

Bridging RStudio and LSAF: A Framework for Faster and Smarter Task Execution

Jake Adler, Alexion, AstraZeneca Rare Disease;
Ben Howell, SAS® Institute Inc.;
Lindsey Barden, Alexion, AstraZeneca Rare Disease

ABSTRACT

Open-source technologies are transforming today's clinical trials. Flexible open-source editors like RStudio are the perfect place to build dashboards that accelerate exploratory programming work. RShiny dashboards built on REST APIs connect directly to clinical data repositories. With available data, users can quickly create multiple TLFs and view them in real time. Any number of input parameter fields allow control over which studies, deliveries, domains, and variables are used. This paper will discuss how to create the dashboard, show examples of specific use cases where the dashboard helps save time, and offer inspiration for future enhancements to make the dashboard a more comprehensive, efficient clinical programming tool.

INTRODUCTION

In certain situations, visually interpreting data without the need to write extensive code is beneficial. The goal of this paper is to streamline the creation of tables, listings, and figures (TLFs) using RStudio. Our proposed solution is an RShiny dashboard that interfaces with a Clinical Data Repository (CDR) through REST APIs, enhancing data handling capabilities. The CDR used in this work is SAS Life Science Analytics Framework (LSAF). Building on the foundational work presented in 2023, where the initial version of the dashboard made individual REST API requests, this new version leverages those concepts to develop a customized interface. The dashboard allows users to create TLFs from clinical data within RStudio, offering enhanced interactivity through inputs that guide TLF creation, visual cues that indicate action performance, and real-time output previews, significantly improving user experience and productivity. This paper will cover the user interface (UI) code and server code that make up the application, an overview of the dashboard layout, example use cases, thoughts on traceability, and the benefits the dashboard provides.

USER INTERFACE CODE

RShiny application code has 2 components – the UI code and the server code. The UI code sets up input fields that feed into the server code as variables and allows for viewing various types of outputs from the server code. Figure 1 shows examples of the types of input objects used in the dashboard.

The figure shows a Shiny dashboard titled "Inputs" with five different input types arranged in a grid:

- textInput():** A text input field labeled "LSAF url" containing the text "https://lsafleeds.ondemand.sas.com".
- passwordInput():** A password input field labeled "LSAF password" with masked characters "*****".
- selectInput():** A dropdown menu labeled "Select a file to download:" with a list of files: "adae.sas7bdat", "adsl.sas7bdat", and "adv.sas7bdat". The first option is selected.
- checkboxInput():** A checkbox labeled "Group by variable" which is checked.
- actionButton():** A button labeled "Create figure" with a bar chart icon.

Figure 1. Types of Input Objects

Each of these input objects is associated with a variable that's used by the server code. For the text input example, the user types in their LSAF URL, and this becomes the value of a variable that's used to construct REST API calls. The password input field similarly sets the value of a variable, but it masks this value from displaying in the dashboard. The checkbox input type provides a true/false selection for a variable. Drop-down inputs ensure that users don't manually type incorrect values. The choices for the dropdown are initially empty and then populated later based on other dashboard actions.

The last type of input is an action button. The dashboard is grouped into sections based on the actions that can be performed, and when all of the inputs in one section are filled in, selecting the action button will run a server code function that uses those inputs to perform a task. When the server code function for an action completes, output objects display the results. Figure 2 shows examples of the types of output objects used in the dashboard.

