PharmaSUG 2025 - Paper OS-288
Putting the 'R' in RWD: Leveraging R and Posit to enhance Real World Data Programming

Darren Jeng

Sachin Heerah

Abstract

The Pfizer Real World Data (RWD) programming team has leveraged R and Posit services to enhance the capabilities of its programmers. We have designed an R package, Shiny apps and even a Quarto website to support all programmers with varying backgrounds, including those with only SAS experience. Our R package is designed to simplify database queries, utilize both R and SAS variable syntax, and standardize deliverables. We have leveraged Posit's RStudio features such as code snippets to make code templates readily accessible for all users within the IDE. Code guides are also presented as snippets to allow all users to load example data and explore standard RWD programming workflows. Overall, embracing and leveraging the features available to us in R and Posit is enhancing our workflow through integrated resources and documentation.

Designed for RWD Programming

To facilitate our work, the RWD Programming team at Pfizer has developed macros in SAS to connect to and query databases, run models, summarize data, and produce TLFs. Over the years, we have turned to open-source programming, with a focus on R (R Core Team 2023), to build and develop our capabilities for the future.

Simplifying SQL Workflow

SQL is an integral tool for working with real world data, and we need easy and repeatable ways to run our queries and store the tables being created. Given the scope of an analysis, we may decide to only have temporary tables created and referenced, or store them as permanent tables for later use. This option is controlled in the function sfcreate() that handles SQL queries. This function includes Boolean flags, allowing for easy of adjustment. A few of these can be seen in Program 1 and Program 2 below in the next section.

As numerous tables are being created in one session there could be conflicts. Our macro tool insures prevention of overwriting existing tables by validating previously defined tables and stopping users from overwriting. Our macro also has the ability to preview queries immediately after running the query to assist during the analysis.

SAS vs R

We designed our R functions to have the same or similar structure as their SAS counterparts. For example, the following SAS code (Program 1) will create the temporary patlist table where enrolid is the patient ID.

```
%sfcreate(
name = patlist  /* Name of object to create */
,index= enrolid /* Space delimited list of columns to use for index */
,temporary=1    /* Temporary table? */
,replace=1      /* Replace table? */
,connection=sf  /* Name of existing PROC SQL database connection */
)
  select enrolid from patient_diagnoses
  where diag_cd like '251%'
  group by 1
&sfcreate_close;
```

Program 1: SAS code to create a SQL table.

In R, the function is called in a similar way (Program 2). The main difference is that strings are enclosed in quotation marks, unlike SAS.

```
sfcreate(
  name = patlist,
  index = enrolid,
  temporary = 1,
  replace = 1,
  .conn = conn,
  query = "
    select enrolid from patient_diagnoses
    where diag_cd like '251%'
    group by 1"
)
```

Program 2: R code to create a SQL table.

By keeping the structure of the functions the same between SAS and R, we enable programmers to seamlessly switch between the two languages, allowing them to leverage the capabilities of both while lowering the barrier to entry for RWD programming.

SQL Queries with SAS and R Support

When translating the existing SAS macro to an R function, we designed the R function to be capable of parsing both R glue syntax, as well as SAS syntax. This would enable SAS programmers to continue to write the SQL queries as they're used to in SAS, as well allow any programmer to reuse existing SQL queries built in SAS. The only consideration is that all variables must be recreated in R (Program 3).

```
DBSTARTDT <- "01-01-2024"
DBENDDT <- "31-12-2024"

#SAS syntax
sfcreate(
name = "enrolled_patients",
index = "patientid",
query = "
select * from enrollment
where claim_dt between date(&dbstartdt.) and date(&dbenddt.)
"
)

#R Glue syntax
sfcreate(
name = "enrolled_patients",
index = "patientid",
query = "
select * from enrollment
where claim_dt between date({DBSTARTDT}) and date({DBENDDT})
"
)

#Raw values
sfcreate(
name = "enrolled_patients",
index = "patientid",
query = "
select * from enrollment
where claim_dt between date('01-01-2024') and date('31-12-2024')
"
)
```

Program 3: R function accepting two different programming formats for dates.

Standardize table outputs

Final outputs for analysis often involve presenting results in a standardized format different from a traditional data frame in R. We have a suite of macro tools available that give us the capability to build these outputs. Depending on the output requests, we have different macros to prepare different table types, while keeping a uniform styling between each option available.

