Paper PP11

Unraveling R logs: A Guide to Powerful and Clear Logging

Vishnu Priya Vishnubhotla, Johnson & Johnson, Hyderabad, India

ABSTRACT

Are you struggling with tracking analysis steps and troubleshooting in R? In the realm of statistical programming, SAS® users transitioning to R often face challenges due to R's lack of a built-in logging system. They must rely on console output and error messages, which complicates the debugging process.

This paper emphasizes the importance of clear logging and highlights the necessity of logging packages in R, exploring packages logrx, logr, tidylog and whirl to enhance log readability and structure. Each package offers unique features; for instance, logrx specifically facilitates logging in a clinical environment with the goal of making code easily traceable and reproducible, while tidylog captures operations within dplyr; logr allows for easy logging of messages with different severity levels (INFO, WARNING, ERROR), and whirl facilitates batch execution summary. Through a comparison of R and SAS® logs, this paper illustrates how customized logging strategies can enhance the R programming experience and support effective debugging.

INTRODUCTION

Programming in R language brings flexibility and modern capabilities, however, one major gap is the **absence of a native logging system** in R.

- **SAS® Advantage:**
  SAS® provides a robust, built-in logging mechanism that automatically captures program execution, errors, warnings, and performance details. Logs are highly structured, customizable, and facilitate debugging, and make reproducibility straightforward.
- **R Limitation:**
  R relies on console output and error messages, which are scattered and lack structure. While basic logging functions exist, R does not generate comprehensive logs by default.

Feature	SAS® Log	R Default Log
Structure	Highly structured and comprehensive log file with NOTE, INFO, WARNING, ERROR	Unstructured console output, scattered and less informative
Customization	System options like MPRINT, SYMBOLGEN, NOTES/ NONOTES to alter log content. Statements like PUTLOG, %PUT, allow developers to write custom messages to the log.	Minimal, requires manual coding. Base R provides sink() to redirect console output to a file. Functions like print(), message(), warning() and stop() are used to output messages or handle exceptions.

Table 01: SAS® vs R Default Logging Mechanism

R DEFAULT LOGGING APPROACH

- By default, R does not automatically generate a log file from the terminal. Only the stderr, stdout are printed to the terminal. These can be redirected to a log file using:
  `Rscript my_script.R > my_script.log 2>&1`

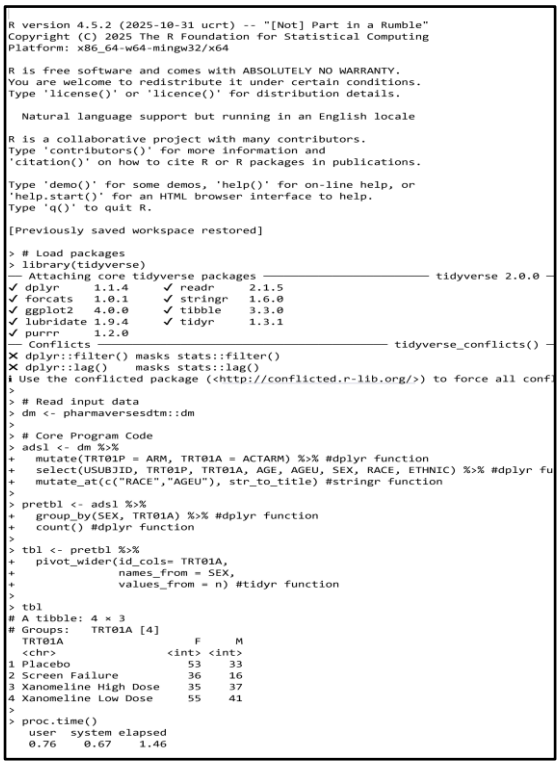
```
── Attaching core tidyverse packages ──────── tidyverse 2.0.0 ──

✓ dplyr     1.1.4     ✓ readr     2.1.5
✓ forcats   1.0.1     ✓ stringr   1.6.0
✓ ggplot2   4.0.0     ✓ tibble    3.3.0
✓ lubridate 1.9.4     ✓ tidyr     1.3.1
✓ purrr     1.2.0
── Conflicts ────────────────────────── tidyverse_conflicts() ──

✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) t
o force all conflicts to become errors
# A tibble: 4 × 3
# Groups:   TRT01A [4]
  TRT01A                F     M
  <chr>             <int> <int>
1 Placebo              53    33
2 Screen Failure       36    16
3 Xanomeline High Dose 35    37
4 Xanomeline Low Dose  55    41
```

