Paper OS02

Navigating and Building Bridges: A Framework for Clinical Programming using Open-Source Tools

Yash Ankurbhai Shah, Boehringer Ingelheim Pharma GmbH & Co. KG, Biberach, Germany
Kavitha Allala, Boehringer Ingelheim Pharmaceuticals, Inc., Ridgefield, USA

ABSTRACT

The rapid evolution of open-source tools and packages presents both opportunities and challenges for clinical programmers. While these resources offer potential for many efficient and interactive outputs, navigating this landscape can be overwhelming, particularly for those transitioning to R programming and open source. To address this, we developed a generalized framework using the well-supported open-source packages Tplyr, Reactable, and Shiny. This framework allows programmers to create summary tables and listings and pass them as arguments and generate clickable tables which offers traceability back to the source data with some added features, without further additions. This enables programmers to focus on their core tasks which is to build and define tables rather than the intricacies of software engineering. Designed to be loosely coupled, the framework is intended to be adaptable to various outputs and environments and provide long-term value.

INTRODUCTION

The pharmaceutical industry has witnessed a gradual yet consistent adoption of open-source tools. This trend is evidenced by the increasing contributions to R packages and the growth of consortiums and collaborations. Open-source tools are being utilized across all stages of clinical submissions, from dynamic applications for monitoring and cleaning activities to the creation of static outputs and the development of standardized functions for these outputs.

However, this transition presents several challenges, primarily the need for upskilling. While a learning curve is inevitable, simplifying this process is crucial. The lack of defined standards and coding practices means programmers often cannot use these tools directly or with minimal changes. Additionally, navigating the many open-source options can be daunting. There are numerous packages available for creating standardized outputs (e.g., gttables, tern, Tplyr, pharmaverse), with frequent updates that are hard to keep track of during regular trial work. Clinical trial reporting is already resource-intensive, so programmers need a stable and defined pathway. This would allow them to focus on programming tables in R, using familiar tools, and provide additional incentives for doing so.

There are several deliverables within a clinical trial that lead up to the final Clinical Trial Report (CTR) like Clinical Quality Monitoring (CQM), interim analysis, multiple clinical report planning meetings. The challenges discussed above are valid to support any of these deliverables leading up to the submission.
As these challenges are valid for any deliverable, we wanted to take CQM as an example to showcase the process from beginning to end.

CQM(CLINICAL QUALITY MONITORING)

CQM is performed on an ongoing basis and is a cross-functional review activity on the aggregated clinical data for monitoring the quality, consistency, and plausibility in the trials to ensure data integrity and accuracy of trial results.

The CQMP (Clinical Quality Monitoring Plan) outlines how the trial will be monitored to ensure participant safety, data quality, and compliance with the protocol. It also describes the scope, nature, frequency and set up of centralized clinical quality monitoring on aggregated clinical data and defines responsibilities and outputs for review.
CQM has a clear focus on the critical data that are defined in the trial Integrated Quality and Risk Management Plan (IQRMP). CQM is performed for clinical plausibility check of data and detection of possible quality issues and trends on clinical data.

The output templates put into the CQMP assist trial team members in examining and assessing the data from their own viewpoint. These templates will need to be reviewed by the team members before they are implemented by the programmer.

CURRENT CHALLENGES WITH THE CQM PROCESS

The implementation of templates from the CQMP involves the programmer using the source CDISC (Clinical Data Interchange Standards Consortium) SDTM (Study Data Tabulation Model) data. The outputs are then sent to the trial team for review during the CQM meetings, which are typically scheduled once a quarter during the trial conduct. Various tools can be used to implement these templates, but for the purpose of this paper, we will focus on SAS®.

The templates are implemented in SAS® and the resulting outputs are shared as PDF files. For a small subset of patients (e.g., 50 patients) in a trial, the PDF file generated to support the CQM can be several thousand pages long. This poses a significant challenge for both the programmer responsible for producing these outputs and the reviewer tasked with reviewing such extensive static outputs.

