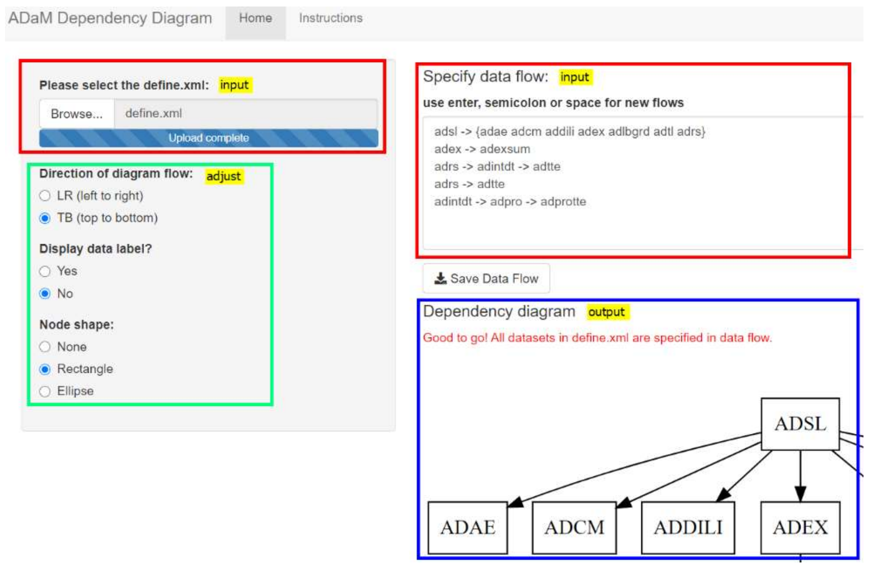
R is a language and environment for statistical computing and graphics, which is gaining popularity in clinical research. It allows data analysis, manipulation, graphical presentation in an easy and effective way. Shiny is an R package that makes it easy to build interactive web apps straight from R. After the app is deployed centrally and hosted on a server, it provides a user-friendly interface to demonstrate examples and get results without knowledge of R.

This paper discusses the idea of building a Shiny app to achieve automation of the data dependency flowchart. The process, implementation of the app, and relevant R package/function are explained in detail.

BENEFITS

By building a Shiny app (**Figure 2**), the current challenges using Microsoft office are addressed with additional benefits:

- Web-based: minimal knowledge of R or Microsoft office required. Users only need to provide ADaM define.xml and information on the data relationship
- Verified: dataset names and labels are pulled from ADaM define.xml, which ensures the flowchart is consistent with the analysis datasets section in the ADRG. Furthermore, there is an in-app assistant to cross check define.xml towards user-input dataflows and report warning messages if any discrepancies exist
- Flexible: multiple options to control the display of flowcharts (direction, label, node shape) are provided to adjust for different study needs
- Efficient: timesaving by reducing manual efforts and eliminating potential human errors



Please select the define.xml: define.xml

Direction of diagram flow: LR (left to right) [✓ TB (top to bottom)]

Display data label? Yes [✓ No]

Node shape: None [✓ Rectangle] Ellipse

Specify data flow:

use enter, semicolon or space for new flows

```
adsl -> {adae adcm addili adex adlbgrd adtl adrs}
adex -> adexsum
adrs -> adintdt -> adtte
adrs -> adtte
adintdt -> adpro -> adprotte
```

Dependency diagram

Good to go! All datasets in define.xml are specified in data flow.

Figure 2. User Interface of the Shiny App

PROCESS OVERVIEW & IMPLEMENTATION

The workflow and key functions of the Shiny app are summarized in **Figure 3**. The implementation is presented in three main steps:

