Paper AD03

An Automation Proof of Concept of Periodic Reporting via R Shiny

Nicholas Masel, Janssen R&D, Raleigh, NC, US
Steven Haesendonckx, Janssen R&D, Beerse, BE
Alexandra Papadopoulou, Janssen R&D, Beerse, BE

ABSTRACT

Periodic reporting aims to provide a comprehensive analysis of the risk of a drug to regulatory authorities. In support of this process, we explored the ability of R and R Shiny to create the required rich text format (RTF) outputs. The intent of this proof of concept (PoC) was to generate outputs that are indistinguishable from the more traditional method of RTF production with SAS. In addition, to create an interactive application to allow flexibility in table content and to account for variance in the analysis data structure. If successful, the framework of this tool could be used as the foundation of a broader application for output generation via R. Herein we will discuss our results and recommendations for expansion upon these results.

INTRODUCTION

SAS has been a main player in providing statistical software solutions to pharma industry. However, the statistical software for analysis and reporting in clinical trials can be chosen freely if its fully documented in the submission documents [1]. R is a powerful alternative to SAS and has the advantage of being open-source and having a large user-base. Although it was primarily used in academic settings, it starts to find its way into the pharma industry to support submission efforts. Several pharma companies have joined forces to discuss the validation and application of R in their day-to-day business processes [2,3].

The Shiny package by R studio [4] provides a flexible framework for creating interactive web applications with R. Shiny has a low entry barrier, allowing the user to create and deploy a simple user interface (UI). These web applications allow the user to interact with the data via sliders, drop-down menus, text fields, check boxes, etc. The results are displayed via tables, listings and figures (TLFs). When an input is changed, the outputs are immediately altered, making it an interesting tool for data visualization, exploration and reporting. Once a Shiny web application has been developed, it can be deployed centrally and shared via a web URL, making it available to a wide customer base. Despite that R Shiny is a powerful tool and easily accessible, it has not yet found a big user base in pharma.

The Development Safety Update Report (DSUR) and Periodic Safety Update Report (PSUR) are an integral part of drug development. They aim at providing a comprehensive overview of both the risks and benefits to exposed subjects. To this end, several pre-specified tables and listings need to be generated. Seeing that the PSUR/DSUR needs to be provided to the relevant authorities on a regular basis and the analysis is straightforward, this reporting effort is an ideal candidate for automatization using R and R Shiny.

Using the PSUR/DSUR reporting effort as a Proof of Concept (PoC), this paper highlights how R and R Shiny can be integrated in the daily life of a drug development team to support ongoing data analysis and reporting efforts.

SETTING UP THE ENVIRONMENT

OUR R ECOSYSTEM

When the idea materialized to try a Shiny application to support periodic reporting, we were fortunate to be able to utilize an existing infrastructure set up with great efforts from our IT and Statistics and Decision Science colleagues. A cluster within our JNJ virtual private cloud (VPCx) Amazon Web Services (AWS) instance is dedicated to the support of our R ecosystem. Within this cluster, we have the R Studio Server Pro which, can run multiple versions of R, share projects, run multiple R and python sessions at once, and use Jupyter. In addition, we include the R Studio IDE for development and a R Shiny server to deploy our applications.

THE SHINY APPLICATION

GETTING STARTED WITH SHINY

To get started with any Shiny application, a source file is created in R that defines a user interface (UI) object and a server function. The UI contains the relevant code for the layout and embedded widgets. The various widgets allow the user to interact with the application, providing the required inputs for the application to understand what the end user wishes as output. Examples of standard widgets provided by Shiny are given in Figure 1 [5].