We leveraged the tidyverse suite of packages (Wickham et al. 2019) to design the tools to be used modularly, handling setup, calculations, and final output generation at different steps of the analysis. Program 4 below shows the modularity used for generating reports

on a patient level cohort. We can use the same function for different types of data (continuous vs categorical) in the same table.

```
# for obtaining counts in the data
oroutput_module(category="cyl", #column with data to be analysed
                category_rename="Cylinder", #name of category
                module_type="cat", #type of analysis
                levels_OG=c(4,6,8), #original levels in the data
                levels_NEW=c("4 cyl","6 cyl","8 cyl") #renamed levels for use
as titles
                )

# for obtaining summary statistics of variable
oroutput_module(category="wt",
                category_rename="Weight",
                module_type="summary",
                round_all = 2 #option for rounding with numerical analyses
                )
```

Program 4: R code showing different modules used to analyze data for final output.

Following the analysis of data as in Program 4, we then pass a final combined dataframe to the openxlsx package (Schauberger and Walker 2023) to produce our final deliverable as seen in Figure 1 below.

Table 1b
Motor Trend Car Road Tests Analysis

Characteristics	All mtcars N/Stat 1 (%/Stat 2)	4 cyl N/Stat 1 (%/Stat 2)	6 cyl N/Stat 1 (%/Stat 2)	8 cyl N/Stat 1 (%/Stat 2)
Cylinder				
4 cyl	11 (34.38%)	11 (100%)	0 (0%)	0 (0%)
6 cyl	7 (21.88%)	0 (0%)	7 (100%)	0 (0%)
8 cyl	14 (43.75%)	0 (0%)	0 (0%)	14 (100%)
Unknown/missing	0 (0%)	0 (0%)	0 (0%)	0 (0%)
Weight				
Mean (SD)	3.22 (0.98)	2.29 (0.57)	3.12 (0.36)	4 (0.76)
N (missing)	32 (0)	11 (0)	7 (0)	14 (0)
Min (max)	1.51 (5.42)	1.51 (3.19)	2.62 (3.46)	3.17 (5.42)
Q1 (Q3)	2.54 (3.65)	1.83 (2.78)	2.77 (3.44)	3.52 (4.07)
Median (IQR)	3.33 (1.03)	2.2 (0.74)	3.21 (0.62)	3.76 (0.48)

Figure 1: Final table output from Program 4, using the default mtcars data in R.

Support for new and experienced R Programmers

Documentation

With R Studio, we can add documentation for all functions directly inside the IDE. Previously, we would maintain text files for each SAS macro in a share drive. The programmer would then have to navigate to this share drive to view any documentation for

the SAS macros. Now, all a user needs to do is simply type ? followed by the function name in the R console.

Code Snippets: Templates and Guides

Code snippets are, in a nutshell, an auto complete feature present in R Studio. By typing out a preset phrase, the user will have the option to auto-fill the code tied to that phrase. We have leveraged this feature to place our programming templates at our programmers' fingertips directly inside the R Studio IDE. Previously, like above, they would need to navigate to the respective template in the shared drive. Now, the programmer just needs to begin typing "template" for the code snippet to appear.

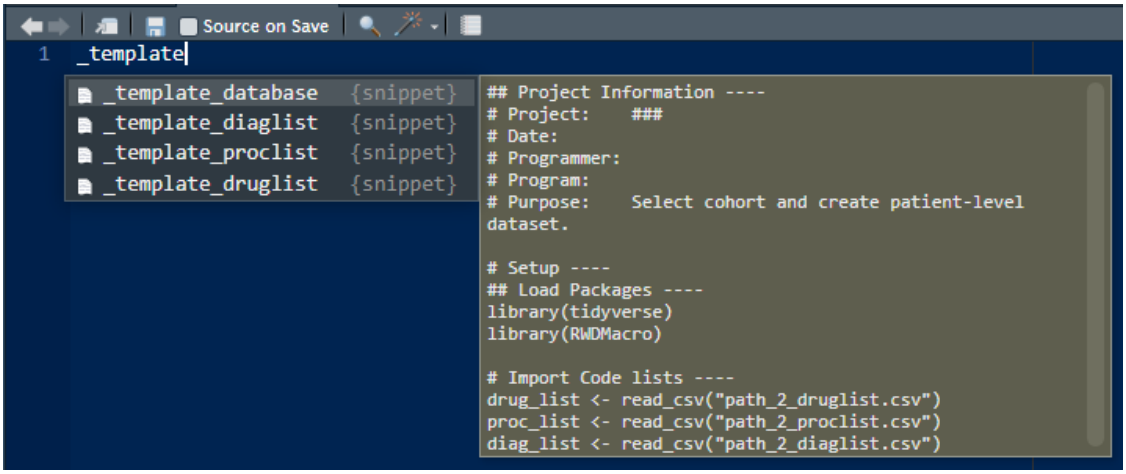

Figure 2: Calling the code template for a database program using the code snippet feature in R Studio.