During the review of these PDF files, if the reviewer has any queries regarding the data, they will reach out to the programmer for investigation and clarification. Sometimes these queries involve requesting a list of patients who meet specific criteria. For instance, the reviewer might ask for a list of patients with HbA1C values above 7%, a list of patients with BMI values between 25 and 30 or explore the potential relationship between female patients and pregnancy test results.

While the reviewer continues their review, they have to wait for the programmer's response to their queries. This waiting period interrupts the reviewer's progress, as they cannot proceed until they receive the query results or clarifications from the programmer. Another outcome of the review process is the identification of data entry issues. These issues are shared with the Trial Data Manager (TDM), who then raises queries in the database and follows up with the site monitoring lead and/or clinical trial monitor. The reviewer interaction with the TDM reduces a lot of extra work and makes their work much more efficient.

As demonstrated, this is an iterative process where the CQMP is updated quarterly, new data is incorporated, query resolutions lead to data updates, and the outputs are refreshed with the updated data for review during the CQM meetings. This process is intensive and time-consuming, and it is common for trials in every therapeutic area to undergo such procedures.

The CQM process is supported by SAS® outputs or a proprietary system. With SAS®, the programmer has the freedom of creating their required customized tables. But they are static with no interactivity possible and time consuming for review. With proprietary tools, there is interactivity possibility but there are limitations where it requires knowledge to create required outputs and limited customizations. Additionally, it is dependent on vendor support for supporting user requirements. Both systems have their own pros and cons, and they lack additional features based on user requirements which makes the review process challenging.

To address all the challenges and take up the use case of CQM, we propose an Interactive CQM framework. This framework acts a middle layer and introduces interactivity while keeping the process simple, allowing the programmers to focus on building outputs without needing to learn complex Shiny and JavaScript architecture. It is designed to be generic enough to support various requirements across different therapeutic areas and loosely coupled to enable outputs in different environments.

CQM FRAMEWORK

The CQM package allows users to pass their tables to a function, add customization arguments, and receive interactive tables as outputs. This approach leverages existing packages rather than reinventing the wheel, providing users with tangible benefits for creating tables using R. One key functionality is the ability to drill down from a summary table to a listing for CQM. Given the vast number of outputs dictated by standards and therapeutic areas, users have the freedom to create their own tables.

The framework, {cqmpackage}, is built using the well-supported open-source packages {Tplyr} [1], {reactable} [2], and {shiny} [3]. The package leverages metadata extraction functionality from {Tplyr} to offer traceability from a summary table back to the source data. Metadata from each table cell is mapped to a click event, which reverse-generates the source data table. Thus, the ability to create desired table outputs in R with {Tplyr} benefits multiple deliverables without requiring additional computer programming training for the programmers to achieve complex resulting outputs. The outputs itself are in {Reactable}, which generates output tables with custom JS functions and extracts click row column values. And lastly Shiny itself which allows for different types of outputs (Custom *Shiny App*, *Quarto [4] Shiny*, etc.).

HERE IS A DETAILED WORKFLOW OF THE FRAMEWORK

Programmer creates tables using {Tplyr} package. Once you have the table object and output, the package provides ui and server functions (*reactableui* and *reactableserver*). Following the shiny modules namespace philosophy, the reactable ui only takes one argument which is unique ID. The server function takes the same id, the table object, and the final output. Along with that it takes some additional requirements which are added based on user requirements for e.g., if programmer wants to pre-specify and pass list of default column variables in the drilldown listing, column widths of {reactable} outputs and default download name of the filtered drilldown listing.

