

## TLFQC: A High-compatible R Shiny based Platform for Automated and Codeless TLFs Generation and Validation

Chen Ling, Yachen Wang, AbbVie Inc.

### ABSTRACT

SAS-R integration and transition is a trending topic in the industry, more and more companies are incorporating R programming. Even though many R functions have been developed, Tables, Listings, and Figures (TLFs) generating still needs substantial repetitive work and the use of R for validating SAS-generated TLFs is very limited. Also, it might be hard for SAS users with no R experience to directly code in R. Taking advantage of the open-source software R, we are able to automate the process to make it easy to use for everyone, hence improving efficiency.

In this paper, we will introduce our application demo: TLFQC, which leverages the R Shiny framework to automate the generation and validation of standard TLF datasets in a Web app, with following features:

- TLFQC can generate multiple TLFs simultaneously with the required parameters entered
- Users can customize TLFs report with wording from table of contents (TOC), ensuring the final output adheres to requirements.
- The in-app interactive data exploration features enhance data review processes by streamlining data selection, filtering and sorting.
- The validation feature provides a comprehensive comparison between datasets generated by R and SAS for TLFs, providing an interactive dashboard for checking validation results intuitively as well as overall and detailed reports for enhanced QC
- TLFQC supports high compatibility, enabling other developers to include their own TLF-generating functions.

Along with these features, we will also introduce the basic structure of R shiny and building the skeleton of the app with {shinydashboard} package. Codes and examples will be shared in this paper, we will walk you through every detail for developing an interactive shiny app.

### INTRODUCTION

The generation and quality control of TLFs are critical tasks for statistical programming in the pharmaceutical industry, essential for ensuring the accuracy and clarity of data presented in clinical study reports and regulatory submissions. Traditionally, SAS is widely used for these tasks, however, it is inflexible to evolving needs and often requires significant manual efforts. Nowadays, an increasing number of companies are exploring the transition from SAS to R (Wang & Ling, 2025), driven by R's free open-source nature, versatility, and strong community support (Wang & Ling, 2025). There is a growing interest in leveraging various R toolboxes to automate and enhance the TLFs generation and validation process.

The purpose of this paper is to introduce TLFQC, a cutting-edge platform developed using the R Shiny framework, designed to automate standard TLF generation and validation to overcome the limitations of traditional SAS-based methods. In addition, the TLFQC enables users to perform a “batch run” for generating all TLFs and validating datasets with one click, which significantly reduces the time and efforts. Moreover, TLFQC platform provides seamless integration of user-defined TLF-generating functions, ensuring flexibility to accommodate diverse project needs.

The TLFQC platform consists of 2 parts, the TLF generation tab and the TLF validation tab. The TLF generation tab allows users to “batch run” multiple TLFs efficiently by uploading ADaM data and inputting parameters. An interactive data checking panel enables users to choose which TLF to display and dynamically filter, sort, and select data. Additionally, a download subtab allows users to customize TLF reports with Table of Contents (TOC) or manual inputs, facilitating the production of TLF reports. On the

validation side, TLFQC excels in its interactive capabilities, providing a thorough comparison between TLF datasets generated by R and SAS. Users benefit from an intuitive dashboard that presents validation results clearly and interactively, alongside comprehensive reports detailing both overall and specific validation aspects.

This paper not only explores TLFQC’s functionalities with example codes but also provides detailed instruction on how to build a user-friendly R shiny application from scratch. We aim to improve the efficiency of TLF generation and validation, and facilitate a smooth transition from SAS to R.

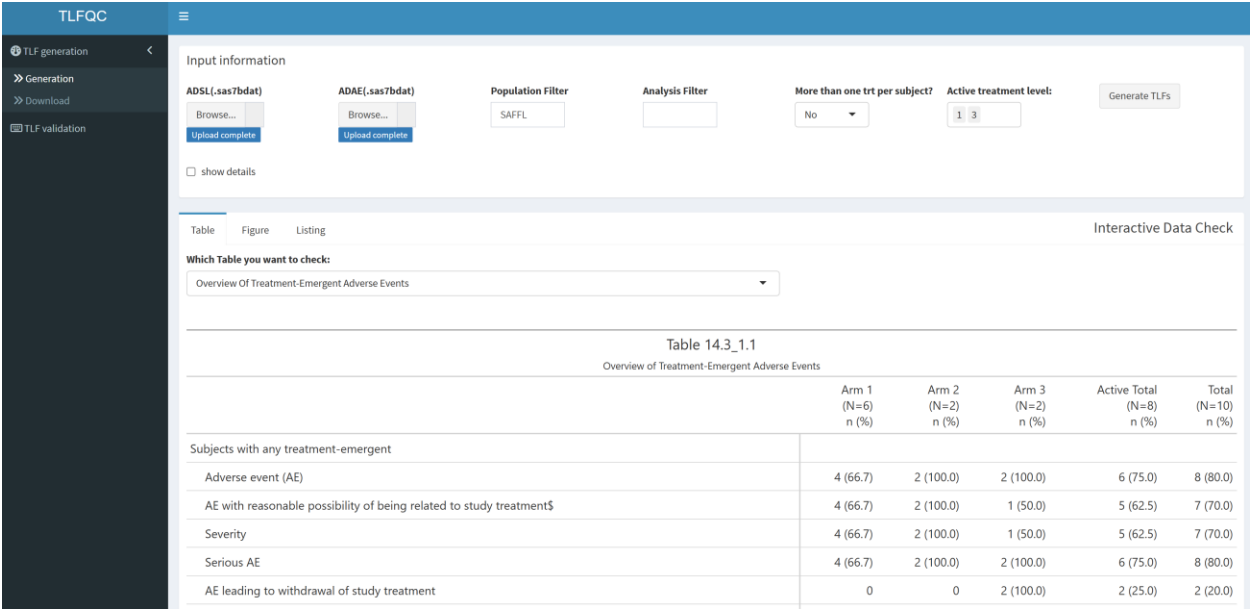
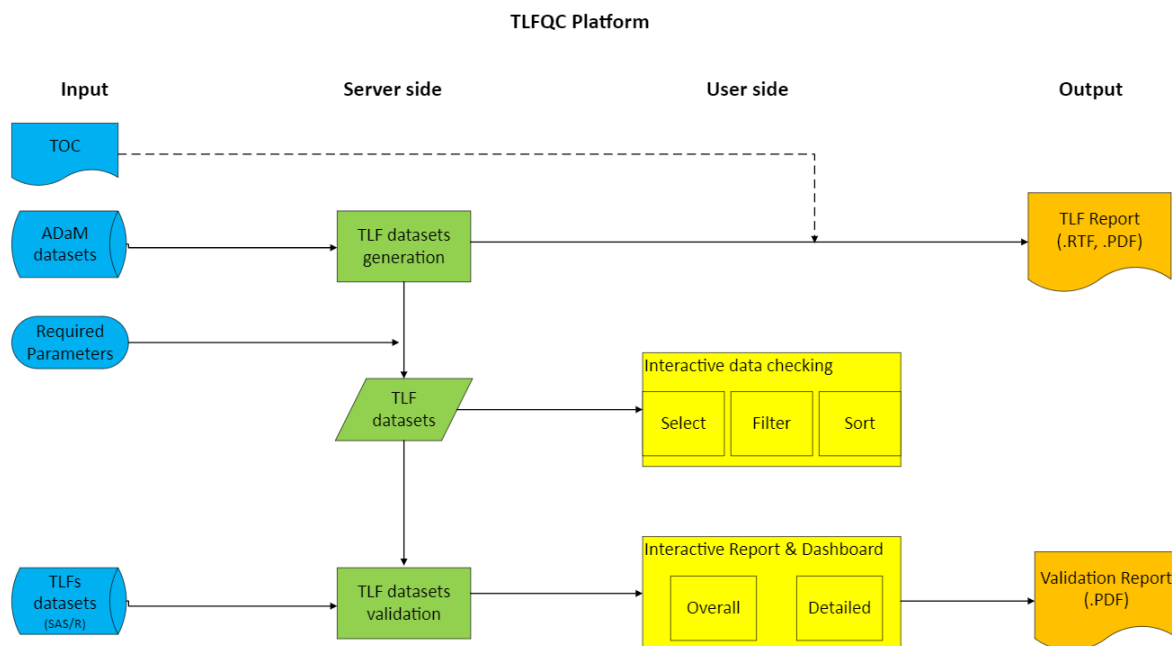