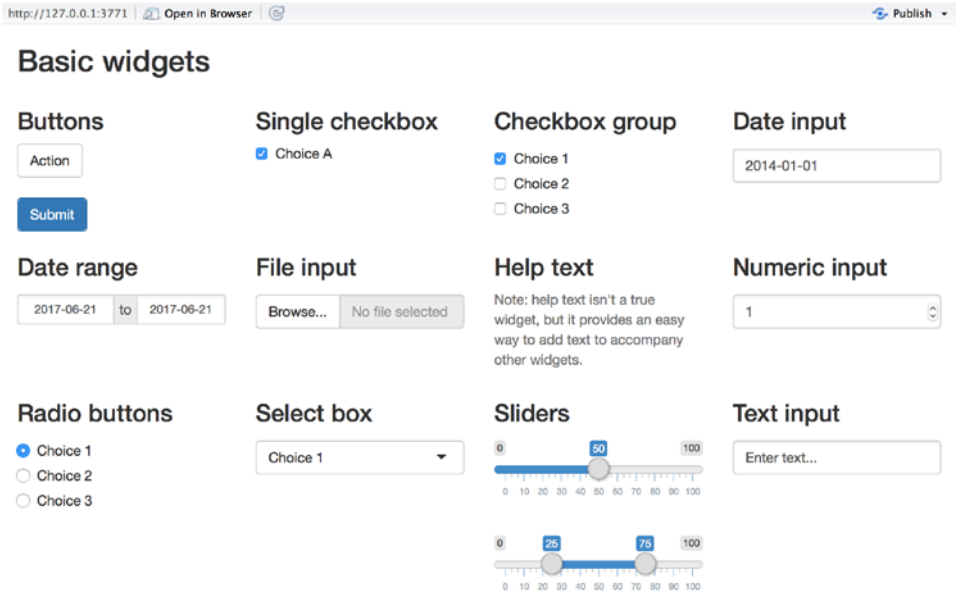

Figure 1: Basic widgets provided by the shiny package

Translation of the input from the user to the displayed output is the task of the server function in the source code. Following minimal example generates a shiny application, resulting one empty fluidPage (web page) saying "Welcome !!!" as can be seen at the right side of the code.

```
library(shiny)

# Define UI for the application
ui <- fluidPage("Welcome !!!")

# Define the server logic
server <- function(input, output) {}

# Run the application
shinyApp(ui = ui, server = server)
```

Welcome !!!

USER INTERFACE – A MODULARIZED APPROACH

For the UI of the PSUR/DSUR application, the navbarPage option was chosen instead of fluidPage. navbarPage allows the user to create several web pages, each which can be accessed using a navigation bar where the name of each web page is displayed (Figure 2 Left). Similar pages can be grouped under the same tab by using the appendTab functionality. This was useful for grouping similar PSUR/DSUR outputs such as listings and tables for safety analysis and reporting (LSF and TSF tab respectively) (Figure 2 Left).

The Shiny app code is usually large and quite complex. Thus, a modularized approach is more desirable. With a modularized approach, the layout of each web page is defined in a separate R script (Figure 2 Right). The source code is split up into smaller files which are sourced at the start of the session. The modules, as they are called, can contain both inputs and/or outputs. Within the PoC they consist from two parts: A part of the UI object and part of the server function, which uses the UI. Modules, although they are part of the Shiny app, cannot be run individually. Using modules with Shiny decreases complexity and improves readability of the source code.