Image 01: Snippet of a log file generated by redirecting stderr, stdout

- In Windows environment, a .Rout file can be created to log the console output (similar command exists for Unix/Linux) by using below command in the R terminal.
  `R CMD BATCH --no-save my_script.R # no-save is to skip saving .Rdata(workspace)`



```
R version 4.5.2 (2025-10-31 ucrt) -- "[Not] Part in a Rumble"
Copyright (C) 2025 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> # Load packages
> library(tidyverse)
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ─
✔ dplyr     1.1.4     ✔ readr     2.1.5
✔ forcats   1.0.1     ✔ stringr   1.6.0
✔ ggplot2   4.0.0     ✔ tibble    3.3.0
✔ lubridate 1.9.4     ✔ tidyr     1.3.1
✔ purrr     1.2.0
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ─
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all confl
>
> # Read input data
> dm <- pharmaversesdtm::dm
>
> # Core Program Code
> adsl <- dm %>%
+   mutate(TRT01P = ARM, TRT01A = ACTARM) %>% #dplyr function
+   select(USUBJID, TRT01P, TRT01A, AGE, AGEU, SEX, RACE, ETHNIC) %>% #dplyr fu
+   mutate_at(c("RACE","AGEU"), str_to_title) #stringr function
>
> pretbl <- adsl %>%
+   group_by(SEX, TRT01A) %>% #dplyr function
+   count() #dplyr function
>
> tbl <- pretbl %>%
+   pivot_wider(id_cols= TRT01A,
+               names_from = SEX,
+               values_from = n) #tidyr function
>
> tbl
# A tibble: 4 × 3
# Groups:   TRT01A [4]
  TRT01A                F     M
  <chr>                 <int> <int>
1 Placebo                  53    33
2 Screen Failure           36    16
3 Xanomeline High Dose     35    37
4 Xanomeline Low Dose      55    41
>
> proc.time()
   user  system elapsed
   0.76    0.67    1.46
```

Image 02: Snippet of a .Rout file

- Interactive R Studio program execution gives a console output like below.

```
> # Load packages
> library(tidyverse)
── Attaching core tidyverse packages ──────
✔ dplyr     1.1.4     ✔ readr     2.1.5
✔ forcats   1.0.1     ✔ stringr   1.6.0
✔ ggplot2   4.0.0     ✔ tibble    3.3.0
✔ lubridate 1.9.4     ✔ tidyr     1.3.1
✔ purrr     1.2.0
── Conflicts ──────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package to force all conflicts to become errors
>
> # Read input data
> dm <- pharmaversesdtm::dm
>
> # Core Program Code
> adsl <- dm %>%
+   mutate(TRT01P = ARM, TRT01A = ACTARM) %>% #dplyr function
+   select(USUBJID, TRT01P, TRT01A, AGE, AGEU, SEX, RACE, ETHNIC) %>% #dplyr
function
+   mutate_at(c("RACE","AGEU"), str_to_title) #stringr function
>
> pretbl <- adsl %>%
+   group_by(SEX, TRT01A) %>% #dplyr function
+   count() #dplyr function
>
> tbl <- pretbl %>%
+   pivot_wider(id_cols= TRT01A,
+               names_from = SEX,
+               values_from = n) #tidyr function
>
> tbl
# A tibble: 4 × 3
# Groups:   TRT01A [4]
  TRT01A                F     M
  <chr>                 <int> <int>
1 Placebo                  53    33
2 Screen Failure           36    16
3 Xanomeline High Dose     35    37
4 Xanomeline Low Dose      55    41
>
```

Image 03: R console output with minimal log messages

DEFINITION OF CLEAR AND COMPREHENSIVE LOG

Based on the best practices for regulated environments, a clear and comprehensive programming log should include the following attributes:

- Environment and Session Details
  - Software version, operating system, and machine information
  - List of loaded packages and their versions, External software dependencies
  - Working directory and program path

- Execution Metadata
  - Log start and end time, Elapsed time for script execution

    - Username or ID
    - Hash or checksum of the script for integrity verification

- Program Context
    - Source code (optional, for full traceability)
    - Notes for each executed step (timestamps, pipeline details)
    - Input and output file paths

- Messages and Diagnostics
    - Informational messages, Warnings (with context), Errors (with traceback if possible)
    - Custom messages inserted by the programmer

- Data Transformation Details
    - Summary of operations (filtering, joins, summarization)
    - Row and column count before and after transformations

- Compliance Features
    - Approved/unapproved package and function checks
    - Record of masked functions
    - Optional linting results for code quality
    - Information to reproduce environment

- Output Structure
    - Organized sections: Header, Body, Footer
    - Human-readable format (text or HTML)
    - Separate message file for errors/warnings (for quick review)

NEED FOR COMPREHENSIVE LOGS IN CLINICAL ENVIRONMENT

Programming Perspective

- Clear logs help programmers to track analysis steps and confirm expected results.
- Identify errors and warnings, pinpointing problematic code sections quickly.

Regulatory Compliance

- In clinical trials, reproducibility and traceability are critical to meet regulatory standards. Logs enable reviewers to verify program execution and ensure data integrity.
- FDA requires that the software package(s) used for statistical analyses should be fully documented in the submission, including version and build identification.
- To support health authority submissions, logs must capture:
    - Environment details (system, packages, versions)
    - Execution steps
    - Warnings and errors

BRIDGING THE GAP IN R WITH LOGGING PACKAGES

To achieve clear and structured logging in R, programmers must use specialized packages and explicit logging calls. While R community has developed several packages to provide robust logging capabilities, most of them were built for R package developers.

This paper focuses on the below logging solutions that are practical for regulated clinical trial environments, helping statistical programmers, analysts, and statisticians maintain clear program execution records and troubleshoot efficiently.

- **tidylog:** Captures and logs data manipulation steps in dplyr/tidyr pipelines.
- **logr:** Simplifies logging with severity levels and autolog for dplyr, tidyr, and sassy functions.
- **logrx:** Creates detailed, structured logs including the R environment details, errors, and warnings.
- **whirl:** Supports batch execution and generates summary logs for multiple scripts.