Figure 1. The user interface of TLFQC

THE FLOWCHART OF THE APP

The below flowchart presents a visual representation of the TLFQC platform’s workflow, illustrating the sequential actions in each process. During the TLF generation phase, the platform utilizes ADaM datasets along with specified input parameters to create TLF datasets. After the TLFs datasets are generated, users can engage in interactive data review tasks such as selecting, filtering, and sorting data. Additionally, users have the option to input a Table of Contents (TOC) to customize the output of the TLF report. For TLFs validation, TLFQC accepts user-provided TLF datasets and performs a comparison against internally generated TLF datasets. The platform features an interactive dashboard that displays overall validation status and detailed validation status for each dataset, allowing users to efficiently review the outcomes. Moreover, TLFQC can output comprehensive validation reports, providing insights into

both overall and detailed validation results.



**Figure 2. Flowchart of TLFQC**

## BASIC STRUCTURE OF R SHINY AND SHINY DASHBOARD

Like many web applications, shiny applications have a front end (user interface or ui) and a back end (server). Users can interact with the ui, then the server will run codes and return the results to the ui. Both parts can be separated into a single file, named ui.R and server.R, also they can be combined in a program named app.R, as shown in R program 1.

```
library(shiny)
ui <- fluidpage(
)
server <- function(input,output){
}
shinyApp(ui = ui, server = server)
```

### R program 1. Basic structure of R shiny app

In our app, we use {shinydashboard} package to generate the skeleton of the app. This package is particularly useful for adding a polished look to R shiny app without in-depth knowledge in R shiny or HTML/CSS. It provides an easy-to-use template comprising a header, a side bar and a body (APPENDIX FIGURE 1). In R program 2, the `dashboardHeader` function allows you to set the app's header, positioned at the upper left corner. The left sidebar can be created using the `dashboardSidebar()` function, where the menu can be customized with `sidebarMenu()`. The `dashboardBody()` function is used to manage the main content of the app. For example, you can add a title using functions `h1()` to `h6()`, which originate from HTML and are used to create a hierarchy of subheadings with decreasing importance. Additionally, the `box()` function can be utilized to add a container for displaying results, such as tables and figures.

```
library(shiny)
library(shinydashboard)
#UI
ui <- dashboardPage(
  dashboardHeader(title="my shiny dashboard"),
  dashboardSidebar(
    sidebarMenu(
```

```

        menuItem("Dashboard",tabname="dashboard"),
        menuItem("Widgt",tabname="widgt")
    ),
    dashboardBody(
      tabItem(tabName="dashboard",
        h3("This is a demo shiny dashboard"),
        box(title="This is a box"),
        box(title="This is another box")
      )
    )
  )
)
#server
server <- function(input, output) { }

shinyApp(ui, server)

```

## R program 2. Shiny dashboard basic structure

## TLFS GENERATION (TOGETHER WITH UI & SEVER CODES)

### 1. DATA UPLOAD

In TLFQC, there are two types of datasets to upload to the app: one is the ADaM datasets for generating TLFs, the other is the TLFs datasets for validation against those generated by TLFQC. To prepare ADaM datasets for use, certain study-specific variables must be populated. For example, if users want to include some AE of special interests into AE overall table, they need to set flags for those records in ADAE. In this demo, only AE related TLFs are included, thus only ADSL and ADAE will be uploaded.

Generally, the files can be uploaded with `fileInput()` function on the UI side. This function reads the file information, including the path. By default, users can only select one file in this step, while `multiple=TRUE` option enables users to choose and upload several files in a single action.

```
fileInput("adae", "ADAE(.sas7bdat)", multiple=TRUE)
```

## R program 3. Data upload in UI side

At this point the data has not been read in the app yet. We need to use the R program 4 on the server side to read in the data. Herein, the `infile` stores all information of the uploaded ADAE, and then `read_sas()` is used to read in the data. One thing we need to notice is that, in R shiny server we use reactive programming to specify dependencies, so that when an input changes, all the related outputs will be updated automatically. So, in R program 4, the `adaeinput` is a dynamic variable that will change with the uploaded ADAE data, we need to use `reactive({})` to enclose the codes to make it “reactive”.

```

adaeinput <- reactive({
  infile <- input$adae
  if (is.null(infile)) {
    return(NULL)
  }
  adae<-read_sas(infile$datapath)
})

```

## R program 4. Data upload on server side

### 1.1 Read in different types of data

Apart from SAS datasets, there are other types of data that TLFQC can read, such as .csv, .xlsx and .xpt data. The `datapath` syntax saves the path of the input files as a list. By extracting the file type from the paths, we can apply the appropriate function to read the file effectively. The program below is an example of how to read multiple .xpt file, and how to save input .xpt datasets in a list named `bdata_list`.

```

#server
base<-input$basedata
basetype<- sub(".*\\."," ", base$datapath) #extract file type

```

```

if (basetype %in% c("xpt")) {
  for(i in 1:nrow(base)) {
    bdata_list[[base$name[i]]] <- read_xpt(base$datapath[i])
  }
}

```

## R program 5. Multiple datasets upload with specified data type

## 2. PARAMETERS INPUT

In the last section, we introduced how to upload datasets to the R shiny app, next we would like to show how to input values to the app. In TLFQC, the mostly used input information are the variable names which are used to manipulate ADaM datasets, such as Subject Identifier in Figure 3. To pass this information to the app we need to use the `textInput()` function in UI, as shown in R program 6, the first argument "id" is the id of this input, which would be used as a reference when it is called in the server. The second argument "Subject identifier" is the displayed name in the UI, and the third argument `value="USUBJID"` assign "USUBJID" as the default value for this input box. The last argument `width="100px"` sets the width of this input box to be 100px.

```

textInput("id", "Subject identifier", value="USUBJID", width="100px")