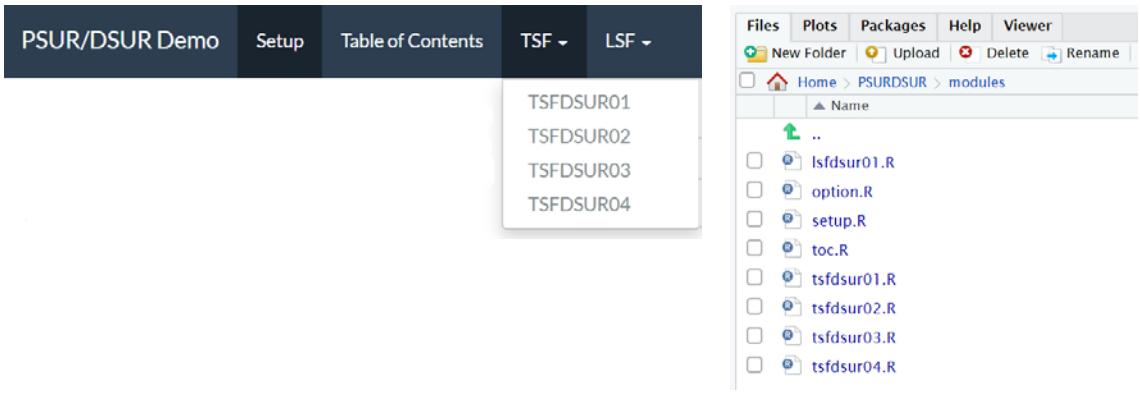

Figure 2:
Left: Navigation bar with defined tabs, generated through the navbarPage option. Some of the interactive web pages can be grouped under the same tab, using appendTab.
Right: Modularized R code, located in a "modules" folder of the R project. Each module generates one interactive web page, which can be accessed by the respective tab inside the navigation bar.

SERVER INPUTS

The PSUR/DSUR is a reporting effort that provides a comprehensive analysis of adverse events occurring during the development of a compound, grouped according to trial and subject characteristics. To perform such an analysis, two Analysis Data Model (ADaM) data sets are pivotal. The first one is a Subject Level data set that contains all baseline characteristics of the subjects under study (ADSL). The second one is an occurrence dataset, containing the information about the experienced adverse events for each subject under study (ADAE). Both datasets need to be uploaded simultaneously in the application via the 'Browse' button in the 'Setup' page (Figure 3) and they need to be in SAS format. For the purpose of this PoC, the ADSL and ADAE from the CDISC SDTM/ADaM Pilot Project were used as a basis (https://www.cdisc.org/sdtmadam-pilot-project). Both datasets were merged back onto themselves to mimic a pooled ADSL and ADAE. In the back-end of the web application, both pooled datasets are combined and used in various downstream algorithms, allowing for the reporting effort to be completed.

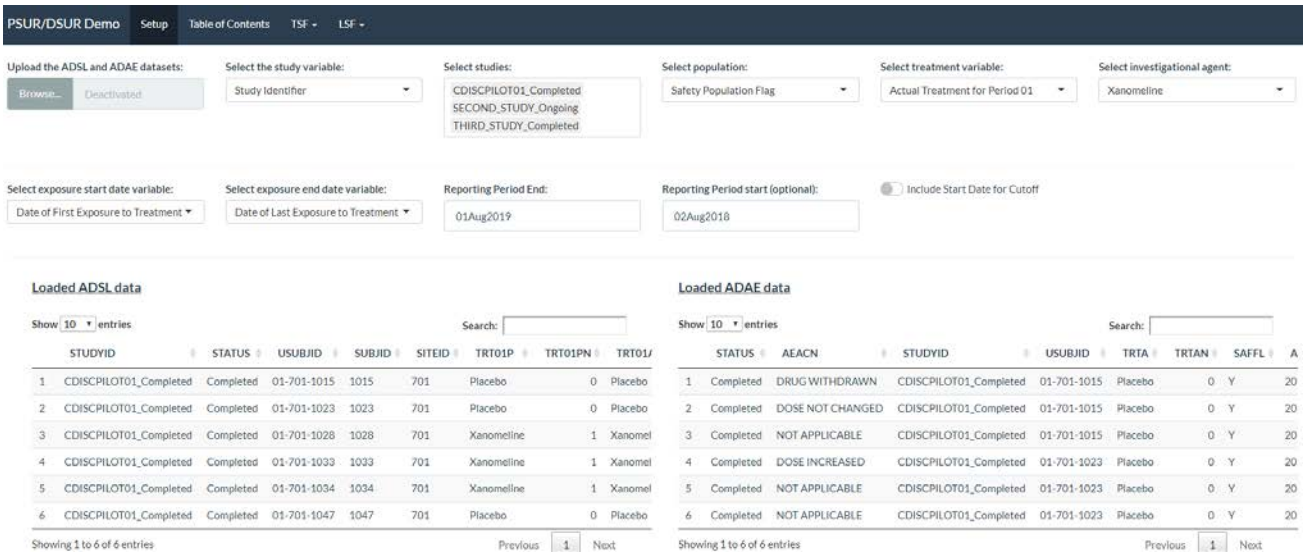

Figure 3: Setup page layout. The Setup page requires both an ADSL and ADAE to be uploaded, after which several drop-down menus become populated. These allow the user to select the relevant variables for analysis. At the bottom of the page, the first rows for each data frame are displayed per included study.

