PharmaSUG China 2024 - Paper DV-10004

Automatic ADaM Network Analysis Extracting from Metadata/Define/Log with R Shiny

Jun Zhai, Johnson & Johnson

ABSTRACT

This paper presents the design of an R Shiny app that could automatically analyze and visualize ADaM dependencies network extracting from different sources, such as metadata (ADaM specification), define.xml and sas code log. By leveraging network analysis methodology and interactive feature of R Shiny, users could easily choose their available sources and quickly investigate complexed network dependencies of ADaMs structures for any clinical studies. Both backend and frontend design will be introduced in this paper, including three key functions for extracting dependencies information, interactive network visualization using R package “visNetwork”, integrated R Shiny user interface construction using R package “shiny” and “flexdashboard”.

INTRODUCTION

In clinical studies, programmers create SDTM and ADaM for clinical analysis and reporting usage, whose dependencies and relationship could vary from straightforward to extremely complexed, depends on the study design and analysis requirements of different studies. There is always an underlying need of analyzing and visualizing the dependencies structure of ADaM datasets in a relatively convenient and automatic way happening in different scenarios: Programming team is required to fully specify the ADaM dependencies in ADRG writing; Some programmers may would like to obtain a general picture when entering into a new study or taking a reference on old studies; Other cross-functional team members such as statisticians or clinicians hope to utilize the ADaM dependencies visualization to comprehensively understand the clinical data.

With R Shiny, an open-source and powerful interactive data visualizing app based on R language and javascript, we are able to design a user-friendly interface that allows different users, with or without programming skillsets, to choose their available input source and obtain an instant analysis and visualization of the ADaM network.

The users may find ADaM dependencies information from multiple sources, including ADaM Metadatas (also known as ADaM specification), define.xml and log files of ADaM production code. Regular expression methods will be used to extract possible dependencies information from different input sources, and summarize as a excel spreadsheet for users to download.

Network analysis methods is leveraged to help visualize the ADaM dependencies structure. Network analysis is a common technique to depict relations among factors. The two key elements of network analysis are nodes and links, in this paper the nodes represent for ADaM or SDTM dataset, the links are plotted as lines with arrows linking between nodes, indicating that specific SDTM or ADaM is one of the source datasets to generate target ADaMs.

BACKEND: ANALYZE & VISUALIZE

ANALYZE ADAM DEPENDENCIES

To analyze ADaM dependencies from multiple sources, three key functions are created to extract related information from three input formats: excel sheet (ADaM Metadatas), XML file (Define.xml) and log file (ADaM generation code logs). Different regular expression patterns are applied based on the conventions of three sources. In this paper, patterns and writing conventions of documents or codes are based on J&J internal analysis standards, but the rationale and code logic can be generalized for other similar usage.

a. From Metadata

Metadata files usually name after ADaM domain's name, so we can easily search for all available metadata files from selected folder using regular expression patterns "`^AD.*\\.xlsx$`".

In metadatas, variables' derivation rules will be documented in one specific column. If copied from source or using other ADaM variables as predecessor, the derivation rule will be simply written as "XX.YYY", where XX stands for source SDTM or ADaM and YYY stands for source variable. If derived variables, the derivation rule could be one or more complexed sentences, but the column will still contain "XX.YYY" if it's related to other variables. Thus we could use regular expression "`(\\w+)\\.(\\w+)`" to identify this kind of patterns and extract from metadatas' derivation columns.

The accuracy of outcomes depends majorly on how clean and well-written our metadatas are. Most common issue is the bad writing style omits space after period (.), e.g. "…set to Y.Else if…", the code will also identify "Y.Else" as matching the pattern. Luckily within CDISC standards, we know well that our SDTM follows the naming convention either in two letters or has prefix "SUPP", and our ADaMs follows the naming convention begins with "AD". We could obtain a clean dependency result by restricting the pattern of first word before period (.).

b. From Define.xml

The rationale of extracting ADaM dependencies information from Define.xml is generally similar to a. From Metadata, since the Define file is generated from ADaM specification sheet, the derivation rules should be the same. The major technical issue is how to read and analyze XML file using R.