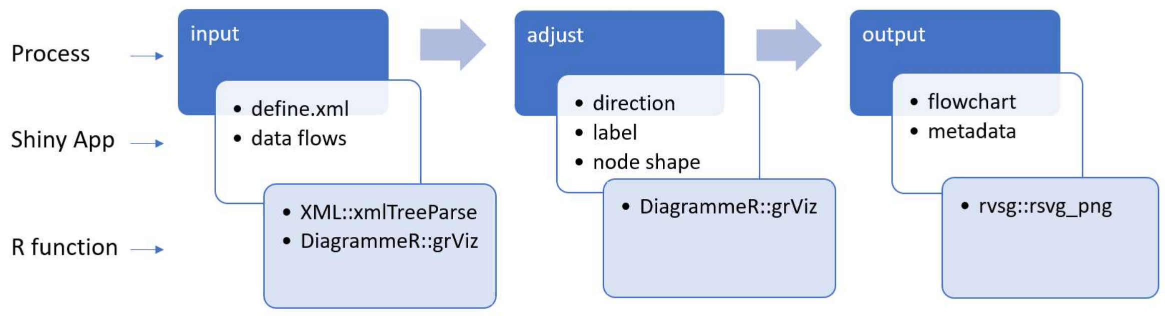


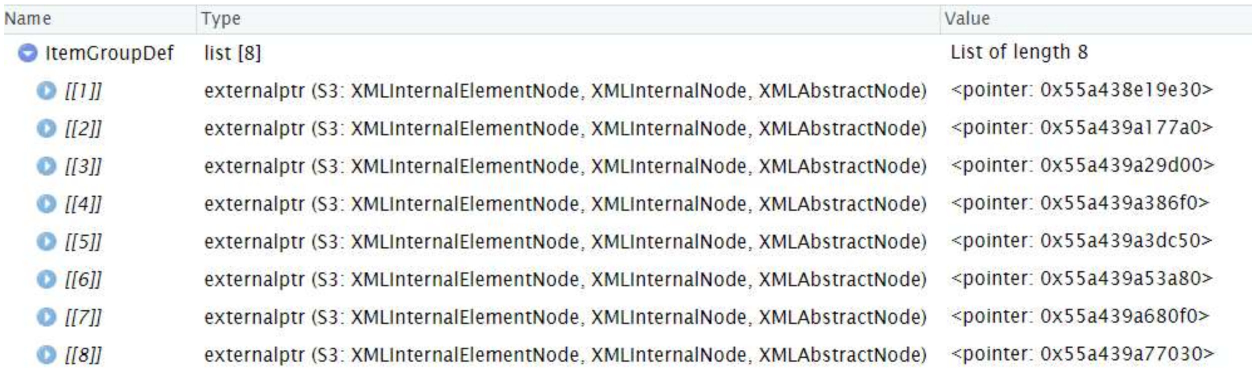
Figure 3. General Process and Key Functions of the App

STEP 1 INPUT:

1.1 Read in define.xml and transform metadata into a data frame

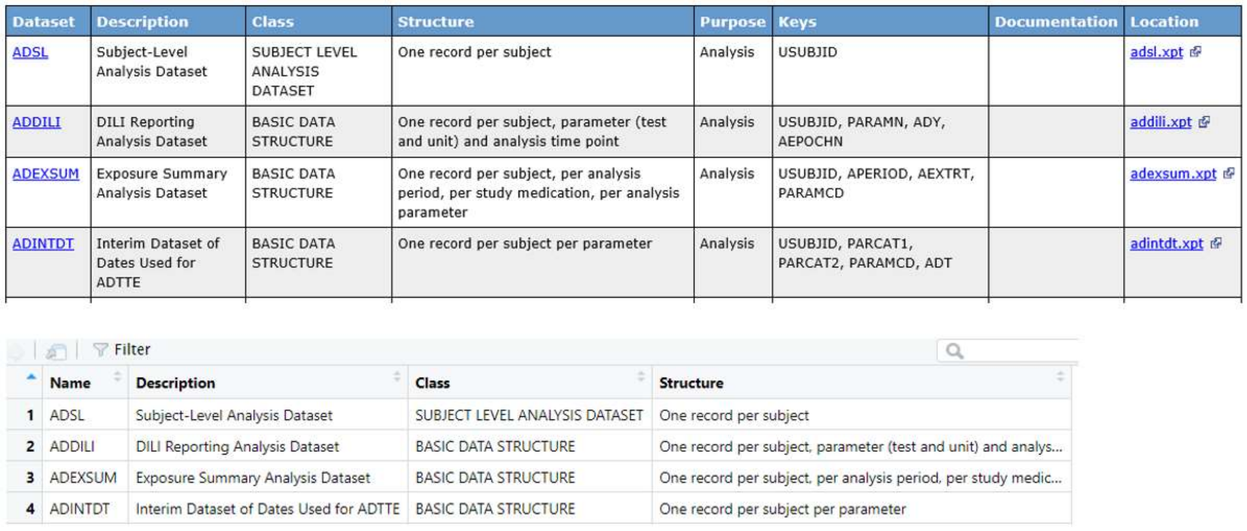
Define.xml transmits metadata that describes any tabular dataset structure. It is required by the United States Food and Drug Administration (FDA) and the Japanese Pharmaceuticals and Medical Devices Agency (PMDA) for every study in each electronic submission. In order to achieve consistency and traceability among submission deliverables, ADaM define.xml is a required component when using this app. After the define.xml is uploaded, 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 ADaM datasets including name, description, structure and class. The general idea of extracting and reorganizing metadata borrows from `R4DSXML::getDLMD()` (Akiya, 2020). Variable name and description are used as input for STEP 2 when plotting the flowchart, while the whole data frame will be presented in the Shiny app using `DT::dataTableOutput()` for reference purpose. **Figure 4** shows the output lists from `XML::getNodeSet()`, each list represents one ADaM dataset. **Figure 5** displays the metadata from ADaM define.xml and the output data frame from R, i.e. the name and structure columns are results from `XML::xmlGetAttr()` in below R code.

```
# Parse XML file
doc <- XML::xmlTreeParse(filepath, useInternalNodes = T)
namespaces <- namespaces(doc)
# Dataset level metadata
ItemGroupDef <- XML::getNodeSet(doc, "//ns:ItemGroupDef", namespaces)
# Extract dataset name
Name <- sapply(ItemGroupDef,
               function(el) XML::xmlGetAttr(el, "Name", default = NA))
# Extract dataset structure
Structure <- sapply(ItemGroupDef,
               function(el) XML::xmlGetAttr(el, "Structure", default = NA))
```