```

## R program 6. Variable names input on UI side

On the server side, the input values can be called by using `input$input_id`, and normally no further steps are needed. However, when the text input is used as a variable name in a function, there might be an issue because text input is a string which cannot be used as a variable with `filter()`, `group_by()` and all other **{dplyr}** functions. When using `id` in R functions, it must be converted into a symbol that R can evaluate. As shown in R program 7, the `sym()` function is used to convert a string to symbol, and `!!` (called bang-bang) is used to unquote the symbol. It's more efficient to implement this conversion directly inside the table generating function (`aeoverall` in this example), it's recommended to consider this at the beginning of developing an R function (Ling & Wang, 2025).

```

aeoverall(adsl=adslinput(),
          adae=adaeinput(),
          id=!!sym(input$id),.....)

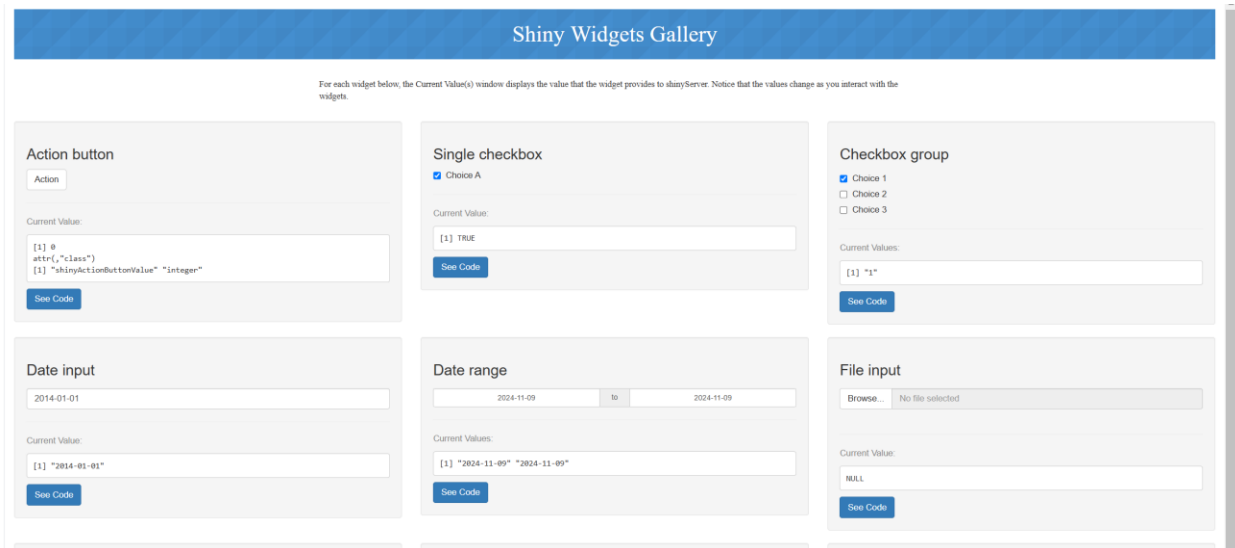
```

## R program 7. Example function to demonstrate using a string as an argument

Figure 3. The input information tab

## 2.1 Different types of input

In R shiny, interactive web elements that allow users to send messages to the server are known as control widgets. The mentioned features like data upload and parameter input are examples of such widgets. Besides these, there are numerous other types of control widgets, as depicted in Figure 4. In the following section, we will explore some types of widgets utilized in TLFQC.



**Figure 4. Shiny Widgets Gallery (cited from <https://shiny.posit.co/r/gallery/widgets/widget-gallery>)**

### 2.1.1 Select box

The `selectInput()` function generates a selection list that allows users to choose one or more items from a set of values. The first argument, “inputId”, is mandatory and is used to access the selected value via “input\$inputId”. The argument “multiple = FALSE” specifies whether multiple items are allowed. The R program 8 offers a single option for users to choose the output style, “outstyle” is the inputid, “Output Style” is the label, and the available options are “PDF” and “RTF”.

```
#UI
selectInput("outstyle", "Output Style",list("RTF","PDF"),multiple = FALSE)
#server
rpt <- create_report(getwd(), output_type = input$outstyle)

Output Style
RTF
RTF
PDF
```

**R program 8. Select input**

### 2.1.2 Selectize Input

The `selectizeInput()` function looks similar to `selectInput()`, however, they don't function in the same way. The key distinction lies in the `options` argument in `selectizeInput()`, which would allow you to enter the input as a list. By setting `options = list(create = TRUE)`, users can type in the elements, and `selectizeInput()` can return a list containing those elements.

```
#UI
selectizeInput("tx_active", "Active treatment level:",
               choices = NULL,
               multiple = TRUE,
               options = list(create = TRUE),width="100px")

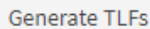
Active treatment level:
1 3
```

**R program 9. selectize input**

### 2.1.3 Action button

The `actionButton()` function creates an action button on a page, which works with `observeEvent()` to trigger a command. As shown in R program 10, `observeEvent()` monitors when users click the button and subsequently triggers the action items enclosed in `{}`.

```
#UI
actionButton("Generate", "Generate TLFs")
#server
observeEvent(input$Generate,{
  #action 1
})
```



**R program 10. Action button**

## 2.2 Hide and display parameters

As you might notice in Figure 3, there is a checkbox named “show details”, allowing users to toggle the visibility of additional parameters. All the required inputs are prefilled with the default value, hiding them can make the UI more concise. The `hide()` and `show()` functions can dynamically hide or show a defined object in shiny. To hide multiple inputs at once, we can create a div element using `div()`. The `<div>` is a fundamental HTML element which is a container for organizing the page layout or styling purposes. Herein, we use it as a container of multiple inputs to dynamically update the visibility. In R program 11, the div block “text\_inputs” is hidden by default, until user checks the checkbox.

```
#UI
checkboxInput("show_detail","show details"),
div(id="text_inputs",
  textInput(id,"Subject identifier:", value="USUBJID",width="100px"),
  textInput("tx_var", "Treatment variable (N):",value="TRT01AN",width="100px"),.....
)
#server
hide("text_inputs")
if (input$show_detail){
  show("text_inputs")
}
```

**R program 11. Hide and display parameters**

## 3. TLF DATASETS GENERATION

Now we have all the required inputs (datasets and parameters). The next step is to generate the datasets for creating TLFs with those inputs. The most efficient method to avoid repeated programming is to create R function for generating TLF datasets (Ling & Wang, 2025). Those R functions can be stored in separate R scripts in the same location as the application and can be included in the application by using `source("R_function.R")`. Like the uploaded ADaM datasets, we can create a dynamic variable to store the generated dataset, as shown in R program 12, `aeov_final` is used to store the data generated by R function `aeoverall()`. More details regarding the R functions are introduced in another paper presented in this conference (Ling & Wang, 2025).

```
source("aeoverall.R")
server<-function(input,output){
  .....
  aeov_final<-reactive({
    aeov<-aeoverall(adsl=adslinput(),
                    adae=adaeinput(),
                    id=input$id,
                    tx_var=input$tx_var,.....)
  })
  .....
}
```

R program 12. Example of TLF datasets generating function

This elegant design endows the TLFQC platform with high compatibility for various TLF datasets generating functions. In this demo, we only included AE related TLFs; however, users can easily incorporate their own custom R functions into the TLFQC platform, promoting sustainability by adapting to evolving needs.