Logging dplyr/tidyr Operations with tidylog

The tidylog package generates logging messages for most dplyr and tidyr functions such as filter, mutate, select, full_join, and group_by and it can be especially helpful in code with longer pipes. This package integrates nicely with other logging packages such as logr to provide automatic logging for common data manipulation tasks.

- Once the tidylog package is loaded, it masks the default dplyr and tidyr functions with its own wrapper functions. So, for it to work correctly, tidylog should be loaded after other packages are loaded.
- Using `tidylog` adds a small overhead to each function call. For instance, because tidylog needs to figure out how many rows were dropped when you use `tidylog::filter`, this call will be a bit slower than using `dplyr::filter` directly. The overhead is usually not noticeable, but can be for larger datasets, especially when using joins.
- The tidylog logging can be switched off for a single long-running command using prefix dplyr:: or tidyr::, such as in dplyr::left_join. Alternatively, to turn off the output more permanently, set the global option tidylog.display to an empty list.
- By default, tidylog.display is set to NULL and tidylog will use the message function to display the log output in the console. To print the log output to both the console and an external log file, use the code below after loading tidylog package in the program.

```
library(tidylog) # Load tidylog package
log_to_file <- function(text)
      cat(text, file = "<path to the log>/<name of the log file>.log",
          sep = "\n", append = TRUE)
options("tidylog.display" = list(message, log_to_file)) # Send log to console, file
```

Image 04: Snippet of log generated by tidylog (left side) in the console compared to SAS® (right side)

Streamlined Logging and Severity Control with logr

The logr package is part of the sassy meta-package. It provides simple functions for manual, direct logging to log specific events in the program to create clean, human-readable log file in R with headers and footers, often in a "SAS-like" manner, which is useful during development or debugging. It supports INFO, WARNING, and ERROR messages, and is designed for analysts who need a straightforward written log of program execution.

Logging may be controlled globally using the "logr.on" option. This option accepts TRUE (default) or FALSE values. If the option is set to FALSE, logr will print to the console, but not to the log.

```
library(logr) # Load logr package
options("logr.on" = FALSE) # logr prints to console only when FALSE
```

A logr log is composed of three parts:

1. Log Header
   The log header provides an introduction to log. It contains:
   - Path of the log file, program path, working directory
   - Name of the user
   - Log start time
   - Information about operating environment such as R version, Machine, Operating system
   - Base packages, other packages that were installed and attached at the point the log_open() function was called when the script was executed.

2. Log Body
   The log body comprises the primary content of log.
   - The body will be populated with printouts from the objects the user sends to the log with log_print(), log_info(), put() and sep(). The objects will be printed to the log just as they are printed to the console.
   - Log Errors with traceback and Warnings.
     - logr will write all errors and warnings to the log at the point they are encountered (like SAS®).
     - A traceback of the error message is printed to the log and message files by default.
     - Additional errors and warnings could be included with log_error() and log_warning().
   - Optional notes will be written to the log for every call to log_print() or put().
     - The notes will follow the printing of the object and will include a date-time stamp and the elapsed time since the last call.
     - If the object is of class data.frame, the number of rows and columns will also be noted.
     - Notes are enabled by default and can be turned off using any of the below:
       ```
       log_open(show_notes = FALSE)
       options("logr.notes" = FALSE) #global option
       ```
   - Source Program Code, if log_code() function was called in the program.
   - It also includes messages written by tidylog to log data manipulation steps automatically when autolog option is enabled. It can be enabled/disabled using any of the below:
     ```
     log_open(autolog = FALSE)
     options("logr.autolog" = FALSE) #global option
     ```

3. Log Footer
   The log footer concludes the log with log end datetime, and total elapsed time for the entire script.
   - During batch runs or when sourcing a script, the log footer will not be generated if the script execution ends with errors. Instead, the last line of the log will be the error message.

Function	Purpose
Primary functions	
log_open()	Opens/Initiates a log
log_print(), put()	Print an R object (like a vector, list, or data frame) to the log file and console. It is useful for debugging, tracking the state of variables and data during script execution.
sep()	Alias for log_print, except it will print a separator before and after the printed text to help organize your log into sections.
log_close()	Closes the log
Exception handling functions	
log_error()	Logs an error
log_warning()	Logs a warning

Utility functions	
log_code()	Log the current program code (like SAS®) at the top of the log. The code lines will be prefixed with a right arrow (“>”) to distinguish these lines from the rest of your log.
log_info()	Logs an informational message (a character string). It does not print an R object like log_print does. The message is written to the log at the point the function is called.