This paper uses functions xmlParse(), xmlToList() from R package "XML" to read Define.xml in R and convert it to ready-to-use dataframes:

```
# read define
define <- xmlParse(file)
define2 <- xmlToList(define)
define3 <- define2$Study$MetaDataVersion
# inspect elements
unique(names(define3))
```

From Figure 1, the output of xmlParse() is an external pointer of XML file, and xmlToList() can read the external pointer into a large list. By inspecting the elements in define2, we find that the main body is stored in define2$Study$MetaDataVersion. With further inspection, we know that the main body includes multiple child elements such as "ValueListDef", "WhereClauseDef", "ItemDef", "CodeList", "MethodDef", "CommentDef", "ItemGroupDef", "leaf", "SupplementalDoc", and variables' derivation rules are stored in "MethodDef".

Data	
define2	Large list (2 elements, 6.8 MB)
define3	Large list (2173 elements, 6.8 MB)
Values	
define	External pointer of class 'XMLInternalDocument'
file	"./define.xml"

Figure 1. Different Define.xml Object in R Environment

Using a simple for loop, we can extract the useful information under “MethodDef” and clean it as a dataframe consisting of three columns “MethodDef.TranslatedText”, “MethodDef..attrs”, “MethodDef.DocumentRef”. Just as Figure 2 shows, first column is the derivation rules and second column is the related variables with different attributes. At this step we successfully scrape the variable names and variable derivation from Define.xml, the remaining codes will search for same pattern mentioned in a. From Metadata, and will not be repeated in this section.

	MethodDef.TranslatedText	MethodDef..attrs	MethodDef.DocumentRef
OID...148	If 12<=age<=17 then cohort='ADOLESCENT'; else if age>=...	MT.ADVSTOX.COHORT	NA
Name...149	If 12<=age<=17 then cohort='ADOLESCENT'; else if age>=...	CM.ADVSTOX.COHORT	NA
Type...150	If 12<=age<=17 then cohort='ADOLESCENT'; else if age>=...	Computation	NA
OID...151	When CENSRFL='Y', set as "Unblinding" if unblinding is earli...	MT.ADVSTOX.CENSRSN	NA
Name...152	When CENSRFL='Y', set as "Unblinding" if unblinding is earli...	CM.ADVSTOX.CENSRSN	NA
Type...153	When CENSRFL='Y', set as "Unblinding" if unblinding is earli...	Computation	NA
OID...154	If subject was treatment unblinded by the site (where DS.DS...	MT.ADVSTOX.CENSRFL	NA
Name...155	If subject was treatment unblinded by the site (where DS.DS...	CM.ADVSTOX.CENSRFL	NA
Type...156	If subject was treatment unblinded by the site (where DS.DS...	Computation	NA
OID...157	Take the earlier datetime from datetime of treatment unblin...	MT.ADVSTOX.CENSRDTM	NA
Name...158	Take the earlier datetime from datetime of treatment unblin...	CM.ADVSTOX.CENSRDTM	NA
Type...159	Take the earlier datetime from datetime of treatment unblin...	Computation	NA

Figure 2. Clean Dataframe Derived from MethodDef

c. From Code Log

Extracting dependencies information from ADaM program logs could be more straightforward. Usually programming team will create two libraries referring to SDTM datasets and ADaM datasets, such as “d_in”, “a_in”, etc. The function has two argument “adlib” “sdlib” to allow users self-define the library names for searching. Then we could search for related libraries names in patterns `paste0("(", tolower(adlib), ")\\.(\\w+)")` or `paste0("(", tolower(sdlib), ")\\.(\\w+)")`. The remaining codes will search for same pattern mentioned in a. From Metadata.

With the help of users-defined library names, we could even investigate ADaM dependencies in legacy studies, whose ADaM/SDTM may not follow any CDISC conventions. Using methods from a. From Metadata and b. From Define.xml, it would be unfeasible to extract information not relying on CDISC naming convention from free text, but extracting from programming language is simpler since the programming language is usually standardized. In this function, a new argument “cdisc” is created to allow users choose whether to apply CDISC standards. When cdisc=FALSE, no naming convention will be restricted for the results.