Below is an example with program screenshots

```
# Table ADSL Summary

# Create summary table object
table1 <- tplyr_table(adsl, ARM) %>%
  add_layer(
    group_desc(AGE, by = "Age (years)")
  ) %>%
  add_layer(
    group_count(EOSSTT, by = "End of Study status (%)")
  ) %>%
  add_layer(
    group_count(vars(DCDECOD, DCSREAS))
  )

# Once the object is created, build the table.

results1 <- build(table1, metadata = TRUE) %>%
  dplyr::mutate(row_label2 = ifelse(row_label2 == row_label1, "Total (%)",
                                    row_label2
  )) %>%
  apply_row_masks(row_breaks = TRUE)
```

Figure 1: ADSL (Subject-Level Analysis Dataset) table creation using {Tplyr}

```
# Table ADAE

tab2 <- tplyr_table(adae, TRTA) %>%
  set_pop_data(adsl) %>%
  set_pop_treat_var(TRT01A) %>%
  add_layer(
    group_count("All subject") %>%
      set_distinct_by(USUBJID) %>%
      set_format_strings(f_str(
        "xx",
        distinct_total
      ))
  ) %>%
  add_layer(
    group_count("Subjects with adverse events") %>%
      set_distinct_by(USUBJID) %>%
      set_format_strings(f_str(
        "xx (xx %)",
        distinct_n, distinct_pct
      ))
  ) %>%
  add_layer(
    group_count(AESEV, by = "Adverse event severity") %>%
      set_distinct_by(USUBJID) %>%
      set_format_strings(f_str(
        "xx (xx %)",
        distinct_n, distinct_pct
      ))
  ) %>%
  add_layer(
    group_count("Subjects with severe AE", where = AESER == "Y") %>%
      set_distinct_by(USUBJID) %>%
      set_format_strings(f_str(
        "xx (xx %)",
        distinct_n, distinct_pct
      ))
  )
results2 <- build(tab2, metadata = TRUE) %>% apply_row_masks(row_breaks = TRUE)
```

Figure 2: ADAE (Adverse Events Analysis Dataset) table creation using {Tplyr}

Once you have the {Tplyr} table object and built table. We put them in a simple shiny app.

```
# Define UI for application
ui <- fluidPage(

  # Application title
  titlePanel("demo CQM app"),

  # Main panel with tabs
  mainPanel(
    tabsetPanel(

      tabPanel("Demographics",
               reactableUI("res1")),
      tabPanel("Adverse Events",
               reactableUI("res2")),
      tabPanel("Demowgraphics",
               reactableUI("res3")),
      tabPanel("Demographics listing",
               customizedlistingUI("listing1")),


    )
  )
)
```

Figure 3: ui code

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

  customizedlistingServer("listing1", mock_adsl, "listthree.csv", showAllcols = TRUE)
  reactableServer("res1", results1, table1, "adsldownloadfile.csv",showAllcols = TRUE)
  reactableServer("res2", results2, tab2, "adaedownloadfile.csv",showAllcols = list("STUDYID"),
                  minimumcolwidth = 50)
  reactableServer("res3", results1, table1, "adsldownloadfile.csv",showAllcols = list("AGE","EOSSTT"),
                  minimumcolwidth = 1)



}

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

Figure 4: Server code

Here is the output (For reference, the entire code is available in the Appendix A.)

		Placebo	Xanomeline High Dose	Xanomeline Low Dose
All subject		86	84	84
Subjects with adverse events		32 (37 %)	46 (55 %)	50 (60 %)
Adverse event severity	MILD	31 (36 %)	41 (49 %)	37 (44 %)
	MODERATE	7 (8 %)	24 (29 %)	26 (31 %)
	SEVERE	0 (0 %)	2 (2 %)	7 (8 %)
Subjects with severe AE		0 (0 %)	0 (0 %)	0 (0 %)

Select columns to display:

TRTAN [Actual Treatment (N)], AGEGR1 [I

TRTAN Actual Treatment (N)	AGEGR1 Pooled Age Group 1	AESEV Severity/Intensity
81	65-80	MILD
81	65-80	MILD
81	65-80	MILD