Table 02: Key functions to create a log file using logr package

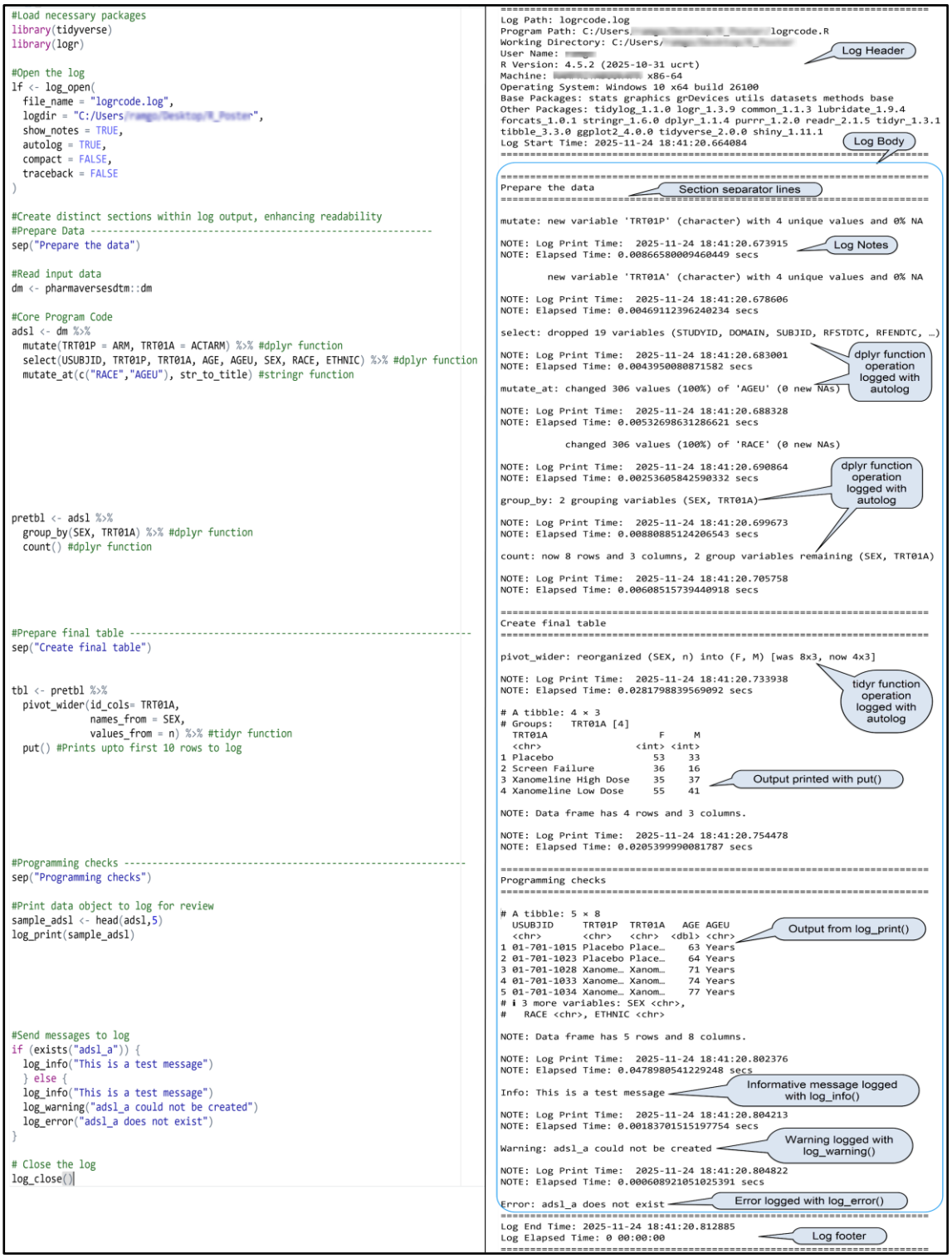


Image 05: A .log file (on right side) created by logr functions (R code on left side) with NOTE, INFO, WARNING, ERROR levels & tidylog integrated via autolog

If errors or warnings are generated, a separate message file is created with the same name as the log file, but with a .msg extension. If the log is clean, this file will not be created. The purpose of the msg file is to give the user a visual indicator from the file system that an error or warning occurred and is useful when running programs in batch.

```
Warning: adsl_a could not be created

Error: adsl_a does not exist
```

Image 06: A .msg file created by logr with list of all errors/warnings from .log file

Generating Structured Logs with logrx

The logrx package is built for automatic, comprehensive logging of an entire R script's execution, primarily for clinical programming applications that require traceability and reproducibility. It allows users to customize sections of the log based on their needs for a log, e.g. users can toggle on/off the messages, outputs, errors.

The structure of a logrx log is different from that of a SAS® log as logrx log is organized into sections whereas SAS® logs sequentially as the program executes. A typical log created by logrx contains the following:

- Logrx Metadata
  - Version of the package, Type of build, Link to the GitHub repository
- User and File Information
  - User that generated the log
  - Name and path of the script for which the log was generated
  - hash_sum: A unique hashsum is created for the log file
- Session Information
  - R version, OS and system, GUI, Language and timezone
  - List of all available packages in the environment
  - List of all external software
- List of all functions masked by packages
- List of all packages and functions used in the script (optional)
- URLs of the repositories (optional)
- Unapproved Packages and Functions (optional) for traceability within a validated environment
- Program Run Time Information - Start, end, and run times
- List of Errors and Warnings created during execution of the script.
- Messages, Output and Results (optional)
- Custom Information to the log (optional)
- Log Output File
  - Name and path of the log
  - Details of packages attached, and objects masked
  - List of Input files/datasets with its path

The logrx package also includes options to record linting results to the log, and to create a renv.lock file to secure project dependencies effortlessly and ensure reproducible development environment.

The logrx package offers multiple ways to generate a log including command line execution, batch execution. Easiest ways to run an R program using logrx and automatically generate a log are as follows:

- Using axecute() directly in the script, R terminal, or in the console or in batch script to run R program. A log is set-up around the program, and its code runs using safely() from purrr package, capturing and logging the errors, warnings, messages, output/result.

```
library(logrx) # Load logrx package
axecute("path-to-my-script/my-script.R") # Execute single program
lapply(c("path/my-script1.R","path/my-script2.R"), axecute) # Execute list of files

r_script_list <- list.files(path = "path-to-directory", pattern = "\\.R$")
lapply(r_script_list, axecute) # Execute all files in a directory
```

- Using logrxaddin, which is a simple point and click shiny-based interface that allows you to run a single program.