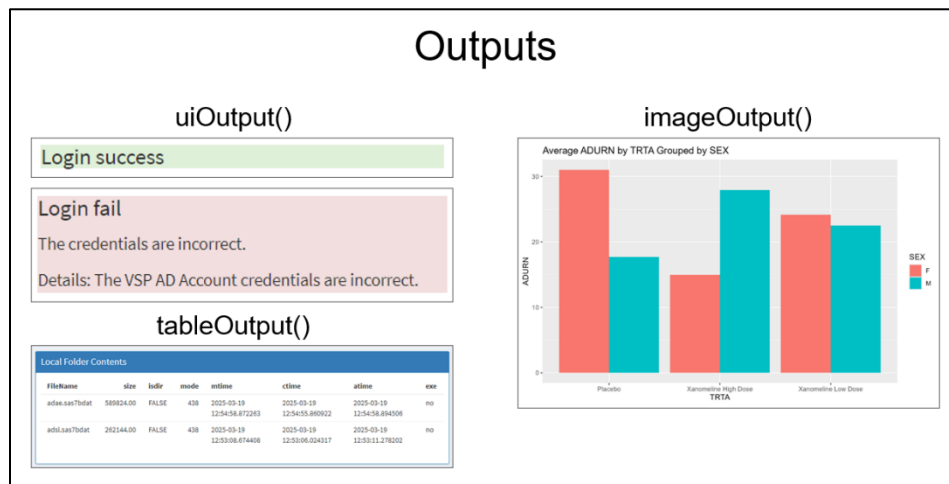


Figure 2. Types of Output Objects

The server code creates outputs, and the UI code simply displays these outputs. Formatted text is displayed with UI output objects (typically containing REST API response details), data frames are displayed with table output objects, and images are displayed with image output objects. The server code functions assign identifiers to each output, and the UI code uses the identifiers to place the outputs appropriately in the dashboard. Output objects are initially blank, and then each time a function runs, its output will be refreshed in the dashboard.

SERVER CODE AND DASHBOARD OVERVIEW

The UI code creates the display where information is entered. Meanwhile, the server code processes the UI responses in the background. HTTP requests like “GET()” and “POST()” are the primary functions that allow data to be sent and received through REST APIs, both of which are part of the “httr” R package. Outlined below is the process by which the server code works with the RShiny application.

LOG IN

Authentication

LSAF url

LSAF userid

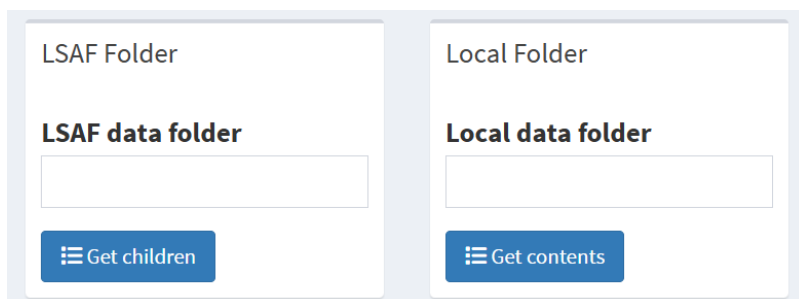
LSAF password

Login

Display 1. RShiny Application Log In Section

When the user clicks the “Log in” action button as shown in Display 1, it triggers an “eventReactive()” function to run the server code. The “GET()” function is then used to retrieve the encrypted password. From there, the “POST()” function sends the username and encrypted password to the “LSAF url”. If the log in is successful, the LSAF access token is pulled from the response header and used for future API requests. The “RenderUI()” function is used to identify whether the login was successful or unsuccessful.

LSAF & LOCAL FOLDER CONTENT

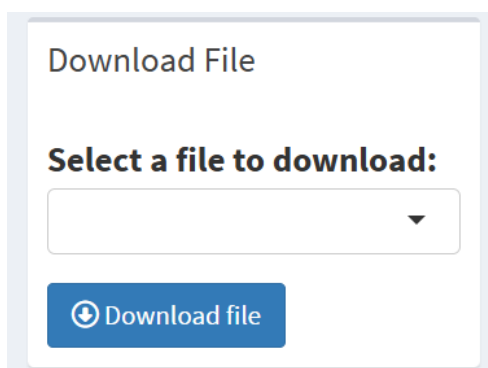


Display 2. RShiny Application Get Folder Information

Both the “Get children” and “Get contents” action buttons, as shown in Display 2, produce similar outputs. When “Get children” is selected, the “LSAF data folder” is concatenated with the original “LSAF url” from Display 1. Then, using the “GET()” function in combination with the access token received from the log in process, information about all data sets in the “LSAF data folder” is collected, such as output names and modified timestamps. Using the “datatables()” function within the “DT (Data Table)” R package, the information is displayed in the RShiny dashboard as an interactive table that you can scroll through.