4. TLFs DISPLAY

In the previous sections, we discussed methods for sending “messages” to TLFQC and generating datasets with those “messages”. Next, we will explain how to output results and display them dynamically within TLFQC. Once users click the generate button, the system executes all the R functions, generating the necessary datasets in the backend. To view the TLFs, users simply select the desired TLFs from a selection box. The chosen TLFs are then displayed below, as illustrated Figure 5. With the filtering, selecting and sorting options, users can interactively explore and analyze the data.

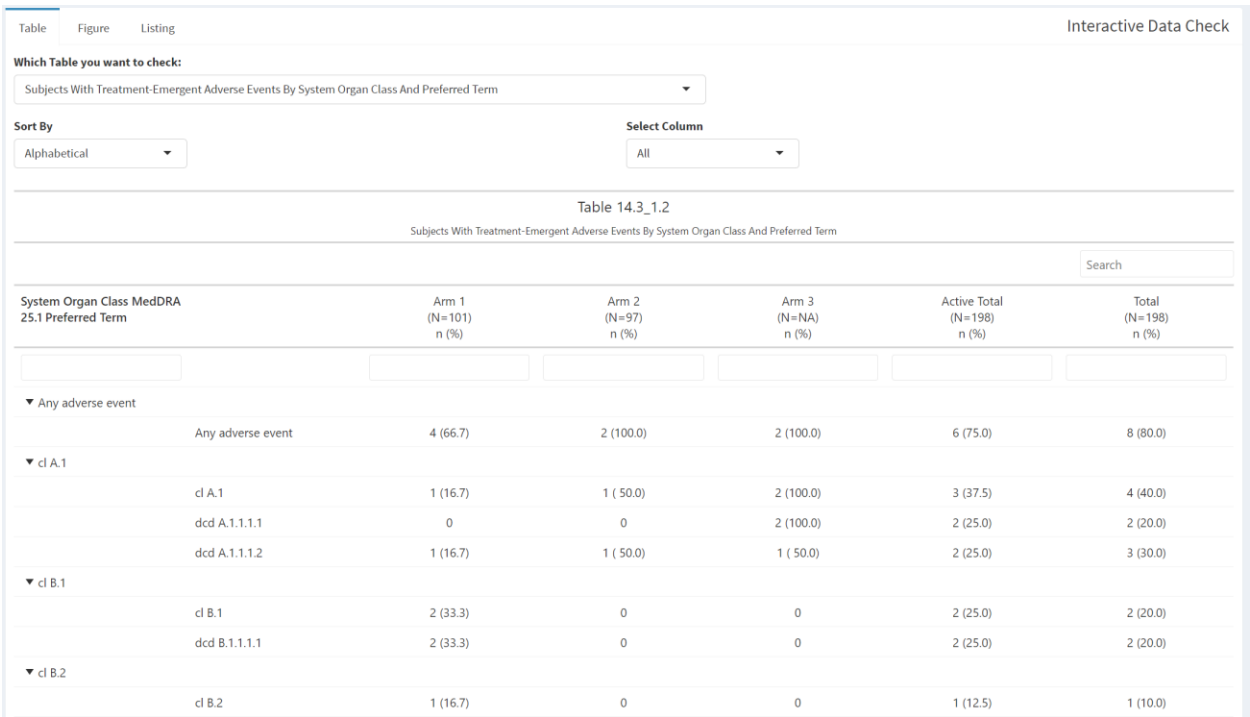


Figure 5. TLFs display and interactive data check

4.1 Display tables listings and figures

The {gt} package is designed to make wonderful-looking tables with a cohesive set of table components. Unlike general data tables, {gt} tables can incorporate advanced features such as headers, labels, color enhancements and interactivity in the shiny application. The render\_gt() and gt\_output() functions work together to incorporate {gt} output into the UI, where gt\_output() serves as a placeholder and render\_gt() renders the gt table and places the results in the placeholder. This is a common method for displaying back-end results to the front-end user, we will introduce displaying plots with plotOutput() and renderPlot() later.

In R program 13, the function t\_ae03\_gt() is used to create a gt table which is then rendered and displayed in the UI. The function t\_ae03\_gt() is stored together with the table generating function t\_ae03() in a single source program, making it convenient to manage functions and maintain the application. Please note that if the label of the column has line breaking sign (" /n"), we should use .fn=md inside cols\_label() to help us render from Markdown.

```
#UI
```



```

library(gt)
library(labelled)
gt_output("tbl")
#server
output$tbl<-render_gt(t_ae03_gt(t_ae03_data))
#In source program
t_ae03_gt<-function(data, columns){
gt_table<-gt(data=data, groupname_col = "ROWLBL1", rowname_col = "ROWLBL")%>%
tab_header(
  title="Table 14.3_1.2",
  subtitle="Subjects With Treatment-Emergent Adverse Events By System Organ Class And
Preferred Term"
)%>%
  tab_stubhead(label = "System Organ Class\n      MedDRA 25.1 Preferred Term\n      ") %>%
  cols_label(.list=var_label(data),.fn=md)%>%
  opt_interactive(use_search = T,use_filters = T, use_resizers=T,
use_page_size_select=T)%>%
  cols_hide(columns = !!cols)
}

```

### R program 13. Data display with {gt} package

In TLFQC, while tables and listings are displayed with gt table, the plots are displayed with the help of {ggplot2} package, which is part of the {tidyverse} family. Similar to render\_gt() and gt\_output() in {gt} package, plotOutput("plot") is used as the placeholder for plots, and renderPlot() is used to render the plot. The function plot1\_final() is from the source program, and it is capable of creating a ggplot object.

```

#UI
library(tidyverse)
library(labelled)
plotOutput("plot")
#server
output$plot<-renderPlot(plot1_final(adaeinput()))
#In source program
plot1_final <- function(data){
  plot<-data%>%distinct(USUBJID,TRTA,AEBOODSYS)%>%
    ggplot(aes(AEBOODSYS)) +
    geom_bar(aes(fill = TRTA))+
    coord_flip() +
    geom_text(stat = 'count', aes(label = after_stat(count),group=TRTA), position =
position_stack(vjust = 0.5))
  return(plot)
}

```

### R program 14. Plot display with {ggplot2} package

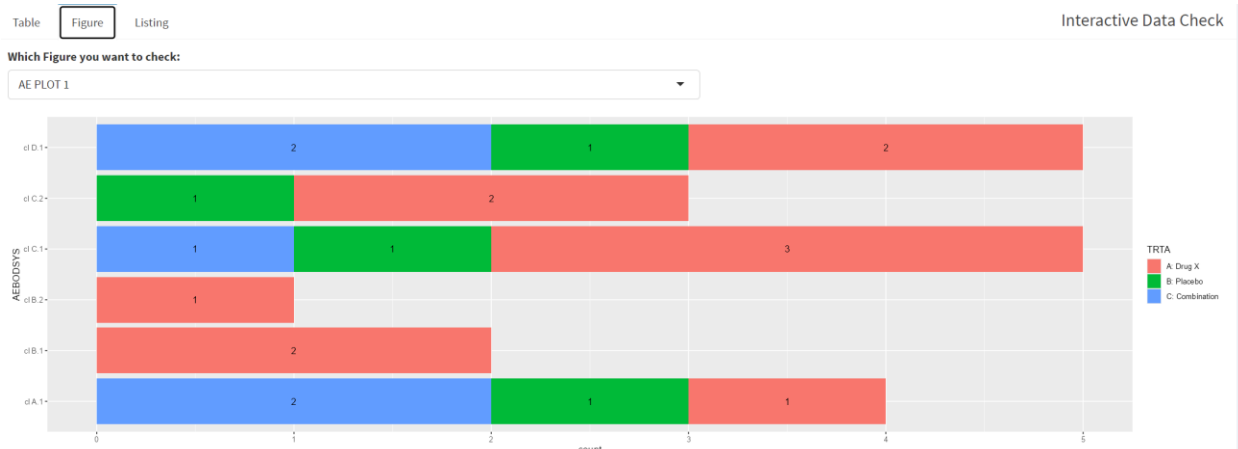


Figure 6. Plot display in TLFQC

## 4.2 Interactive data checking

Interactive data checking allows users to filter, select and sort data. In TLFQC, there are two ways to interact with the data in the table. First, we can simply filter and sort the data with the embedded function in `{gt}`; Second, two select boxes have been created to allow users to select specific columns and to sort them either alphabetically or in descending order.