Name	Type	Value
ItemGroupDef	list [8]	List of length 8
[[1]]	externalptr (S3: XMLInternalElementNode, XMLInternalNode, XMLAbstractNode)	<pointer: 0x55a438e19e30>
[[2]]	externalptr (S3: XMLInternalElementNode, XMLInternalNode, XMLAbstractNode)	<pointer: 0x55a439a177a0>
[[3]]	externalptr (S3: XMLInternalElementNode, XMLInternalNode, XMLAbstractNode)	<pointer: 0x55a439a29d00>
[[4]]	externalptr (S3: XMLInternalElementNode, XMLInternalNode, XMLAbstractNode)	<pointer: 0x55a439a386f0>
[[5]]	externalptr (S3: XMLInternalElementNode, XMLInternalNode, XMLAbstractNode)	<pointer: 0x55a439a3dc50>
[[6]]	externalptr (S3: XMLInternalElementNode, XMLInternalNode, XMLAbstractNode)	<pointer: 0x55a439a53a80>
[[7]]	externalptr (S3: XMLInternalElementNode, XMLInternalNode, XMLAbstractNode)	<pointer: 0x55a439a680f0>
[[8]]	externalptr (S3: XMLInternalElementNode, XMLInternalNode, XMLAbstractNode)	<pointer: 0x55a439a77030>

Figure 4. Output List from XML::getNodeSet()



Dataset	Description	Class	Structure	Purpose	Keys	Documentation	Location
ADSL	Subject-Level Analysis Dataset	SUBJECT LEVEL ANALYSIS DATASET	One record per subject	Analysis	USUBJID		adsl.xpt
ADDILI	DILI Reporting Analysis Dataset	BASIC DATA STRUCTURE	One record per subject, parameter (test and unit) and analysis time point	Analysis	USUBJID, PARAMN, ADY, AEPOCHN		addili.xpt
ADEXSUM	Exposure Summary Analysis Dataset	BASIC DATA STRUCTURE	One record per subject, per analysis period, per study medication, per analysis parameter	Analysis	USUBJID, APERIOD, AEXTRT, PARAMCD		adexsum.xpt
ADINTDT	Interim Dataset of Dates Used for ADTTE	BASIC DATA STRUCTURE	One record per subject per parameter	Analysis	USUBJID, PARCAT1, PARCAT2, PARAMCD, ADT		adintdt.xpt