When the “Get contents” action button is clicked, the path entered in the “Local data folder” is modified using the “chartr()” function to convert backslashes to forward slashes so it can be read in RStudio. The metadata is stored using the “file.info()” function and from there, the time variables are converted to character using the “as.character()” function. Using the cleaned metadata, another data table is developed for display in the RShiny application.

FILE DOWNLOAD



Display 3. RShiny Application Download File

Referenced in the step above, when the “Get children” action button is selected, the “updateSelectInput()” function updates the drop-down list with all the output names from the “LSAF data folder”, as shown in Display 3. Once a file is selected from the dropdown and “Download file” is clicked, the URL is constructed by concatenating the entered “LSAF url” (Display 1), “LSAF data folder” (Display 2) and the chosen file from the dropdown (Display 3). The file is pulled using the “GET()” function and is then written to the “Local data folder” from Display 2 using the “write_disk()” function. Note, if the file already exists in the local area, it will overwrite it. The UI will display the status as successful or unsuccessful depending on the status of the download.

RUNNING CODE TO GENERATE AN OUTPUT

Figure Settings

Select x variable:

Select y variable:

Select a dataset for your figure:

Select aggregation:

Average

☐ Group by variable

Select group by variable:

Local output folder

Figure name

Get variables

Create figure

Display 4. Create Output

The dropdown titled “Select a dataset for your figure:” (Display 4) uses the same steps as above. When “Get contents” is clicked (Display 2), the drop-down list appears with all the output names from the “Local data folder”. When “Get variables” in Display 4 is clicked, the data set is loaded to RStudio using the function “read_sas()” from the “haven” R package. This function reads SAS data sets and creates a data frame.

The “name()” function captures all of the column variables from the data frame and updates the “Select x variable:”, “Select y variable:” and “Select group by variable” dropdowns in Display 4. The “Select x variable” and “Select y variable” are always populated, while the “Select group by variable” can optionally be selected by checking the “Group by variable” box. The “aggregate()” function is used to produce summary statistics based on the selection made from the “Select aggregation:” dropdown and depending on the selection made, the respective output will be produced. Using the “ggplot2” R package, the aggregated data is plotted as a bar chart. The “aes_string()” function maps the “x” and “y” variables, while the “labs()” function adds titles and labels.

Using the “ggsave()” function, the figure is saved as a png file to the “Local output folder” as “Figure name.” The output is then posted to the RShiny application using the “renderimage()” function.

FILE UPLOAD

Local Folder

Local output folder

Get contents

LSAF Folder

LSAF output folder

Get children

Upload File

Select a file to upload:

Upload file

Display 5. Upload File

“Get contents” and “Get children” are clicked to view the refreshed folder metadata, as referenced in Display 5. The “Select a file to upload:” dropdown is updated with the “Local output folder” file names. While in previous steps a “GET()” function was used to retrieve information and download LSAF files, a “PUT()” function is now used to upload files to the “LSAF output folder”, an already existing path.