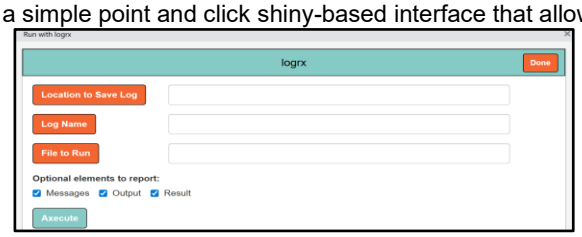


Image 07: A snippet of logrxaddin interface

```r
# Load packages
library(dplyr)
library(tidyr)
library(stringr)

# Read input data
dm <- pharmaversesdtm::dm

# Core Program Code
adsl <- dm %>%
  mutate(TRT01P = ARM, TRT01A = ACTARM) %>% #dplyr function
  select(USUBJID, TRT01P, TRT01A, AGE, AGEU, SEX, RACE, ETHNIC) %>% #dplyr function
  mutate_at(c("RACE","AGEU"), str_to_title) %>% #dplyr, stringr
  mutate(ASR = str_c(AGE, SEX, RACE, sep = "/"))

pretbl <- adsl %>%
  group_by(SEX, TRT01A) %>% #dplyr function
  count() #dplyr function

tbl <- pretbl %>%
  pivot_wider(id_cols= TRT01A,
              names_from = SEX,
              values_from = n) #tidyr function

print(tbl)

adsl_a <- adsl %>%
  #throws warning as USUBJID has alpha-numeric values
  mutate(USUBJID_new = as.numeric(USUBJID))

#Send messages to log
message("This is a custom message inserted with message()")

#Below code throws an error since haven package was not loaded
adsl_b <- read_sas(read_path(a_in, "adsl.sas7bdat"))
```

```
--------------------------------------------------------------------------------
-                              logrx Metadata                                  -
--------------------------------------------------------------------------------
This log was generated using logrx 0.4.0
logrx package version: 0.4.0
logrx build: CRAN (R 4.5.2)
logrx link to repository: https://github.com/pharmaverse/logrx
--------------------------------------------------------------------------------
-                         User and File Information                            -
--------------------------------------------------------------------------------
User: rango
File Name: logrxcode.R
File Path: C:/Users/rango/Desktop/R_Poster
File HashSum: c75bcefa45777f859c8e76fde4bb1d71f8e2fa88
--------------------------------------------------------------------------------
-                            Session Information                               -
--------------------------------------------------------------------------------
─ Session info ─────────────────────────────────────────────────────────────────
 setting  value
 version  R version 4.5.2 (2025-10-31 ucrt)
 os       Windows 11 x64 (build 26100)
 system   x86_64, mingw32
 ui       RTerm
 language (EN)
 collate  English_India.utf8
 ctype    English_India.utf8
 tz       Asia/Calcutta
 date     2025-11-26
 pandoc   NA
 quarto   NA @
    C:\\PROGRA~1\\RStudio\\RESOUR~1\\app\\bin\\quarto\\bin\\quarto.exe

─ Packages ─────────────────────────────────────────────────────────────────────
 package          * version date (UTC) lib source
 cli                3.6.5   2025-04-23 [1] CRAN (R 4.5.2)
 digest             0.6.38  2025-11-12 [1] CRAN (R 4.5.2)
 dplyr            * 1.1.4   2023-11-17 [1] CRAN (R 4.5.2)
 generics           0.1.4   2025-05-09 [1] CRAN (R 4.5.2)
 glue               1.8.0   2024-09-30 [1] CRAN (R 4.5.2)
 lifecycle          1.0.4   2023-11-07 [1] CRAN (R 4.5.2)
 logrx              0.4.0   2025-05-05 [1] CRAN (R 4.5.2)
 magrittr           2.0.4   2025-09-12 [1] CRAN (R 4.5.2)
 pharmaversesdtm    1.3.1   2025-08-21 [1] CRAN (R 4.5.2)
 pillar             1.11.1  2025-09-17 [1] CRAN (R 4.5.2)
 pkgconfig          2.0.3   2019-09-22 [1] CRAN (R 4.5.2)
 purrr              1.2.0   2025-11-04 [1] CRAN (R 4.5.2)
 R6                 2.6.1   2025-02-15 [1] CRAN (R 4.5.2)
 rlang              1.1.6   2025-04-11 [1] CRAN (R 4.5.2)
 rstudioapi         0.17.1  2024-10-22 [1] CRAN (R 4.5.2)
 sessioninfo        1.2.3   2025-02-05 [1] CRAN (R 4.5.2)
 stringi            1.8.7   2025-03-27 [1] CRAN (R 4.5.2)
 stringr          * 1.6.0   2025-11-04 [1] CRAN (R 4.5.2)
 tibble             3.3.0   2025-06-08 [1] CRAN (R 4.5.2)
 tidyr            * 1.3.1   2024-01-24 [1] CRAN (R 4.5.2)
 tidyselect         1.2.1   2024-03-11 [1] CRAN (R 4.5.2)
 utf8               1.2.6   2025-06-08 [1] CRAN (R 4.5.2)
 vctrs              0.6.5   2023-12-01 [1] CRAN (R 4.5.2)
 withr              3.0.2   2024-10-28 [1] CRAN (R 4.5.2)

 [1] C:/Users/rango/AppData/Local/R/win-library/4.5
 [2] C:/Program Files/R/R-4.5.2/library
 * ── Packages attached to the search path.

─ External software ────────────────────────────────────────────────────────────
 setting         value
 cairo           1.18.4
 cairoFT
 pango
 png             1.6.50
 jpeg            9.6
 tiff            LIBTIFF, Version 4.7.1
 tcl             8.6.17
 curl            8.16.0
 zlib            1.3.1
 bzlib           1.0.8, 13-Jul-2019
 xz              5.8.1
 deflate         1.24
 PCRE            10.46 2025-08-27
 ICU             77.1
 TRE             TRE 0.8.0 R_fixes (BSD)
 iconv           win_iconv
 readline
 BLAS
 lapack
 lapack_version 3.12.1

─ Python configuration ─────────────────────────────────────────────────────────
 Python is not available

--------------------------------------------------------------------------------
-                            Masked Functions                                  -
--------------------------------------------------------------------------------
function `plot` from {package:base} by package:graphics
function `body<-` from {package:base} by package:methods
function `kronecker` from {package:base} by package:methods
--------------------------------------------------------------------------------
-                         Used Package and Functions                           -
--------------------------------------------------------------------------------
{!!! NOT FOUND !!!} read_sas, read_path
{package:base} library, c, print, as.numeric, message
{package:dplyr} mutate, select, mutate_at, group_by, count
{package:stringr} %>%, str_c
{package:tidyr} pivot_wider
--------------------------------------------------------------------------------
-                      Program Run Time Information                            -
--------------------------------------------------------------------------------
Start time: 2025-11-26 16:44:03 IST
End time: 2025-11-26 16:44:04 IST
Run time: 1 seconds
--------------------------------------------------------------------------------
-                            Errors and Warnings                               -
--------------------------------------------------------------------------------
Errors:
  could not find function "read_sas"

Warnings:
  There was 1 warning in `mutate()`.
ℹ In argument: `USUBJID_new = as.numeric(USUBJID)`.
Caused by warning:
! NAs introduced by coercion
--------------------------------------------------------------------------------
-                         Messages, Output, and Result                         -
--------------------------------------------------------------------------------
Messages:
  Attaching package: 'dplyr'
  The following objects are masked from 'package:stats':    filter, lag
  The following objects are masked from 'package:base':    intersect, setdiff, setequal
  This is a custom message inserted with message()
Output:
  # A tibble: 4 × 3
  # Groups:   TRT01A [4]
    TRT01A                 F     M
    <chr>              <int> <int>
  1 Placebo               53    33
  2 Screen Failure        36    16
  3 Xanomeline High Dose  35    37
  4 Xanomeline Low Dose   55    41

Result:
  NULL
--------------------------------------------------------------------------------
-                            Log Output File                                   -
--------------------------------------------------------------------------------
Log name: logrxcode.log
Log path: C:/Users/rango/Desktop/R_Poster
```