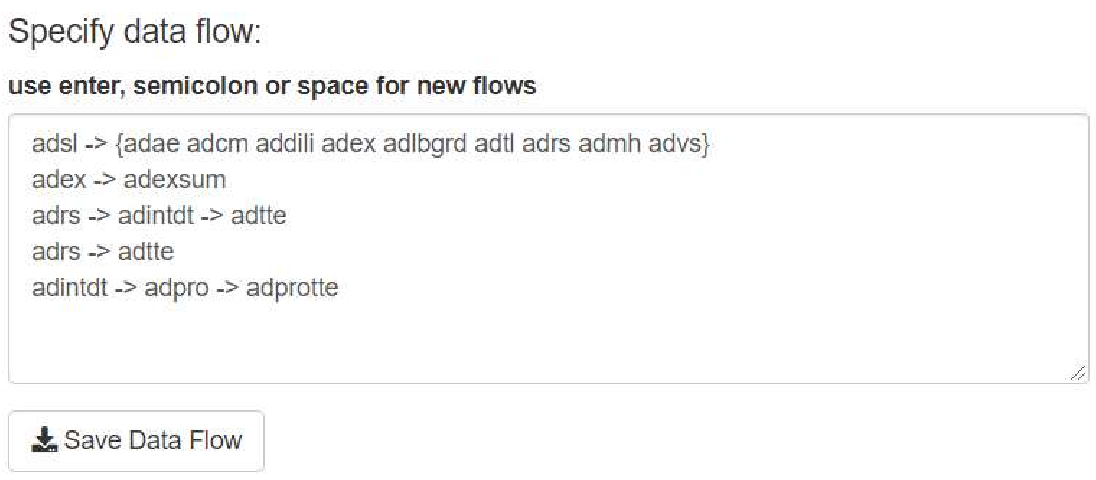
Filter

	Name	Description	Class	Structure
1	ADSL	Subject-Level Analysis Dataset	SUBJECT LEVEL ANALYSIS DATASET	One record per subject
2	ADDILI	DILI Reporting Analysis Dataset	BASIC DATA STRUCTURE	One record per subject, parameter (test and unit) and analys...
3	ADEXSUM	Exposure Summary Analysis Dataset	BASIC DATA STRUCTURE	One record per subject, per analysis period, per study medic...
4	ADINTDT	Interim Dataset of Dates Used for ADTTE	BASIC DATA STRUCTURE	One record per subject per parameter

Figure 5. Metadata from ADaM define.xml (top) and Data Frame from R (bottom)

1.2 Adapt user-input dataflows

Once define.xml is uploaded successfully, the Shiny input box (template dataflows are provided within the app) can be modified according to data relationship within the project. A fundamental part of the input is to use "->" to link datasets. For example, "adsl -> adae" means "adae" is depended on "adsl". The enter key (recommended), semicolon or space is used to separate different dataflows (**Figure 6**). If there are mismatches between define.xml and user-input dataflows, an in-app warning message will appear. This serves as an app assistant when sorting out the dependency relationship. For reproducibility, "Save Data Flow" feature is provided to save all the content from Shiny input box into a .txt file.


Specify data flow:

use enter, semicolon or space for new flows

```
adsl -> {adae adcm addili adex adlbgrd adtl adrs admh advs}
adex -> adexsum
adrs -> adintdt -> adtte
adrs -> adtte
adintdt -> adpro -> adprotte
```

Figure 6. Dataflows Input Box in the App

STEP 2 ADJUST:

In addition to number of datasets, complexity of data relationship also affects the presentation of the flowcharts. Therefore, three options are provided within the app to give user flexibility to control for better display.

2.1 Flowchart direction

Flowchart direction can be chosen from left to right or from top to bottom. It is suggested to display left to right if data labels are included.

2.2 Dataset labels

Dataset labels can be chosen to hide or display in the flowchart. For configurations of 13 or more datasets, it is recommended to suppress dataset labels to obtain a clear quality diagram that can be used in the ADRG.

2.3 Node shape

The node shape of each dataset can be chosen as none, rectangle or ellipse. Rectangle is the most used node shape.

`DiagrammeR::grViz()` aims to make a diagram in R using `viz.js` javascript (Iannone, 2020). After collecting information from ADaM define.xml, input dataflows and all the selected options, the Shiny app utilizes `DiagrammeR::grViz()` to process and assemble all the components into one flowchart.

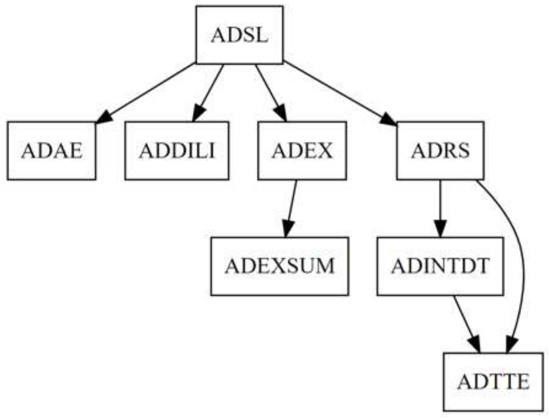
```
# Check if label need to be displayed
if (displaylabel == "Y") {
  ADaM <- noquote(paste(paste0(tolower(ADaM0$Name),
                               " [label = <<b>",
                               ADaM0$Name, "<br/> </b>",
                               ADaM0$Description, ">]"), collapse = " "))
} else {
  ADaM <- noquote(paste(paste0(tolower(ADaM0$Name),
                               " [label = '",
                               ADaM0$Name, "']"), collapse = " "))
}
```

```
# Compile all display option and dataset to the diagram
p_svg <- DiagrammeR::grViz(paste0(
           "digraph {
             graph [layout = dot, rankdir = ", direction, "]
             node [type = lower, fontname = Times, shape = ", shape, "]",
             ADaM, tolower(dataflow), "}"
))
```