Once the datasets have been uploaded, drop-down lists become populated with possible variable names that should be used in the analysis. These educated guesses are based on string patterns found in the labels associated with the variable names and within the variable names themselves. For example, all variables ending with "-FL" that have the word 'population' in their variable label are retained as possible analysis set variable. Using these drop-down lists, both the involved studies and the required analysis variables can be selected. An additional button allows a date to be used as cutoff for the start of the reporting period. Hover boxes explain the purpose of each widget. At the bottom of the page a preview of both the uploaded ADSL and ADAE is given for reference, with the 5 first records per study produced (Figure 3). These tables are generated, based on the DT package [6]. The DT packages provides an R interface to the JavaScript library DataTables [6], allowing for dynamic interaction with tables displayed on HTML pages. These include filtering, searching and drilling down in the data.
Based on the information provided in the 'Setup' page, the content of all other tabs is updated.

SERVER OUTPUTS

The tab 'TOC' contains a Table Of Content (TOC) of all required outputs with their output identifier and title (Figure 4), generated using the DT package. The output title of the individual outputs can be manually adjusted, later on, in the tab of the respective output.
The LSF tab groups all the listing outputs, while the TSF tab groups all table outputs. For this PoC, the number of listings is limited to one, which will be referred to as LSFDSUR01. The number of tables is limited to four (TSFDSUR01-TSFDSUR04).

Figure 4: Table Of Contents for the reporting effort. The PoC outputs consist of one listing and 4 tables.

If an output requires more analysis variables than provided through the 'Setup' tab, additional dropdown lists will be made available to the user through the respective 'TSF' and 'LSF' tabs. When all information is available, the output is generated on the spot in HTML format.
Although being a powerful tool, the DT package, it is not yet flexible enough to provide a preview of the desired output according to a company's standard requirements for output layout. An elegant workaround is provided by the huxtable package, developed by David Hugh-Jones [7]. Functions in this package manipulate a data frame to add

layers of formatting information such as bolding, indentation, underlining and new rows and allow it to be displayed as such on a HTML page with the 'quick_html' command. Creative usage of the package also allows to add a header with table identifier and title, clarifying footnotes and page numbers. Titles and footnotes can be changed dynamically using the UI of the shiny web application. The interaction of the UI with the table is provided by clicking on the 'Options' button (see Figure 5) A feedback loop ensures that the title in the TOC is updated simultaneously.
The 'Options' button also gives access to another interesting feature. One can save its current selections with regard to the input datasets and the selected analysis variables through the 'Save current selection' button. These inputs can be uploaded in a next session using the 'Upload previous selection'.

Figure 5: Option button

Next to the options button, an additional widget is placed that allows the user to download the displayed table as an RTF document (Figure 5).
RTF, originally developed by Microsoft in 1987 [8], is a flexible language and allows for easy and full control over document properties. It is therefore not surprising that it is still used in many software as the standard language to generate outputs. In R several options are available to write outputs to RFT. In this paper, the huxtable package [7] was chosen as it provides the most flexibility to manipulate the output before and after it has been translated to RTF. Similar functions as for the HTML display were used to put formatting layers on the table of interest after which is written to an RTF document.