In R program 13, utilizing `opt_interactive(use_search = T, use_filters = T)` argument in `{gt}` allows users to interact with data effectively. The text box under System Organ Class (SOC) can be used to filter adverse events based on a specific SOC. Additionally, the search box under the titles on the right enables users to search for a particular adverse event by its preferred term.

R program 15 is an example of column selection in the output table. The `selectInput()` function allows specific columns to be selected before rendering. The following code snippet generates the list of columns to be hidden in the `gt` table. The function `cols_hide(columns = !!cols)` in R program 13 is used to hide these columns in the output interactive table. Note that `cols_hide(columns = NULL)` has been deprecated since `gt` v0.3.0, so we need to use `-everything()` to replace `NULL` in this case.

```
#UI
selectInput("colshow", "Select Column ", list("All", "Treatment", "Active Total", "Total"))
#server
if ( "Treatment" %in% input$colshow ){columns <- quo(c(TRT98, TRT99))}
else if ( "Total" %in% input$colshow ){columns <- quo(c(!TRT99))}
else if ( "Active Total" %in% input$colshow ){columns <- quo(c(!TRT98))}
else if ( "All" %in% input$colshow ){columns <- quo(-everything())}
```

**R program 15. Column selection for displaying in `gt` table**

## 5. REPORT GENERATION

In the previous section, we introduced the in-app display of the TLFs. However, the ability to customize their format is limited, and they do not always comply with company standards. In this section, we will introduce how to customize the TLF reports based on the information from user input or the table of content (TOC), which holds wordings like title and footnotes for each TLF.

The screenshot displays the TLFQC application interface. On the left is a dark sidebar with navigation options: 'TLF generation' (selected), 'Generation', 'Download', and 'TLF validation'. The main content area is divided into two sections. The top section, 'Study Information', contains input fields for 'Compound Number', 'Study Number', and 'R&D Number'. To its right is the 'TOC upload' section, which includes a 'TOC UPLOAD (excel, csv)' button, a 'Browse...' button, a 'No file selected' status, an 'Output Style' dropdown set to 'RTF', and a 'Download Output' button. The bottom section, 'TLFs Information', features a grid of controls: 'Title' (dropdown with 'Overview Of Treatment-Emergent Adverse Events'), 'Output Type' (dropdown with 'Table'), 'Section Number' (dropdown with '14.3'), 'Footnotes(separate by |)' (input field), 'Population' (dropdown with 'Safety Analysis Set'), 'Output Number' (input field with '1'), 'Output Style' (dropdown with 'RTF'), and another 'Download Output' button.

### 5.1 Customize report with manual input information

Apart from interactive checking the table, users can customize headers, titles, and footnotes, as well as download table reports in RTF or PDF format. Herein, we would like to introduce the `{reporter}` package in `{sassy}` (Bosak, 2024), which is comparable to PROC REPORT in SAS, and capable of generating similar reports. In R program 16, the `create_table()` function is used to create a table which is then inserted into the report. Within `create_report()`, several customizations are possible:

- The header can be customized using `page_header()` with all study level information.
- Titles are customizable with user input, including study number, title and population,

- Then the table created earlier is inserted into the report.
- The footnotes can be customized with `footnotes()`, utilizing the information from `input$ftnt`.
- The program path is added to the footer using `page_footer()`.

```
#server
tabel <- create_table(df2, first_row_blank=TRUE, borders = "top", width=9)%>%
  column_defaults(from="TRT1", to = "TRT99", width=6/n)%>%
  column_defaults(vars = "ROWLBL", width=3)%>%
  define(pg, visible = FALSE, page_break = TRUE)

report <- create_report(temp_location, output_type = input$outstyle, font = "COURIER",
font_size = 8) %>%
  page_header(left = c(paste0(Sys.time(), "<TLFQC.R>"),
    input$compoundno,
    paste0("STUDY ", input$studyno, " CLINICAL STUDY REPORT"),
    paste0("R&D/", input$rdnum, " - CLINICAL/STATISTICAL"),
    "Page [pg] of [tpg]")) %>%
  titles(paste0(input$outtype, " ", input$secnum, "_", input$outputnum),
    input$title, input$popu, borders = "bottom") %>%
  add_content(tabel) %>%
  footnotes(str_replace_all(input$ftnt, "\\|", "\\n"), borders = "top", blank_row =
"none", valign="top") %>%
  page_footer(paste0("Program source code:", getwd()))
```

## R program 16. Create reports with user inputs

The reports can then be generated by `write_report()` and downloaded by `downloadHandler()`, please see an example report in APPENDIX TABLE 1.

## 5.2 Customize report based on TOC

With user input, only one report can be generated at a time. If users wish to generate multiple reports, they need to re-enter table information, such as headers, titles, and footnotes. To increase efficiency in report generation, users can also upload their table of contents (TOC) to provide information to TLFQC. TLFQC will first match the table-generating functions using the Table Function Name column in the TOC. While study-level information in headers remains unchanged, TLFQC uses the information from TOC in the `create_report()` function to generate the report. Please see R program 17 as an example, where TOC information is used for generating titles in the report.

```
#server
report <- create_report(...) %>%
...
titles(paste0(toc[["Output Type"]], " ", toc[["Section Number"]], "_", toc[["Output
Number"]]), toc[["Title"]], toc[["Analysis Population Title"]], borders = "bottom") %>%
...

```

## R program 17. Create report title with TOC information

Output Type	Section Number	Output Number	Title	Analysis Population Title	Footnotes	Table Function Name
TABLE	14.3	1.1	Overview Of Treatment-Emergent Adverse Events	(Safety Analysis Set)	Arm 1: Dummy drug X  Arm 2: Placebo Arm 3: Dummy drug Y Note: Subjects are counted once in each row, regardless of the number of events they may have had. \$ As assessed by investigator.	T_AE01
TABLE	14.3	1.2	Subjects With Treatment-Emergent Adverse Events by System Organ Class And Preferred Term	(Safety Analysis Set)	Arm 1: Dummy drug X  Arm 2: Placebo Arm 3: Dummy drug Y Note: Subjects are counted once in each row, regardless of the number of events they may have had.	T_AE03

**Table 1. Example table of content**

## 5.3 Download

The `downloadButton()` function in UI creates a download button, it will initiate a browser download when clicking it. The `downloadHandler()` function defined in the server function, it allows users to download content from R shiny. Generally, `filename` and `content` option need to be defined in this function.