EXAMPLE

When the dashboard is opened, there is a main viewing window, as well as tabs down the left-hand side, as shown in Display 6. In the first tab of “Setup”, there are several boxes contained in the viewing window. The user first logs in to LSAF within the “Authentication” box using their specific URL and credentials to authenticate the connection between RStudio and LSAF. Next, the LSAF data folder pathway and local folder pathway are specified, and the “Get children” and “Get contents” action buttons are clicked. After selecting “Get children” in the “LSAF Folder” box, the user will see the “LSAF Folder Contents” table populated with all data sets available in the specified pathway, along with useful metadata, such as the last modified date. Similarly, selecting “Get contents” in the “Local Folder” box, the user will see the “Local Folder Contents” table refreshed. Before downloading files, the last modified date is useful to compare between the LSAF and Local folder contents to ensure the latest file is being used. Once the latest information is pulled, the user can then download available data files from LSAF using the drop-down menu in the “Download File” box. The user can download multiple files as needed, and then refresh the local folder by selecting “Get contents” again to confirm the files downloaded and are available to use. After each of the action buttons are clicked within the boxes displayed, the status is shown, as seen as successful in Display 6.

Authentication

LSAF url:

LSAF userid:

LSAF password:

Login success

LSAF Folder

LSAF data folder:

Get children success

Local Folder

Local data folder:

Get content success

Download File

Select a file to download:

Download success

LSAF Folder Contents

Show 10 entries Search:

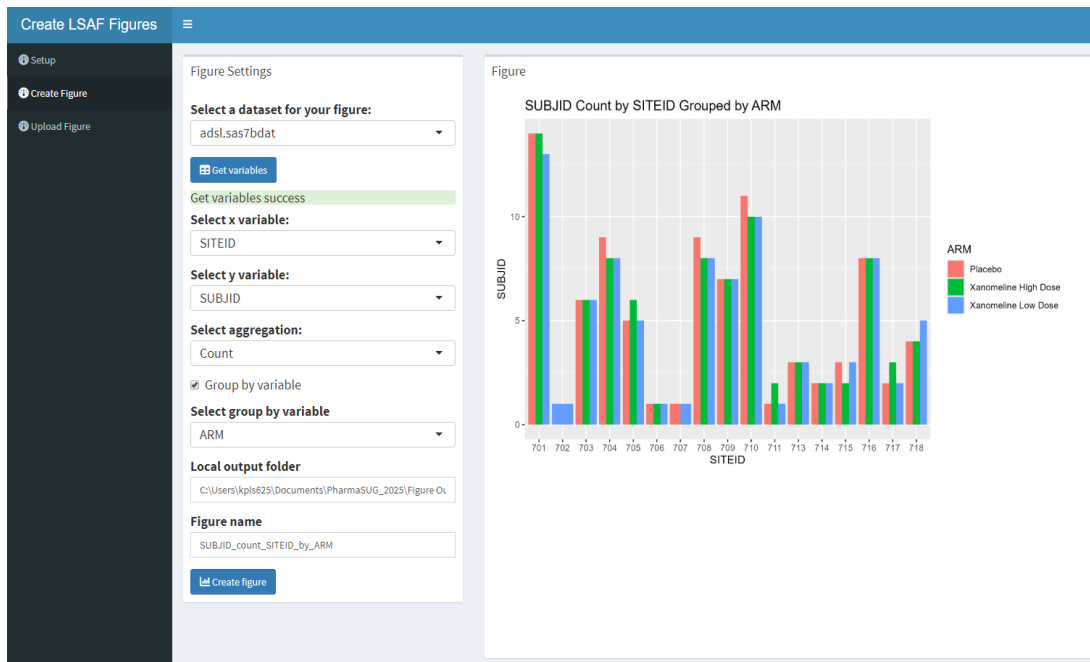
	schemaType	schemaVersion	typeId	id	name	path
1	file	1	sas:sasdataset	fca4167e-dc10-41df-9530-7ae2677c193f	adae.sas7bdat	/Users/kpls625/PharmaSUG 2025/Sample Data/adae.sas7bdat
2	file	1	sas:sasdataset	6de15115-90ce-436a-9ebe-ec6eac4785ac	adcm.sas7bdat	/Users/kpls625/PharmaSUG 2025/Sample Data/adcm.sas7bdat
3	file	1	sas:sasdataset	c80daf7d-1739-4e17-a6dc-e209220701d6	adsl.sas7bdat	/Users/kpls625/PharmaSUG 2025/Sample Data/adsl.sas7bdat
4	file	1	sas:sasdataset	22ce126e-f837-40f7-aa8a-98741648a743	ae.sas7bdat	/Users/kpls625/PharmaSUG 2025/Sample Data/ae.sas7bdat