VISUALIZE NETWORK

In our first section, Analyze ADAM Dependencies will output an excel spreadsheet of ADaM dependencies for users to download and do manual revision. In this section, the function requires users to upload a same structure of excel spreadsheet as input file, and process for further network analysis and visualization using R package “visNetwork”.

As Table 1 shows, our input file includes two columns, DOMAIN is our target ADaM name, and SOURCE is its source SDTM or ADaM. For network analysis, we need to process the data into two datasets: nodes and links. Each nodes represent for a ADaM or SDTM dataset, the links are plotted as lines with arrows linking between nodes. The datasets nodes and links has the structure like Table 2 and Table 3: nodes dataset contains all unique ADaM and SDTM, requires a column of related numeric code value, whose index must start from 0; links dataset includes coding records of original input data, one is the source nodes, another is target nodes, both must in numeric format.

SOURCE	DOMAIN
A	B
C	B
B	D

Table 1. Input Excel Structure

name	code
A	0
B	1
C	2
D	3

Table 2. Nodes Data Structure

from	to
0	1
2	1
1	3

Table 3. Links Data Structure

As the last step, visNetwork() function is called to create an interactive network analysis visualization:

```
visNetwork(nodes = nodes,
           edges = links,
           # main = title,
           width = "100%")
```

Users can also add additional attributes provided by visNetwork to nodes and links dataset, such as group, size, shape, color, arrows, etc. visOptions() can be further applied after visNetwork() to adjust the javascript effects in interactive plot. In this paper, as Figure 3 shows, different shape and color are

designed to distinguish ADaM and SDTM, users could add more features provided by visNetwork package based on their needs.

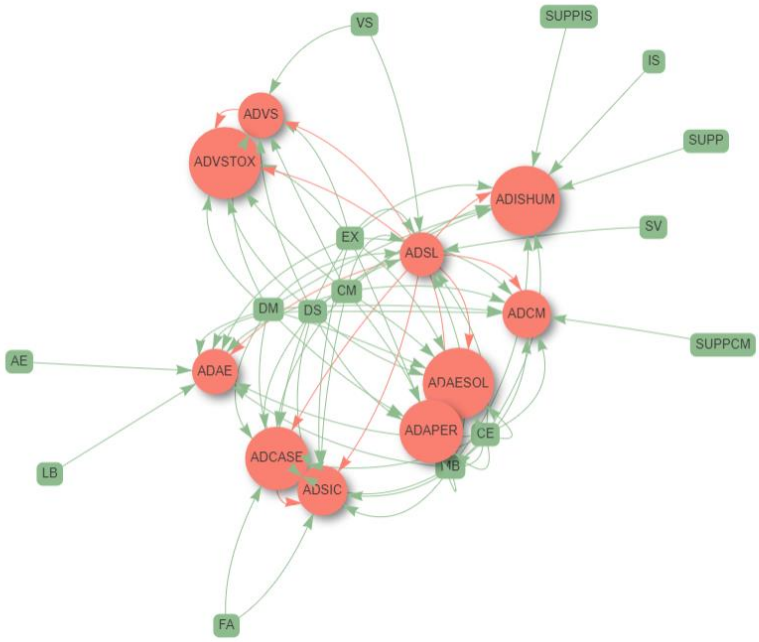

Figure 3. Network Analysis Visualization by visNetwork

FRONTEND: EMBEDDED IN R SHINY

The Shiny App designs two pages: Page 1 for Analyze and Page 2 for Visualize, which correspond to two backend functions introduced in Analyze ADAM Dependencies and Visualize Network. As Figure 4 shows, the website structure and styles directly apply features and theme from R package "flexdashboard".