`filename` includes the name of the file and its type, `content` only has one argument "file", it passes the content of the file path to the output document. In R program 18, we can download an rtf report with name defined by the input table number, and the content is a report object created by `create_report()` from `report2()`. `write_report` function can be replaced by other functions, such as `write_csv`, `write_excel_csv`, etc to output different types of documents.

```
#UI
downloadButton("downloadr2", "Download Output")
#server
output$downloadr2 <- downloadHandler(
  filename = function() {paste0( input$secnum,"_",input$outputnum,".rtf")},
  content = function(file){
    write_report(report2(),file)
  }
)
```

### R program 18. Download a report

However, this method won't work if we want to download multiple TLF reports simultaneously. To download multiple TLF reports, we need to compress them into a zip file. The process involves using `create_report()` and `write_report()` to generate each TLF report at a temporary location for each TLF that matches the table of contents (TOC). Then inside `downloadHandler()`, the `zip()` function can be used to compress all files from the temporary location and download them as a zip file.

```
#server
output$downloadr1 <- downloadHandler(
  filename = "reports.zip",
  content = function(file) {
    .....
    files_to_zip <- list.files(path = temp_directory,
                             pattern = paste0(".", input$outstyle), full.names = FALSE)
    zip::zip(
      zipfile = file,
      files = files_to_zip,
      root = temp_directory
    )
  },
  contentType = "application/zip"
)
```

### R program 19. Download multiple reports in a zip file

## TLFS VALIDATION

The other very important feature of TLFQC is TLF validation. In this section we introduce how to check the discrepancy between base (from SAS) and validation datasets (from TLFQC), and how to present the results in R shiny.

### 1. VALIDATION WITH {DIFFDF} PACKAGE

The `{diffdf}` package is designed to offer functionality similar to PROC COMPARE for secondary programming and review in R (Gower-Page & Martin, 2024). Like SAS, the results from `{diffdf}` include differences in columns, rows, attributes, values, etc., between two datasets. In TLFQC, we use these comparison results and create both an interactive dashboard and downloadable reports.

To determine if two datasets are fully matched, we can use `diffdf()` function in `{diffdf}` package. The `diffdf()` function compares datasets by evaluating several aspects, such as column numbers, row numbers, row id, values, column types, and column labels. Although it also supports comparison of factor

levels, this feature is disabled in TLFQC. To check if two datasets are matched, `diffdf_has_issues()` function can be used, and `FALSE` will be returned if no discrepancies are found.

```
#server
difference<-diffdf(basedata(),validata(),suppress_warnings = T,strict_numeric =
FALSE,strict_factor = FALSE)
diffdf_has_issues(difference)
```

## R program 20. Compare datasets using diffdf()

In R program 21, you can check for specific comparison results, such as column labels, by subsetting the comparison results with 'AttribDiffs'. This is done using the expression

`is.null(difference()[["AttribDiffs"]][[1]]) == TRUE`. Apart from “success” and “danger”, the `lbl()` function will also return a status called “warning”, indicating that the user has manually chosen not to compare labels by checking the checkbox. The `attribdiff` is used to hold information about label mismatches, which will be included in the box in UI, when mismatches are found. Whereas for the other scenarios, the box would show “PASS!” and “Not Evaluated!” for `lbl()` equals “success” and “warning” respectively.

```
#server
###Determine if two datasets have different labels
lbl<-reactive({
  if (is.null(difference()[["AttribDiffs"]][[1]])==TRUE ) {
    if (input$checklbl==TRUE) {
      status="success"
    }else {
      status="warning"
    }
  }else{
    status="danger"
  }
  return(status)
})
###Get a comparison table between two datasets
attribdiff<-reactive({
  attri<-data.frame(difference()[["AttribDiffs"]])%>%
    mutate(across(everything(),~ifelse(.x=="NULL",NA,.x)))
  attril<-attri[,-2]
  names(attril)<-c("Variable","Base","Compare")
})
###Assign values to the box
if (lbl()=="success") {
  output$passlabel<-renderText({"PASS!"})
  output$label<-NULL
  output$labeltbl<-NULL
}else if (lbl()=="warning") {
  output$passlabel<-renderText({"Not evaluated!"})
  output$label<-NULL
  output$labeltbl<-NULL
}else{
  output$passlabel<-renderText({"FAIL!"})
  output$label<-renderText({"Variables with different labels"})
  output$labeltbl<-renderTable({attribdiff()})
}
```

## R program 21. Check column types comparison results

## 2. OVERALL STATUS DASHBOARD & REPORT FOR ALL COMPARED DATASETS

The first tab in TLF Validation page is overall status (

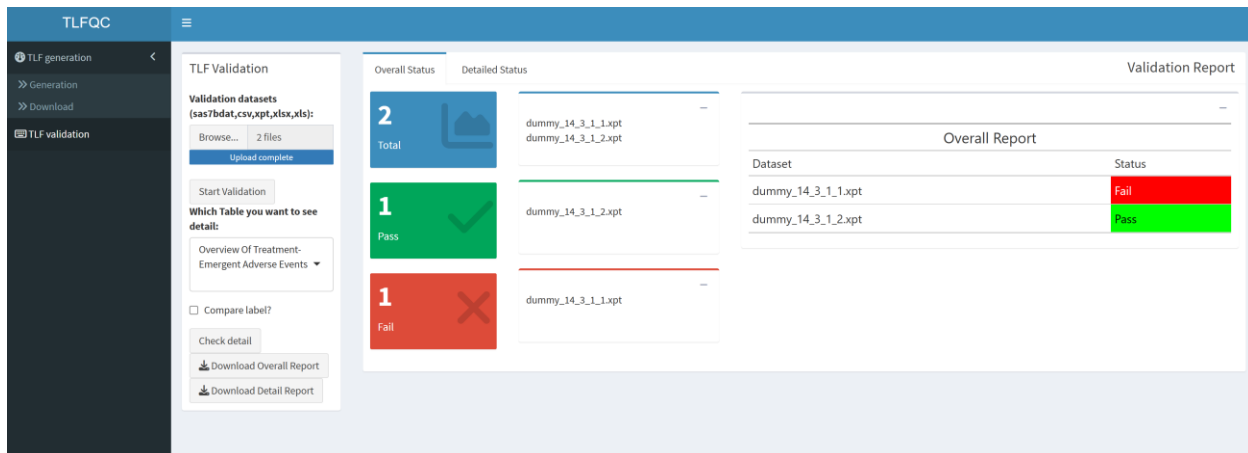
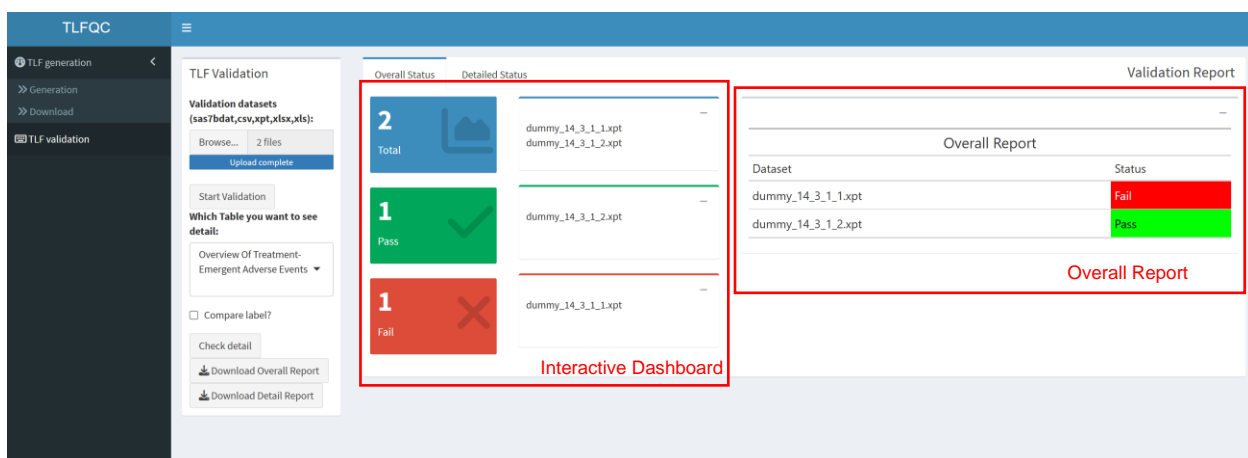