Image 08: A snippet of log generated by logrx (right side) for the R script (on left)

The logrx also supports integration with tidylog. This combined use allows programmers to create a clear log with complete record of the R script's execution, including detailed step-by-step information on data transformations, which is essential for review and validation in a regulated industry.

```
--------------------------------------------------------------------------------
-                          Messages, Output, and Result                        -
--------------------------------------------------------------------------------
Messages:
  mutate: new variable 'TRT01P' (character) with 4 unique values and 0% NA
          new variable 'TRT01A' (character) with 4 unique values and 0% NA
  select: dropped 19 variables (STUDYID, DOMAIN, SUBJID, RFSTDTC, RFENDTC, …)
  mutate_at: changed 306 values (100%) of 'AGEU' (0 new NAs)
             changed 306 values (100%) of 'RACE' (0 new NAs)
  mutate: new variable 'ASR' (character) with 92 unique values and 0% NA
  group_by: 2 grouping variables (SEX, TRT01A)
  count: now 8 rows and 3 columns, 2 group variables remaining (SEX, TRT01A)
  pivot_wider: reorganized (SEX, n) into (F, M) [was 8x3, now 4x3]
  mutate: new variable 'USUBJID_new' (double) with one unique value and 100% NA
```

Image 09: A snippet of logrx log with messages logged by tidylog

Summarizing Batch Execution Logs with whirl

The whirl is an R package designed to log the execution of R scripts suitable for clinical environment and is part of the pharmaverse suite of packages. It generates nice human readable logs using 'Quarto'.

The primary function to execute the script and create log is run(), which takes the path to one or more scripts and executes them, producing a log file and a summary file in HTML format.

```
# Load whirl package
library(whirl)

#Execute single program and define name of the summary output file
run(input = "path-to-my-script/my-script.R", summary_file = "summary.html")
```

The whirl enables execution of scripts in batch, while simultaneously creating logs for the execution of each script, and providing an overview summary log of the entire batch execution. It is possible to define batches of scripts to run simultaneously using n_workers parameter.

```
# Executes 2 scripts in parallel
run(input = c("path/my-script1.R", "path/my-script2.R"), n_workers = 2)

# Executes all R scripts in the directory with up to 2 scripts in parallel
run(input = "path-to-directory/*.R", n_workers = 2)
```

A log generated by whirl contains information about below:

- Execution end date and time
- Path to script
- Summary status of the execution (Success/Error/Warning)
- Script that was executed along with the printed console output/messages/warnings, errors with full traceback
- The environment the script was executed under (Session info)
    - Platform – R version, Operating system, UI, external software etc.
    - Information about R packages that were utilized with optional feature to check approved packages
    - Environment variables and Options

A summary log generated by whirl consists of the list of all the programs executed with the corresponding program path, status of execution (Success/Error/Warning) and hyperlink to the program log.