New to our R space, are guides. While templates can be useful for illustrating workflow, and code documentation can show examples of how code can be used, we created programming guides that can be called using the code snippets. These guides load sample data and show users how to use the RWD Programming R functions on this sample data. This approach shows programmers how to use the functions, while giving them their own sandbox to experiment with the sample data.

Posit Connect: Shiny and Quarto

We have also leveraged the capabilities of Posit Connect to automate and distribute tools to all RWD Programming colleagues via Shiny apps and Quarto documents and websites.

We have a few R Shiny apps developed to create project folders based on incoming requests, to others fetching the latest public data (example the Bureau of Labor Statistic's Consumer Price Index) and updating our look-up tables in Snowflake. Other R code run on a schedule in Posit Connect to update our Monday project management space based on incoming requests to our ticketing system.

Recently, we have also begun exploring Quarto. While we do store function documentation, templates and guides within the R Studio IDE, we still need a repository for other information a programmer may need. This can include standard approaches in programming, notes on various databases, instructions on how to access various restricted data and tools, etc.

Previously, we collected all this information in a OneNote Notebook that we restricted access to. While it did serve its purpose, there were drawbacks. The most important was that the information was searchable across the entire notebook, but only page by page. The notebook was also quite slow, and updating information in multiple places was time consuming and can lead to inconsistencies if one page was forgotten during the update.

Considering the need to keep our guidance document restricted, while allowing search, and ease of use while updating and maintaining the information, we opted to host a Quarto website on the internal Pfizer Posit Connect. First, and foremost, the Quarto website would be available to all RWD Programmers on our Posit Workbench security group. The website is fully searchable allowing programmers to find relevant information quickly. Now, we can also programmatically update the website by keeping a centralized document with variables that can be called across the hundreds of pages we have. These variables can include file paths, dates and package versions, etc.

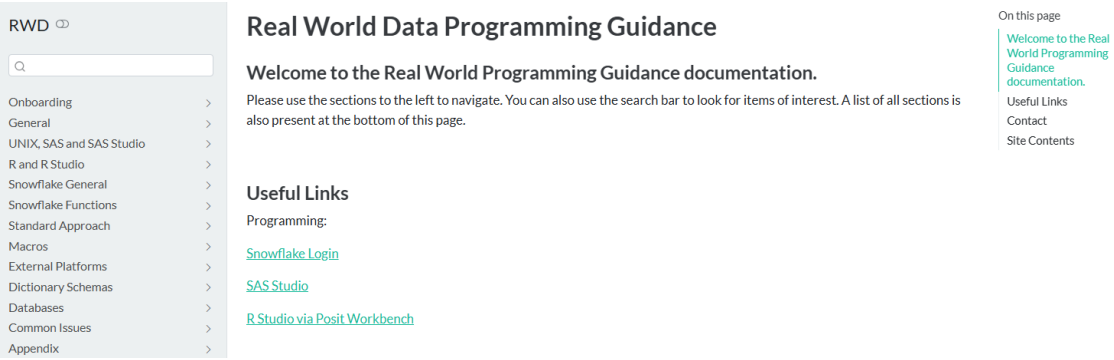

Figure 3: Landing page for the RWD Programming Guidance Quarto page.

Conclusion

R and Posit has become an integral part of RWD Programming over recent years. The suite of tools and technology available for programmers to leverage is vast and can enable our team to onboard new programmers with lower barriers to programming, while providing continued support to continuing programmers.

References

R Core Team. 2023. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Schauberger, Philipp, and Alexander Walker. 2023. *Openxlsx: Read, Write and Edit Xlsx Files*.

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." *Journal of Open Source Software* 4 (43): 1686. https://doi.org/10.21105/joss.01686.

Contact Information

Darren Jeng

Pfizer Inc.

Darren.Jeng@pfizer.com

Sachin Heerah

Pfizer Inc.

Sachin.Heerah@pfizer.com