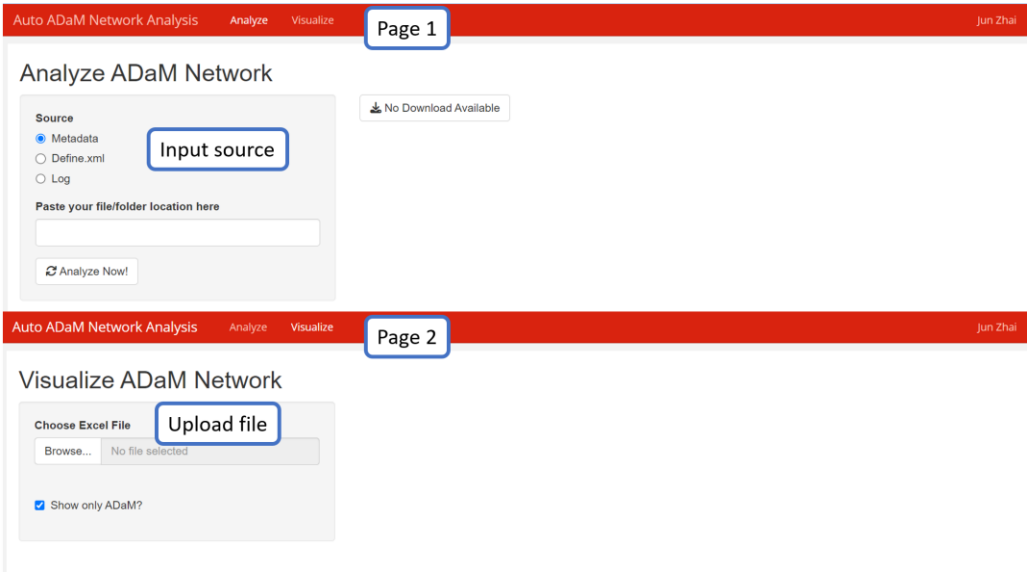

Figure 4. Initial Page of Shiny App

To embed two individual R Shiny app using flexdashboard, we run the app in a RMarkdown file in this paper. The code structure needs to do some modification comparing to a usual shiny app in R file:

```
---
title:
pagetitle:
output:
  flexdashboard::flex_dashboard:
    theme: simplex
    navbar:
      - {…}
runtime: shiny
---

```{r global, include=FALSE}
# Load packages here
```

Page 1
=====================================

```{r,  echo=FALSE}
```
```

```
# Your first shiny app here
shinyApp(
      ui <-fluidPage(…),
      server <- function(input, output, session) {…},
      options = list (height=…)
)

```

Page 2
==========

```{r,  echo=FALSE}

# Your second shiny app here
shinyApp(
      ui <-fluidPage(…),
      server <- function(input, output, session) {…},
      options = list (height=…)
)

```

PAGE 1: ANALYZE

In the first page, a sidebarLayout() is designed for users to select sources and input file/folder location. Considering that extra input arguments are required when selecting “Log” as source, a conditionalPanel() is enabled to let users further input library name and cdisc usage checkbox as Figure 5 presents.

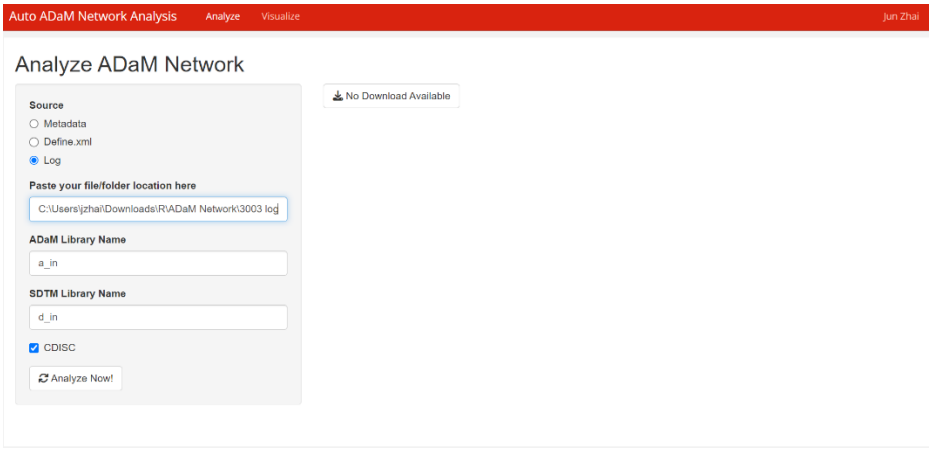

Figure 5. Conditional Panel after Selecting Log as Source

To avoid instant backend calculation when users are still entering file location, an actionButton() is added. When users finish entering, he can click the “Analyze Now!” button to submit all arguments to backend. Then a notification “Start Analyzing” will pop in screen indicating that the analysis is ongoing. When the analysis ends, as Figure 6 shows, the results will show in main page, and “No Download Available” button