Local Folder Contents

FileName	size	isdir	mode	mtime	ctime	atime	exe
adsl.sas7bdat	262144.00	FALSE	438	2025-03-19 12:53:08.674408	2025-03-19 12:53:06.024317	2025-03-19 12:53:08.690983	no

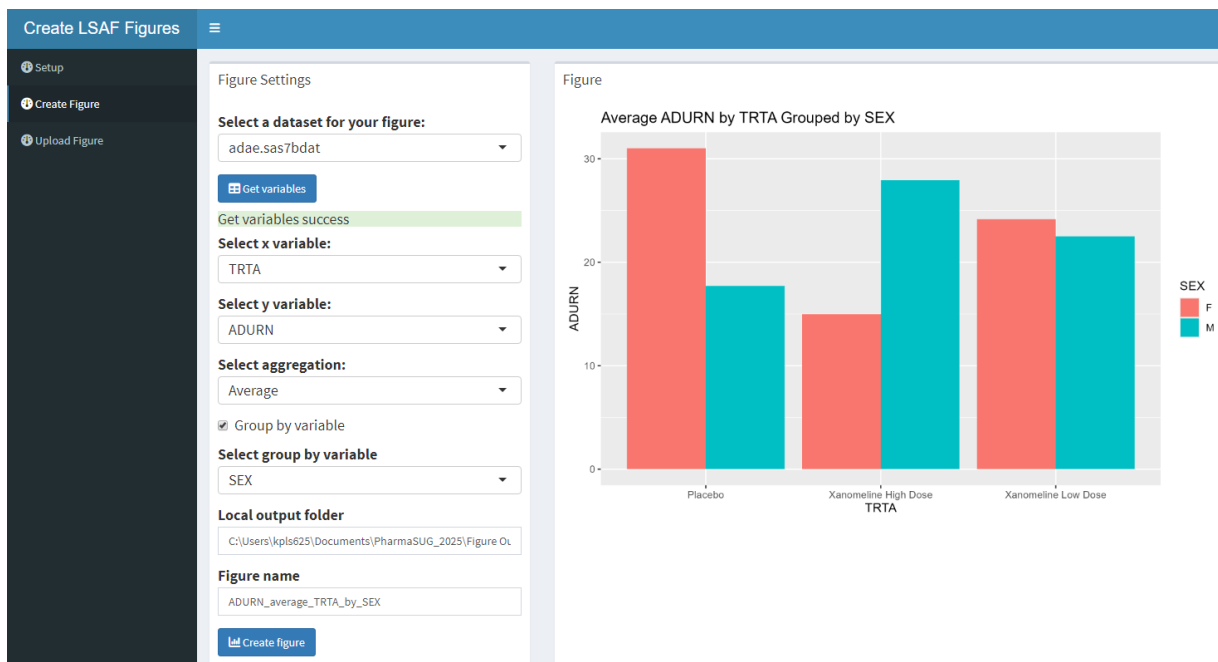
Display 6. Dashboard Interface for Setup Tab

In the second tab of “Create Figure”, the user can create figures using the previously downloaded data from the “Setup” tab. Using the drop-down menu and selecting ADSL shown in Display 7, the user can click “Get variables” within the “Figure Settings” box to see the available variables in the subsequent drop-down lists. For example, the user can select SITEID for the x variable, SUBJID for the y variable, Count as the aggregation function, click the “Group by variable” check box, and ARM as the group by variable. The local output folder pathway and figure name for the .png file are also specified. Once the “Create figure” action button is selected, the figure is created using the specified figure settings, output to the local pathway, and displayed in the viewing window. As shown in Display 7, a bar chart is created from ADSL showing the counts for each site (SITEID) using the distinct number of subject IDs (SUBJID), grouped by treatment variable (ARM).