Figure 5: Simple output

The framework can be used within multiple outputs which is as shown below:

Table 5.1 Disposition of Patients - Screened Set

Table 7.2 Last digit preference: weight by site - Treated set

Not Treated				9 (34.6%)
Treated				17 (65.4%)
Permanently discontinued trial medication	6 (100.0%)	9 (100.0%)	1 (33.3%)	16 (94.1%)
Completed planned treatment period	6 (100.0%)	8 (88.9%)	0 (0.0%)	14 (82.4%)
Withdrawal By Subject	0 (0.0%)	1 (11.1%)	0 (0.0%)	1 (5.9%)
Other	0 (0.0%)	0 (0.0%)	1 (33.3%)	1 (5.9%)
Permanently discontinued from the trial	6 (100.0%)	9 (100.0%)	0 (0.0%)	15 (88.2%)
Completed planned observation time	6 (100.0%)	8 (88.9%)	0 (0.0%)	14 (82.4%)
Withdrawal By Subject	0 (0.0%)	1 (11.1%)	0 (0.0%)	1 (5.9%)
On treatment at time of reporting	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)

1–11 of 12 rows Show 10 Previous 1 2 Next

Select columns to display:

DCTREAS [No Label], DCSREAS [No Lab

DCTREAS No Label	DCSREAS No Label	TRTFL No Label	ENRLFL No Label	ONTRTREP No Label	INTRLREP No Label
Completed planned treatment period	Completed planned observation time	Y	Y	N	N