will turn to “Download Spreadsheet”, which users can download the output excel file by clicking this button.

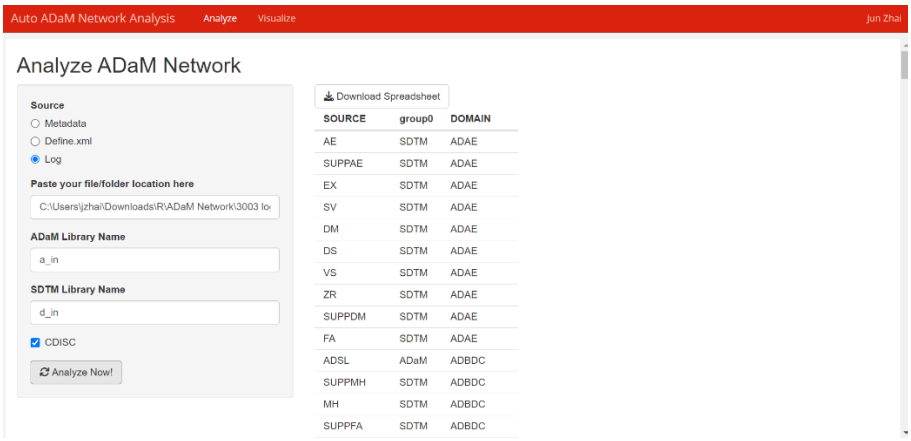

Figure 6. Results and Download Button in Analyze Page

An interactive download button will become enabled only after the results is calculated and available for download, this functionality requires the design of a dummy actionButton() and a downloadButton():

```
ui <-fluidPage(
      …
      uiOutput("dlbutton")
      …
),
server <- function(input, output, session) {
      …
      #download button
        output$dlbutton <- renderUI({
          if(isTruthy(input$gobutton))
            downloadButton("downloadData", "Download Spreadsheet")
          else
            actionButton("dummybutton","No Download Available",icon =
      icon("download"))
        })

        #dummy download button
        observeEvent(input$dummybutton,{
          showModal(modalDialog(
            "Please submit your request first"
          ))
        })

        #download data
        output$downloadData <- downloadHandler(
```

```
            filename = function() {
              paste(input$datasource, "_report_", Sys.Date(), ".xlsx", sep="")
            },
            content = function(file) {
              writexl::write_xlsx(results(), file)
            }
          )
        …
}
```

PAGE 2: VISUALIZE

In Page 2, a fileInput() is embedded in siderbarLayout() to allow users select and upload their network excel spreadsheet; a checkboxInput() is added to select whether to show only ADaM in the visualization plot.