Display 7. Dashboard Interface for Create Figure Tab Example 1

Using the “Select a dataset for your figure” drop-down menu, the user can select a different data set, such as ADAE, as shown in Display 8. Clicking “Get variables” refreshes the available variables to use in the figure settings. For example using ADAE, the user can select TRTA for the x variable, ADURN for the y variable, Average as the aggregation function, select the “Group by variable” check box, and SEX as the group by variable. The local output folder pathway and figure name for the .png file are updated for the new figure. Once the “Create figure” action button is selected again, the figure is created using the updated figure settings, output to the local pathway using the new figure name, and refreshed in the viewing window. As shown in Display 8, the bar chart is refreshed, created from ADAE showing the average adverse event duration (ADURN) for each treatment (TRTA), grouped by gender (SEX).



Display 8. Dashboard Interface for Create Figure Tab Example 2

Once the desired figures are created, the user can then open the third tab of “Upload Figure”, shown in Display 9. Similar to the “Setup” tab, there are several boxes with inputs or drop-down menus, as well as the local and LSAF folder contents tables. The user enters the local pathway where the figures are output, within the “Local Folder” box, and clicks “Get contents” to see the refreshed folder, which should contain the created .png files in the “Local Folder Contents” table. Next, the LSAF folder pathway is entered in the “LSAF Folder” box, to specify where the figure should be uploaded on LSAF. Selecting “Get children” will show the contents of the specified LSAF folder, along with the metadata of the files for reference. Lastly, the drop-down menu in the “Upload File” box, is used to select the figure file name. Once “Upload file” is clicked, the user is shown the status of the upload to LSAF and can continue to upload additional files as needed. The “Get children” action button can then be selected again to refresh the LSAF folder and see the newly uploaded figures within the “LSAF Folder Contents” table.

Create LSAF Figures

Local Folder

Local output folder
C:\Users\kpl625\Documents\PharmaSUG_2025\Figure Output

Get contents
Get content success

Local Folder Contents

fileName	size	isDir	mode	mtime	ctime	atime	exe
ADURN_average_TRTA_by_SEX.png	55900.00	FALSE	438	2025-03-19 13:04:31.89369	2025-03-19 13:01:43.161213	2025-03-19 13:08:25.699959	no
SUBJID_count_SITEID_by_ARM.png	71240.00	FALSE	438	2025-03-19 12:58:55.404081	2025-03-19 12:58:55.315836	2025-03-19 13:08:25.761049	no

LSAF Folder

LSAF output folder
/Users/kpl625/PharmaSUG_2025/Figure Output

Get children
Get children success

LSAF Folder Contents

schemaType	schemaVersion	typeId	id	name	path
1	1	sasfile	a2c649d5-1b2b-4413-9386-4c18e879435	ADURN_average_TRTA_by_SEX.png	/Users/kpl625/Pharm Output/ADURN_aven
2	1	sasfile	db7f9a7e-17f0-4063-b447-bf3748bb9d00	SUBJID_count_SITEID_by_ARM.png	/Users/kpl625/Pharm Output/SUBJID_cour

Upload File

Select a file to upload:
ADURN_average_TRTA_by_SEX.png

Upload file
Upload success

Display 9. Dashboard Interface for Upload Figure Tab

TRACEABILITY

The dashboard discussed in this work would primarily be used for exploratory analysis. LSAF maintains an Audit History of actions that occur in the repository. While using this dashboard, the Audit History records the download of data sets and upload of output files, but there is no record of the code that was run to create the outputs. To create TLFs for regulatory submission, there are some traceability gaps that should be filled.