Figure 6: Customized shiny app

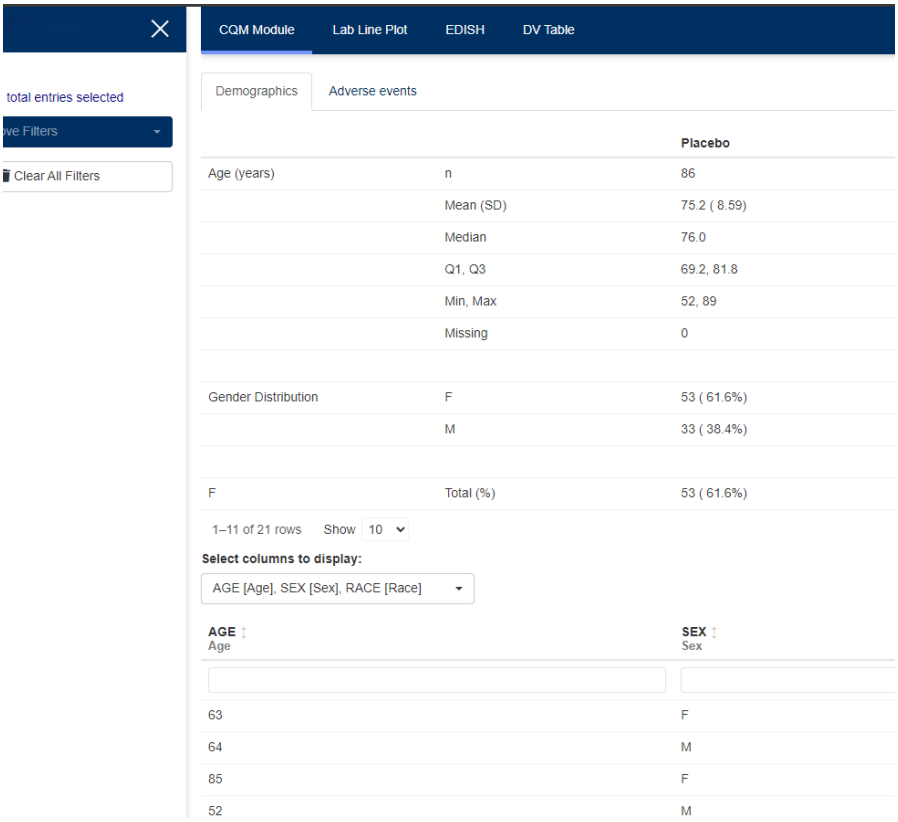

Figure 7: Framework within customized framework

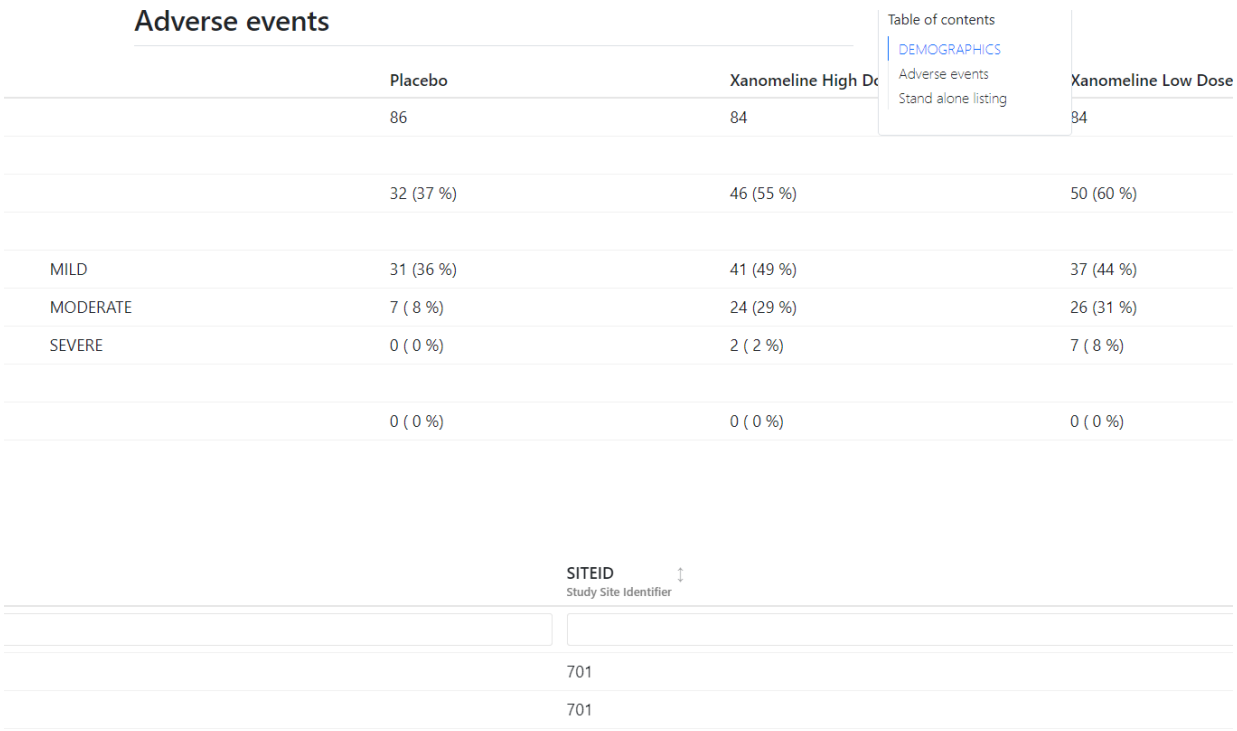

Figure 8: Framework within Quarto Shiny document

Initially, the framework started as custom functions that could be loaded into any Shiny app. Over time, it evolved into a package to accommodate new requirements, facilitate testing and documentation, and make sharing with end users and co-development easier. Developing a user community around this package is equally important. We worked with a team of programmers who had different levels of R knowledge. Along with the tutorials for table creation, we developed a guide which contained all the information with sample code and demo applications to explore package features. They learned to create several table outputs for their own trial and created different types of interactive outputs using this framework. A standard template for a Shiny app was also developed, which they can use and customize with their own requirements.