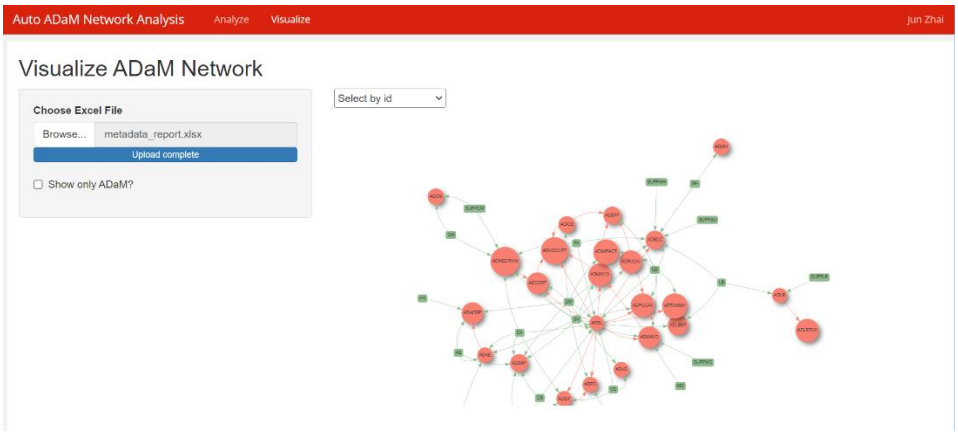

Figure 7. Upload and Visualize in Page 2

Users can also use mouse-hover or dropdown list to select a node (ADaM/SDTM) and obtain an interactively highlighted plot. This is achieved by using `visOptions(highlightNearest = list(enabled = TRUE, hover = TRUE), nodesIdSelection = TRUE).`

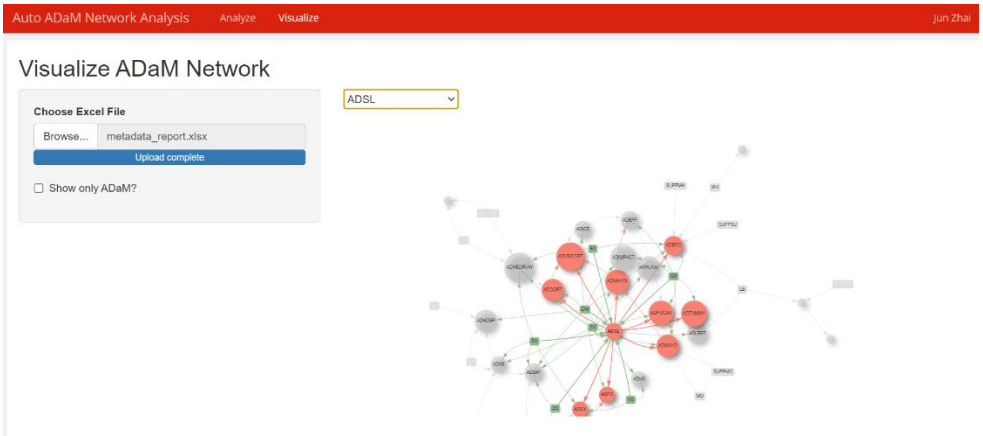

Figure 8. Selecting Nodes and Highlight Features in Page 2

CONCLUSION

With the R Shiny App introduced in this paper, users can easily extract and download ADaM dependencies report from three sources: Metadatas, Define.xml and code logs; and then visualize the ADaM network using uploaded spreadsheets. The powerful R functions and handy user interface design allows users, with or without programming capabilities, to process complicated data and achieve an instant visualization interactively in simple mouse clicks.

The limitations of the functionality are that the accuracy of results depends much on the validity of our input sources and how organized the documents are. Secondly, this app's design and rationale is based on J&J internal standards, thus may not be able to generalize to other external utilizations directly. Nevertheless, I believe the methods and functions introduced in this paper are still valuable and beneficial for our peer programmers, and could be adapt according to people's own needs and scenarios.

Overall, this R Shiny App provides an automatic and interactive approach to analyze and visualize ADaM dependencies, which can benefit programmers' daily work, also provides a powerful tool for cross-functional team members, such as statisticians and clinicians, to easily obtain a straightforward investigation and illustration of clinical data.

REFERENCES

Wickham, Hadley. 2020. *Mastering Shiny*. Sebastopol, CA: O'Reilly Media.

Xie, Yihui, et al. 2023. *R Markdown: The Definitive Guide*. London, UK: Chapman & Hall/CRC.

Xie, Yihui. 2024. *bookdown: Authoring Books and Technical Documents with R Markdown*. London, UK: Chapman & Hall/CRC.

B.V., Almende. visNetwork: Network Visualization using 'vis.js' Library. https://cran.r-project.org/web/packages/visNetwork/index.html

CONTACT INFORMATION

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

Jun Zhai
Johnson & Johnson
jzhai@its.jnj.com