Figure 7). This tab presents the overall status in two formats: a dashboard and a report. In the report, two columns are presented: one for datasets and another for their corresponding status. The status is color-coded for clarity, appearing red if the validation fails and green if it passes. As mentioned earlier, the overall status can be obtained using `diffdf_has_issues()`. Multiple study statuses can then be combined into a table, which is rendered with `renderDataTable()` and displayed in the UI by using `dataTableOutput()`.



**Figure 7. Overall validation results dashboard and report in TLFQC**

On the left side of the report, the same information is presented in a dashboard format. The main components of this dashboard are value boxes and regular boxes, which aesthetically group results from `{diffdf}`. The value boxes display the total number of TLF datasets, along with counts of those that have passed and failed. To the right of these boxes, a detailed list of the datasets is presented, offering additional insights into each dataset's status. By interacting with the dashboard, users can intuitively understand and assess the validation results.

```
#UI
valueBoxOutput("total")
box(status="primary",textOutput("totaldataset"))
#server
valueBox(value = length(report$Dataset),
  subtitle = "Total",
  icon = icon("area-chart"),
  color = "light-blue")
output$totaldataset<-renderText({reportg()}$Dataset})
```

**R program 22. Using value box and box for interactive dashboard**

### 3. DETAILED STATUS FOR EACH COMPARED TLFS DATASETS

The Detailed Status tab on the TLFs validation page provides an in-depth view of discrepancies between the primary and validation datasets for each TLF. As previously mentioned, TLFQC examines discrepancies across six aspects. In the interactive dashboard for detailed validation results, there are six boxes that display the validation results for each of these aspects. When the “check detail” button is clicked, the boxes change color based on the validation results—green for pass, red for fail, and orange for not evaluated. This feature requires a reactively generated UI, thus we need to create this box inside `renderUI()` in the server and output it to the placeholder `uiOutput()` in UI. The `lbl()` function created in R program 21 acts as the indicator for the color of the boxes. Please see APPENDIX R PROGRAM 1 for more details.

Initially, all the boxes collapsed, offering users a clear and intuitive overview of the validation results. Users can manually expand the boxes to view detailed information. For instance, as shown in Figure 8 Detail validation results dashboard, the red box for label checking contains a table that displays columns with unmatched labels and their corresponding values. Additionally, a table beneath the six boxes presents the differences between the base and compared data for unmatched values.

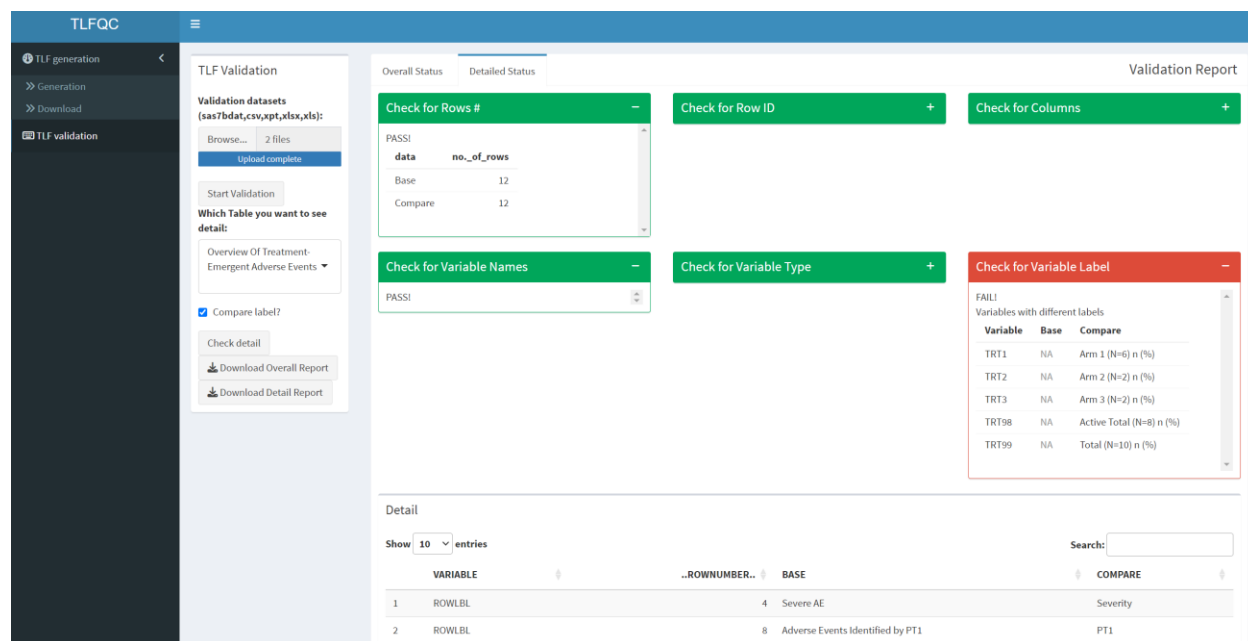


Figure 8 Detail validation results dashboard in TLFQC

#### 4. DOWNLOAD REPORTS

TLFQC also enables users to download both an overall validation results report and a detailed validation results report. These reports are created with `{gt}` package and downloaded by `downloadHandler()`, both of which were introduced earlier. Please see below Figure 9 for the detailed validation report generated by TLFQC.