STEP 3 OUTPUT:

The output format from `DiagrammeR::grViz()` is Scalable Vector Graphics (svg). The app employs `diagrammeRsvg::export_svg()` and `rsvg::rsvg_png()` to convert svg to Portable Network Graphics (png), and provides the download feature for users. **Figure 7** shows the example outputs from the app using the same dataflows as **Figure 1**. By setting direction from top to bottom, not to display labels and adding rectangle as node shape, we get the first flowchart below. On the other hand, the second flowchart is the result when we set the direction from left to right, display labels and choose none as the node shape.

```
# Convert svg file to png
p_svg %>% DiagrammeRsvg::export_svg() %>%
  charToRaw %>%
  rsvg::rsvg_png("p.png")
```



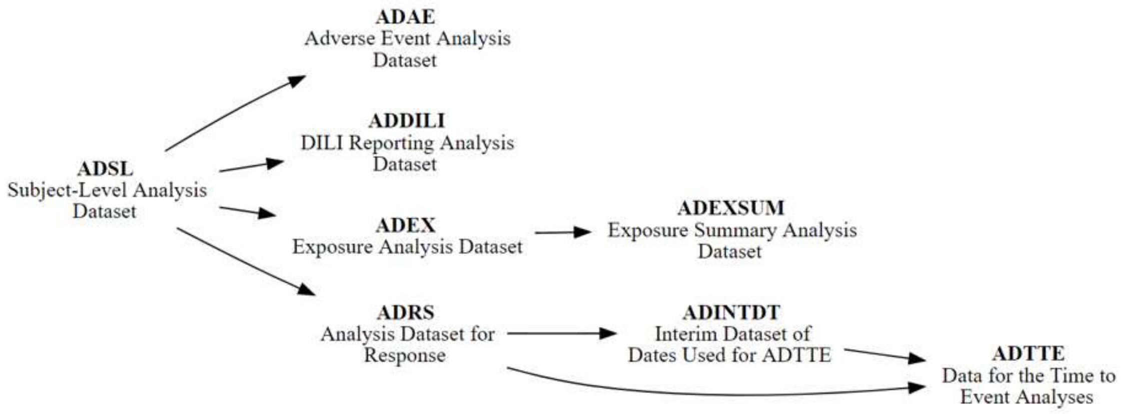

ADAE
Adverse Event Analysis Dataset
ADDILI
DILI Reporting Analysis Dataset
ADSL
Subject-Level Analysis Dataset
ADEX
Exposure Analysis Dataset
ADEXSUM
Exposure Summary Analysis Dataset
ADRS
Analysis Dataset for Response
ADINTDT
Interim Dataset of Dates Used for ADTTE
ADTTE
Data for the Time to Event Analyses

Figure 7. Example Outputs from the App

CONCLUSION AND FUTURE WORK

The Shiny app eliminates many manual steps to create the analysis datasets flowchart and ensures the flowchart matches with ADaM define.xml. While providing flexible controls of the flowchart display and high-quality outputs, it requires minimal knowledge and coding experience of R.

For future release, SDLC (Software Development Life Cycle) is to be implemented to build high-quality app with compliant outputs. Four levels of testing (non-reactive functions, reactive functions, javascript and app visuals) will be covered to provide a fuller simulation of the user experience of the app (Wickham 2020).

REFERENCES

Food and Drug Administration. 2018. “Study Data Technical Conformance”. https://www.fda.gov/media/88173/download

Shunbing Zhao, Jeff Xia. 2020. “Automating of Two Key Components in Analysis Data Reviewer’s Guide”. Proceedings of PharmaSUG 2020.

Ippei Akiya. 2020. “R4DSXML, Read CDISC Data Files”. R Package version 0.6.3. Available at https://github.com/i-akiya/R4DSXML

Richard Iannone. 2020. “DiagrammR”. R Package version 1.0.6.1. Available at https://github.com/rich-iannone/DiagrammeR.

Hadley Wickham. 2020. “Mastering Shiny” https://mastering-shiny.org/index.html. O’Reilly Media.

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 Jeff Xia, Shunbing Zhao, April Shubert and Daniel Woodie 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., Kenilworth, NJ, USA
simiao.ye1@merck.com

Yilong Zhang, Ph.D.
Merck & Co., Inc., Kenilworth, NJ, USA
yilong.zhang@merck.com