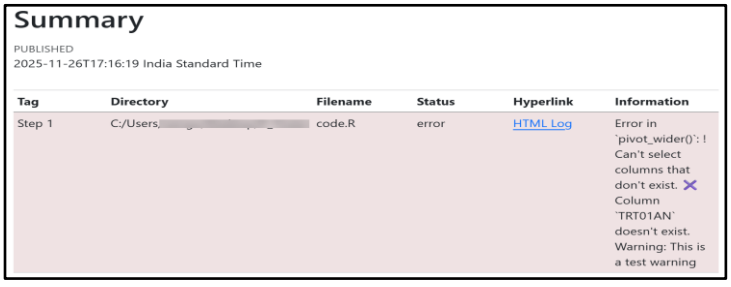


Image 10: Summary log generated by whirl

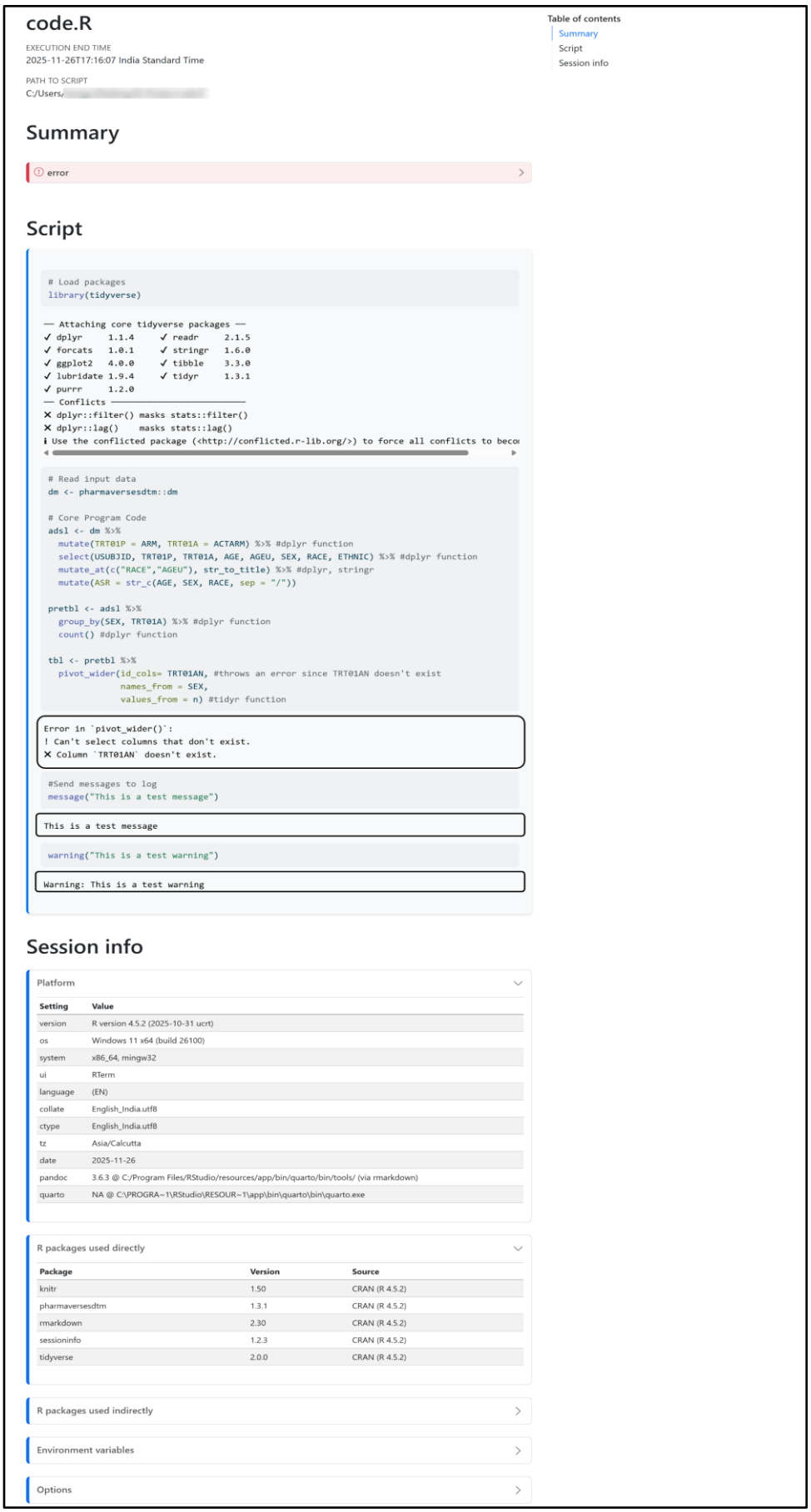


Image 11: A snippet of log generated by whirl with custom messages using base R functions

The whirl package can be integrated with tidylog. It also offers helper functions to log custom messages and can include linting results.