Overall Status = FAIL			
Check	Status	PROD	QC
Status Check			
Rows(#)	PASS		
Columns(#)	PASS		
Same Attribute	FAIL		
Same Variable	PASS		
Same Value	FAIL		
Different Attribute			
TRT1		NULL	Arm 1 (N=6) n (%)
TRT2		NULL	Arm 2 (N=2) n (%)
TRT3		NULL	Arm 3 (N=2) n (%)
TRT98		NULL	Active Total (N=8) n (%)
TRT99		NULL	Total (N=10) n (%)
Different Value			
ROWLBL		Severe AE	Severity
ROWLBL		Adverse Events Identified by PT1	PT1
ROWLBL		Adverse Events Identified by PT2	PT2
TRT98		6 (75.0)	5 (62.5)
TRT98		6 (75.0)	5 (62.5)
TRT98		0	2 (25.0)

**Figure 9 Detailed validation report generated by TLFQC**

## CONCLUSION

The {shiny} package in R empowers users to develop Shiny applications, leveraging its inherently powerful features. The TLFQC platform was developed based on R shiny, which stands as a powerful and adaptable solution for automating the generation and validation of TLF. Users benefit from a streamlined workflow that ensures efficiency and reliability, from parameter input through to dataset generation, manipulation, validation, and report preparation. Its modular, flexible design accommodates custom R functions, fostering a high level of compatibility and long-term sustainability. By maintaining the independence of its core modules, the platform ensures the reliable operation of its functionalities, such as information input, TLF validation, and report downloads. Future enhancements may focus on more complex interactive data manipulation methods, more advanced figures comparison methods and further expanding adaptability for various TLFs.

## REFERENCES

- Bosak, D. J. (2024). *sassy: Makes 'R' Easier for Everyone. R package version 1.2.5*. Retrieved from <https://github.com/dbosak01/sassy>, <https://r-sassy.org>
- Gower-Page, C., & Martin, K. (2024). *diffdf: Dataframe Difference Tool. R package version 1.1.1*. Retrieved from <https://github.com/gowerc/diffdf/>
- Ling, C., & Wang, Y. (2025). Writing SAS MACROs in R? R functions can help! *PharmaSUG 2025 conference proceedings*. San Diego, CA.
- Wang, Y., & Ling, C. (2025). Comparing SAS® and R Approaches in Reshaping data. *PharmaSUG 2025 Conference Proceedings*. San Diego, CA.
- Wang, Y., & Ling, C. (2025). Controlling attributes of .xpt files generated by R. *PharmaSUG 2025 conference proceedings*. San Diego, CA.

## APPENDIX

The package and its version mentioned above: tidyverse 2.0.0, haven 2.5.4, reporter 1.4.4, ggplot 2 3.5.1, shiny 1.9.1, diffdf 1.1.1, shinydashboard 0.7.2, gt 0.11.1 and labelled 2.13.0.



Table 14.3\_1.2  
 Subjects With Treatment-Emergent Adverse Events By System Organ Class And Preferred Term  
 Safety Analysis Set

System Organ Class MedDRA 25.1 Preferred Term	Arm 1 (N=6) n (%)	Arm 2 (N=2) n (%)	Arm 3 (N=2) n (%)	Active Total (N=8) n (%)	Total (N=10) n (%)
Any adverse event	4 (66.7)	2 (100.0)	2 (100.0)	6 (75.0)	8 (80.0)
cl A.1	1 (16.7)	1 ( 50.0)	2 (100.0)	3 (37.5)	4 (40.0)
dcd A.1.1.1.1	0	0	2 (100.0)	2 (25.0)	2 (20.0)
dcd A.1.1.1.2	1 (16.7)	1 ( 50.0)	1 ( 50.0)	2 (25.0)	3 (30.0)
cl B.1	2 (33.3)	0	0	2 (25.0)	2 (20.0)
dcd B.1.1.1.1	2 (33.3)	0	0	2 (25.0)	2 (20.0)
cl B.2	1 (16.7)	0	0	1 (12.5)	1 (10.0)
dcd B.2.1.2.1	1 (16.7)	0	0	1 (12.5)	1 (10.0)
cl C.1	3 (50.0)	1 ( 50.0)	1 ( 50.0)	4 (50.0)	5 (50.0)
dcd C.1.1.1.3	3 (50.0)	1 ( 50.0)	1 ( 50.0)	4 (50.0)	5 (50.0)
cl C.2	2 (33.3)	1 ( 50.0)	0	2 (25.0)	3 (30.0)
dcd C.2.1.2.1	2 (33.3)	1 ( 50.0)	0	2 (25.0)	3 (30.0)
cl D.1	2 (33.3)	1 ( 50.0)	2 (100.0)	4 (50.0)	5 (50.0)
dcd D.1.1.1.1	1 (16.7)	1 ( 50.0)	1 ( 50.0)	2 (25.0)	3 (30.0)
dcd D.1.1.4.2	2 (33.3)	0	1 ( 50.0)	3 (37.5)	3 (30.0)

Arm 1: Dummy drug X

Arm 2: Placebo

Arm 3: Dummy drug Y

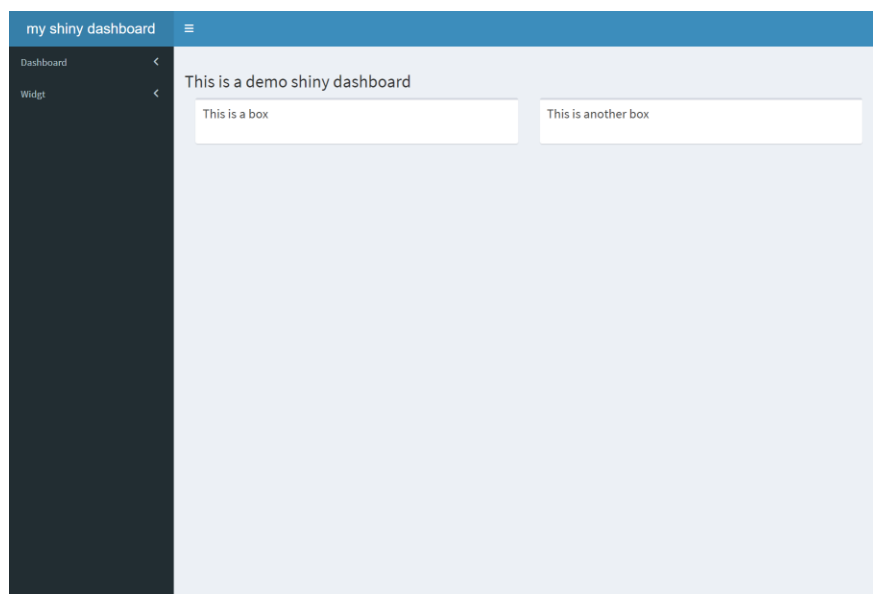
Note: Subjects are counted once in each row, regardless of the number of events they may have had.

Program source  
 code:U:/R/One-step-table-generation-and-comp  
 are-tool

## APPENDIX TABLE 1. Example table report output by TLFQC

```
#UI
uiOutput("chk1bl")
#Server
observeEvent(input$Start2,{
  output$chk1bl<-renderUI({
    box(title="Check for Variable Label",width=4,solidHeader=TRUE,status =
lbl(),collapsed = TRUE,collapsible = TRUE,
      div(textOutput("passlabel"),
        textOutput("label"),
        tableOutput("labeltbl"))
    )
  })
})
```

## APPENDIX R PROGRAM 1. Example of generating UI in server



**APPENDIX FIGURE 1. Demo app generated with {shinydashboard}**

## ACKNOWLEDGMENTS

We would like to express our sincere gratitude to Rina Loke, Christelle Raynaud **AND THE R TEAM**, Xiangdong Zhou, Holly Peterson and Ujjwala Powers for their support, trust, and encouragement all along.

## CONTACT INFORMATION

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

Chen Ling  
AbbVie Inc.  
Chen.ling@abbvie.com

Yachen Wang  
AbbVie Inc.  
Yachen.wang@abbvie.com