REPRODUCIBILITY

When beginning the development within R Studio, it was decided to include PackRat [9] to manage the packages and their dependencies. As packages are installed, they become specific to the PSUR/DSUR project and the packages source code will be wrapped up in a tarball. This tarball can be stored within the PSUR/DSUR project.
User's selections are also important to document in order to be able to restore the user's selections that generated a specific output. To address this, we added save inputs and load inputs to our application's options, as discussed previously. Storing the inputs is quick and easy with the reactiveValuesToList function within Shiny. To reload the inputs, we needed to define a standard so we could update any widget. This was achieved with a naming convention for the input id. For example, all select input ids should be prefixed with "si_". When we load our inputs and an id starts with "si_" we will call the updateSelectInput function from Shiny to update the selected value of that widget to the value we stored.
Once the generated results are satisfactory, one can download the bundle which will include the tarball of source code for the packages, the UI selections, the RTF for all outputs, and all the R scripts necessary to deploy the application.
This approach does not cover all aspects of reproducibility so potential future additions could include R Studio Package Manager, Docker containers, and shinyMeta.

SCALABILITY

Scalability is the ability of the process to be fast and function well regardless the size or volume of the user's necessities. This application can accommodate multiple users at the same time by generating temporary folders for each user that is currently within the application. This allows each user to build their project specific bundle without overwriting other users work. Once the session is complete, the temporary folders and project specific files are removed. If the PoC moves forward into a pilot or for global use, an effort to optimize the code will occur to ensure the application is as quick as possible.

RESULTS

Figure 6 shows an example of a table, TSFDSUR01. This table summarizes the cumulative subject exposure to the investigational drug. For this table to be produced, some parameters needed to be defined. First, at the 'Setup' tab, the Studies and the Investigational Agent (Xanomeline) were chosen. In the TSFDSUR01 tab, additional treatments can be selected for display. Moreover, the possibility is given to the user to group the results according to another variable, eg SEX. The result is shown in Figure 6. As already discussed, the title and footnotes can be updated via the 'option button'. An RTF can be downloaded by clicking on the RTF widget.

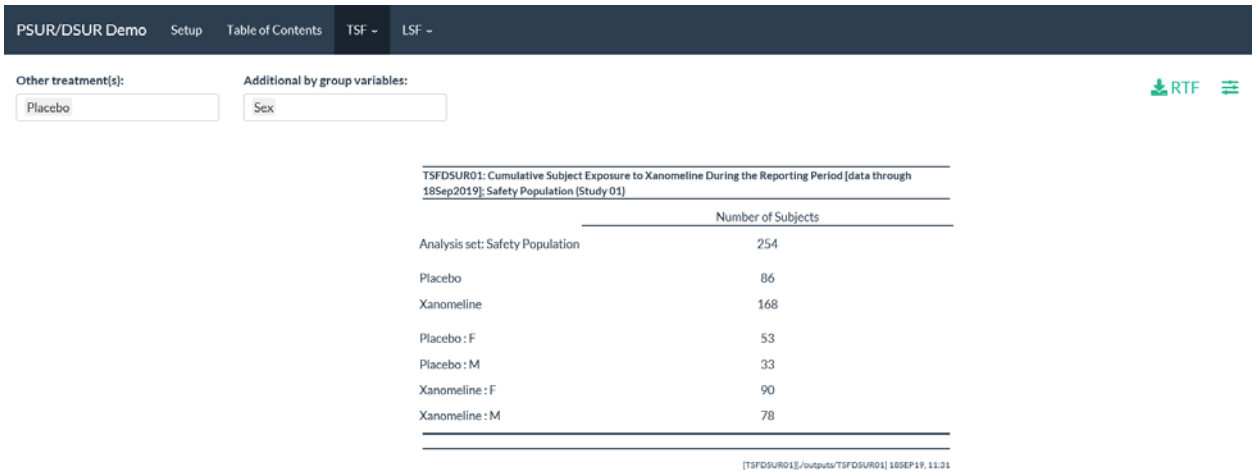

Figure 6: Output example (TSFDSUR01)

CONCLUSION

The need of reporting safety reports/efforts to the authorities is frequent. Pre-specified tables and listings with a relatively simple output need to be generated for the DSUR/PSUR reporting effort. Through this application, R and Shiny showed us that automation is possible. The choice of the specific programming language and packages was considered because of the existing R knowledge from statisticians and statistical programming. Furthermore, there is a low barrier to entry, since a different group within J&J already had the infrastructure setup (R server, R studio, Shiny server). R has a large user base, an active community, is easy to learn and cost-effective. Additionally, it has cutting edge statistical analysis and extensive graphical tool boxes. Shiny is a R package used to build interactive websites. No knowledge of any programming language (like JS, JAVA, HTML etc) is required from the user and this makes it an ideal candidate for interactive analysis and reporting.