The dashboard could be updated to create a log file that tracks which user ran the application, the values entered to the input fields, and date/time stamps of data downloads, output creation, and output uploads. LSAF Audit History records can be extracted using the LSAF API. The relevant Audit History records for the data sets and output files could be extracted and then combined with information collected in the application log to produce a manifest file. This manifest file could be stored in the LSAF repository for reference to show how various outputs were created. It would also be helpful to upload the application source code to the repository as well.

Data security is another consideration for using this dashboard. With the current version, data sets are downloaded from LSAF to be used by RStudio. Instead of running the application from your PC, you could run the application from a terminal server equipped with RStudio so that downloaded data sets won't be stored on users' laptops.

Alternatively, the process could be modified so that data is never downloaded and the code that creates TLFs resides in LSAF. The inputs in the dashboard would be used to pass parameters to LSAF jobs that would create TLFs in the repository instead of creating local TLFs from the RShiny server code. The built-in Audit History functionality in LSAF would track TLF creation without the need for extra logs from the RShiny application. The dashboard would be a user interface for LSAF job execution, and flexible jobs could allow non-programmers to create TLFs in LSAF by providing inputs through the dashboard. One disadvantage to this approach is that the user would no longer be able to view outputs in real time, but the traceability and data security would be greatly improved.

BENEFITS

The RShiny dashboard, which is seamlessly integrated with LSAF through REST APIs, provides a wealth of benefits that significantly enhance the workflow of data visualization in clinical research. One of its primary advantages is the interactive user interface, which is designed to engage users more effectively. This interactive aspect not only simplifies the navigation of complex data sets but also allows users to focus on data interpretation and decision-making rather than manual data handling. Furthermore, this allows the dashboard to be accessible to those without programming experience, which could enable cross-collaboration between different functional areas.

Moreover, the dashboard enhances data integrity and usability through controlled parameter selection. By replacing free text inputs with drop-down menus, it limits user inputs to predetermined options. This approach minimizes potential errors that arise from manual data entry, ensuring that the data input process is both efficient and reliable. Users can confidently select from pre-set parameters, enhancing data accuracy and consistency.

Quick execution is another standout benefit of this dashboard. It is engineered to handle tasks swiftly, drastically reducing the time required to generate TLFs. This speed is particularly beneficial in environments where timely data analysis is critical, enabling researchers to promptly respond to new data insights and trends.

Finally, the dashboard's design facilitates the easy repetition of actions across various studies. This feature is crucial for ensuring consistency in data handling, as it allows researchers to apply the same processes and standards to multiple data sets. By enabling the reuse of predefined workflows, the dashboard not only saves time but also upholds the quality and uniformity of outputs across different studies, thus enhancing overall research productivity and reliability.

CONCLUSION

This RShiny dashboard is an efficient way to explore clinical data and create TLFs. The dashboard's design is both flexible and structured, allowing you to create a variety of figures across multiple studies and view the outputs in real time, while preventing manual user error by populating drop-down menus with valid input values. Further improvements could be made to enhance the dashboard's traceability and data security, and while the example shown produces bar charts, the same framework could be used to create other types of visualizations.

REFERENCES

1. Elliott, L. and Cheng, C. 2023. "RESTful Thinking: Using R Shiny and Python to streamline REST API requests and visualize REST API responses" *Proceedings of the PharmaSUG 2023 Conference*, San Francisco, CA: PharmaSUG. Available at <https://www.lexjansen.com/pharmasug/2023/AP/PharmaSUG-2023-AP-063.pdf>

ACKNOWLEDGMENTS

We'd like to thank Laura Elliott and Crystal Cheng. This work was inspired by their initial version of the dashboard and developed using building blocks from their original application.

CONTACT INFORMATION

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

Jake Adler
Alexion, AstraZeneca Rare Disease
Jake.Adler@alexion.com

Ben Howell
SAS © Institute Inc.
Ben.Howell@sas.com

Lindsey Barden
Alexion, AstraZeneca Rare Disease
Lindsey.Barden@alexion.com

Any brand and product names are trademarks of their respective companies.