The reviewer now has the ability to interactively review the data using the framework designed to address static output challenges. This includes sorting, filtering, subsetting, and exporting the data into Excel®. Consequently, the reviewer no longer relies heavily on the programmer for tasks such as providing a list of patients that meet specific query criteria.

CONCLUSION

The updates have received positive feedback, with reviewers appreciating the autonomy to review data based on their interests. To further improve the reviewing process for trials with a large number of patients (over 1000), we need to explore additional features and determine how the framework can handle such scenarios.

Currently, in the pilot phase, the framework has a limited set of features. However, our goal is to continue enhancing the framework to reduce dependencies between the programmer and the reviewer.

One important feature we would explore is to track the review. This will allow reviewers to easily pick up where they left off from their last review, saving time by avoiding re-reviewing already reviewed data and allowing them to focus on new and updated data.

Currently, we are collaborating with the stakeholders involved in the CQM process. Our goal is to adapt the framework to enable an efficient review of the clinical trial data.

REFERENCES (HEADER 1)

1. Miller E, Stackhouse M, Tarasiewicz A (2024). *Tplyr: A Traceability Focused Grammar of Clinical Data Summary*. R package version 1.2.1, https://github.com/atorus-research/Tplyr.
2. Lin G (2024). *reactable: Interactive Data Tables for R*. R package version 0.4.4.9000, https://github.com/glin/reactable, https://glin.github.io/reactable/.
3. https://shiny.posit.co/
4. https://quarto.org/

CONTACT INFORMATION

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

Author Name : Yash Ankurbhai Shah
Company : Boehringer Ingelheim Pharma GmbH & Co. KG
Address : Birkendorfer Str. 65 | 88397 Biberach
Email : yash_ankurbhai.shah@boehringer-ingelheim.com

Author Name : Kavitha Allala
Company : Boehringer Ingelheim Pharmaceuticals, Inc.
Address : 900 Ridgebury Road | Ridgefield CT 06877
Email : kavitha.allala@boehringer-ingelheim.com

APPENDIX A

Full example code creating one ADAE created in Tplyr and using it in cqmframework and output in a shiny app.

```
library(shiny)
library(dplyr)
library(Tplyr)
library(cqmpackage)
# Let us create another summary table as an example from ADAE
# Table ADAE

tab2 <- tplyr_table(adae, TRTA) %>%
  set_pop_data(adsl) %>%
  set_pop_treat_var(TRT01A) %>%

  add_layer(
       group_count("All subject") %>%
       set_distinct_by(USUBJID) %>%
       set_format_strings(f_str("xx", distinct_total))
           ) %>%
  add_layer(
       group_count("Subjects with adverse events") %>%
       set_distinct_by(USUBJID) %>%
       set_format_strings(f_str("xx (xx %)",distinct_n, distinct_pct))
            ) %>%
  add_layer(
       group_count(AESEV, by = "Adverse event severity") %>%
       set_distinct_by(USUBJID) %>%
       set_format_strings(f_str("xx (xx %)", distinct_n, distinct_pct ))
            ) %>%
  add_layer(
       group_count("Subjects with severe AE", where = AESER == "Y") %>%
       set_distinct_by(USUBJID) %>%
       set_format_strings(f_str("xx (xx %)", distinct_n, distinct_pct))
            )
results2 <- build(tab2, metadata = TRUE) %>% apply_row_masks(row_breaks = TRUE)
```

Once you have the {Tplyr} table object and built table. We put them in a simple shiny app.

```
# Define UI for application
ui <- fluidPage(

  # Application title
  titlePanel("demo CQM app"),

  # Main panel with tabs
  mainPanel(
       tabsetPanel(
       tabPanel("Adverse Events",reactableUI("res2"))
                  )
           )
               )
# Define server logic
server <- function(input, output) {
       reactableServer("res2", results2, tab2, "adaedownloadfile.csv")
                                    }

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