As already mentioned, this application is just a starting point to automize and standardize periodic reporting. There is still a long way to go and improvements need to be implemented. One important enhancement relates to the efficiency of the code used. Currently, most of the summary measures are open coded, which makes some parts of the code bulky. The development of a package that calculates summary functions on the fly can resolve this issue. They will have the added benefit that the code will be much easier to debug.

So far only tables and listings are produced. The possibility of the user to produce graphs and manipulate them with the same flexibility that can manipulate tables and listings would be ideal. Being able to reproduce different types of TLF's would be ground breaking. By producing TLF's, users with no programming experience would be able to explore and visualize data in a manner that has not been previously introduced in the pharma industry. This would be cost and time effective. Furthermore, using 'sessionify' will allow multiple users to work in the same application to access and generate outputs for their need, simultaneously.

Another important asset would be the validation of the outputs, packages, and the application. We will continue to look to pharmaR for industry wide validation guidance while we also consider future in-house solutions. Furthermore, SOP's should be conducted, packages with user defined functions should be created and unit testing should be used. Some ideas for reproducibility would be the Docker containers and R Studio Package Manager.

In conclusion, much work still is to be done to integrate R and R shiny in our daily workflow, but this PoC will support the current revolution of using R which awaits Pharma.

REFERENCES

1. https://www.fda.gov/media/109552/download
2. Pharma R. Retrieved from https://www.pharmar.org/
3. R/Pharma. Retrieved from http://rinpharma.com/
4. Winston Chang, Joe Cheng, JJ Allaire, Yihui Xie and Jonathan McPherson (2019). shiny: Web Application Framework for R. R package version 1.3.2. https://CRAN.R-project.org/package=shiny
5. Shiny from R Studio. Retrieved from https://shiny.rstudio.com/tutorial/written-tutorial/lesson3/
6. Yihui Xie, Joe Cheng and Xianying Tan (2019). DT: A Wrapper of the JavaScript Library 'DataTables'. R package version 0.8. https://CRAN.R-project.org/package=DT
7. David Hugh-Jones (2019). huxtable: Easily Create and Style Tables for LaTeX, HTML and Other Formats. R package version 4.6.1. https://CRAN.R-project.org/package=huxtable
8. Wikipedia contributors (2018). Rich Text Format. Wikipedia, the free encyclopaedia. https://en.wikipedia.org/w/index.php?title=Rich_Text_Format&oldid=51788903
9. Kevin Ushey, Jonathan McPherson, Joe Cheng, Aron Atkins and JJ Allaire (2018). packrat: A Dependency Management System for Projects and their R Package Dependencies. R package version 0.5.0. https://CRAN.R-project.org/package=packrat

ACKNOWLEDGMENTS

We would like to thank Satish Murthy, Paulo Bargo and his team for their efforts in developing and maintaining the infrastructure and their willingness to help and share their experience. We would also like to thank Sumesh Kalappurakal, Wilfred Willems, Olivier Leconte and the leadership team for their support, allowing us to focus some of our efforts towards this project. Lastly, a thank you to everyone within Janssen involved with the PSUR/DSUR integration and standardization.

CONTACT INFORMATION

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

Nicholas Masel
Janssen R&D US920 Route 202 Raritan, NJ 08869
Work Phone: 919-917-7690
Email: nmasel@its.jnj.com

Steven Haesendonckx
Janssen R&D Turnhoutseweg 30 B-2340 Beerse, Belgium
Work Phone: 0032 14 60 7774
E-mail: shaesen2@its.jnj.com

Alexandra Papadopoulou
Janssen R&D Turnhoutseweg 30 B-2340 Beerse, Belgium
Work Phone: 0032 14 60 5581
E-mail: ppapadop@its.jnj.com