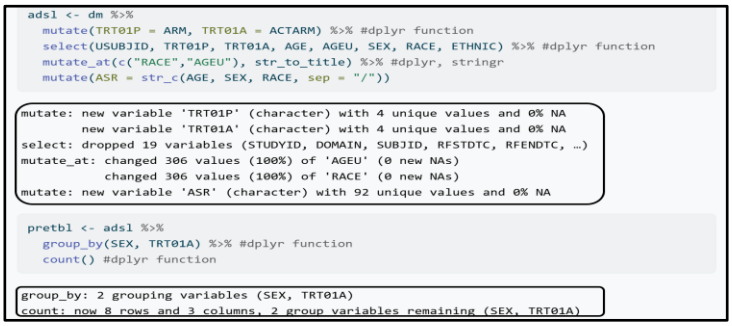


Image 12: Snippet of the log generated by whirl with tidylog integration

COMPARISON OF R LOGGING PACKAGES

Log	tidylog	logr	logrx	whirl
Type	• Standalone R package. • Automatic logging via wrapper functions	• Part of meta-package sassy. • Requires manual coding input by the programmer.	• Standalone R package. • Provides comprehensive log automatically.	• Part of pharmaverse. • Provides comprehensive log automatically.
Purpose	• Logging messages for data manipulation operations by dplyr and tidyr functions	• General SAS-like logging with severity levels - INFO/WARNING/ ERROR. • Offers greater control over the output.	• Advanced Logging Utility Focused on Clinical Trial Programming Workflows for compliance, traceability, reproducibility, environment capture.	• Facilitates execution of multiple scripts and provides a structured batch execution summary. • Individual execution logs.
Log Output	• Console output by default; Option for Text-based log	• Text-based log with Header, Body, Footer	• Text-based log reports with comprehensive sections, designed for readability and inclusion of program output.	• HTML log reports, including printed results, session information and packages information.

Table 03: Comparison of R Logging Packages

BENEFITS OF ENHANCED R LOG

Depending on the logging package chosen, an enhanced R log offers a powerful and structured log like SAS® which helps in clear identification of errors & warnings for easier troubleshooting, traceability and reproducibility.

R Log created by	Comparison with SAS® Log	Strengths
logr (with tidylog integration)	Like SAS® log for data steps with severity levels	Easy troubleshooting as the errors, warnings, messages are displayed at the place they occurred.
logrx	Comprehensive, Different structure - Log categorized into sections	Environment capture, reproducibility, easy and faster to use, structured logs.
whirl	Comprehensive, Different format - HTML log	Batch execution summary, easy navigation, Environment capture, reproducibility.

Table 04: Enhanced R Log versus SAS® Log

Also, while R by default does not return line numbers along with error or warning messages as we have in SAS®, R offers a detailed traceback for errors which can be written to the log while using any of these R logging packages.

In addition, the summary log created by whirl provides the programmer with a logparser-like output containing the list of programs from a batch execution that have errors or warnings. Likewise, the logs generated by logrx and the message files generated by logr could be utilized to get a list of all the programs with errors and warnings for quick review post batch execution using the log parsing approaches in R (with readLines() and grep() functions).

CONCLUSION

Each R logging package serves a distinct purpose. The logrx offers a comprehensive solution for submission compliance. The logr combined with tidylog offers a simpler way for programmers to track their analysis steps, debug and troubleshoot during development. The whirl package generates comprehensive logs suitable for clinical environment while offering summary of batch execution status.

Together, these packages can deliver logging capabilities in R that are equivalent to SAS®, effectively enhancing R workflow, making it more transparent and reproducible! Programmers can get enhanced R user experience by adopting these powerful logging packages in R workflows and incorporating customized logging features as needed.

REFERENCES

- Packages referred in this paper: **tidylog v1.1.0, logrx v0.4.0, logr v1.3.9, whirl v0.3.1, logrxaddin v0.0.1**.
- https://www.r-project.org/doc/R-FDA.pdf
- https://cran.r-project.org/web/packages/logr/
- https://cran.r-project.org/web/packages/tidylog/
- https://cran.r-project.org/web/packages/logrx/
- https://cran.r-project.org/web/packages/whirl/
- Straub, Ben. 2025. “Logrx 0.4.0: Logs for Your R Programs.” May 19, 2025. https://pharmaverse.github.io/blog/posts/2025-05-19_logrx_0.4.0…/logrx_0_4_0_logs_for_your__r_programs.html
- https://pharmaverse.github.io/examples/logging/logging.html
- https://github.com/pharmaverse/logrx/issues/270

ACKNOWLEDGMENTS

I thank my colleagues and leadership for their support and valuable input during the preparation of this paper.

CONTACT INFORMATION

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

Author Name: Vishnu Priya Vishnubhotla
Company: Johnson & Johnson
Address: Johnson & Johnson, 21st floor, RMZ Nexity, Silpa Gram Craft Village, HITEC City, Hyderabad, Telangana 500032, India
Work Email: vvishnub@its.jnj.com
Alternative Email: vishnupriyavishnubhotla@gmail.com

DISCLAIMER

The views and opinions expressed in this paper are those of the author and do not necessarily reflect the views and policies of Johnson & Johnson.