Public



Translating SAS Analysis Macros into R Functions: Practical Strategies and Key Insights

Hiroaki Fukuda, MSD K.K.

He has worked as a statistical programmer at MSD K.K. since December 2015. He has 12+ years of SAS programming and 4+ years of R programming experience. His responsibilities include conducting statistical analyses for clinical trials, supporting electronic data submissions to the PMDA, and contributing to the development and maintenance of organizational R packages for clinical trial analysis and reporting.

Public



Translating SAS Analysis Macros into R Functions

Practical Strategies and Key Insights

Public



Disclaimer

The views and opinions expressed in this presentation are solely those of the speakers and do not necessarily reflect the official policy or position of any organization or entity with which they are affiliated.

Public

Agenda

Introduction

1. Translation from SAS Macros to R Functions

- Translation Strategy
- Strengths & Aspects Requiring Careful Consideration
- Case Study of MMRM
- Considerations for Function Design

2. Practical Workflow to Generate TLFs

Summary

Public

Introduction

Background & Motivation



Industry Shift from SAS

Growing interest in OSS, especially R, with some already transitioning and others still evaluating

Existing SAS Assets

Established workflows, reporting standards, and accumulated knowledge

Natural Approach

Reuse of existing assets for less redevelopment and consistency with established processes

Objective of Presentation

Sharing practical experience in translating SAS analysis macros into R functions

Public

Basic Design Principles

Preserve processing and analytical logic

One SAS macro

```
%macro xxx(...);
```

ADaM datasets (input)

↓

Preprocessing

↓

Analysis

↓

RTF output

Convert →

One R function

```
xxx <- function(...) {
```

Leverage native R features

- Natural implementation
- Simple code
- Wider tools

Public



Translation Strategy

Examples of Leveraged R Features

Flexible data structures and classes

```
c("val1", "val2", "val3")
list(
  var1 = data1,
  var2 = data2,
  var3 = data3
)
```

Vectors and Lists

Multiple inputs are passed directly and iterated over concisely

```
display <- TRUE

if (display) {
 ...
}
```

Logical Values

Directly used for conditional functions which can simplify code

```
factor(
  c("F", "M"),
  levels = c("M", "F")
)
```

Factors

Labels and ordering live in one variable, driving both analysis and display order

Public



Translation Strategy

Package Dependencies

Minimum package dependencies for simplicity and reproducibility

- R built-in functionality (`base`, `stats`, etc.)
- `tidyverse` framework data manipulation
- Analysis-specific reliable packages
- RTF generation using `r2rtf`

Public

Strengths & Aspects Requiring Careful Consideration

Lesson 1: Multiple Inputs with Ordering Information

A delimited macro parameter becomes a character vector plus factors

SAS Macro

```
%macro xxx(analysis_data, analysis_vars, ...);
%* Parse analysis_vars into display and ordering variables;
%* e.g., _analysis_var1=AGEGR1, _analysis_fmt1=AGEGR1N;
...
  select trt01pn, trt01p,
         &&_analysis_var&i.., &&_analysis_fmt&i..,
         count(distinct usubjid) as n
  from &analysis_data.
  group by trt01pn, trt01p,
           &&_analysis_fmt&i.., &&_analysis_var&i..
  order by trt01pn, &&_analysis_fmt&i..;
%mend;

%xxx(
  analysis_data = adsl,
  analysis_vars =
AGEGR1^AGEGR1N|SEX^SEXN|RACE^RACEN,
 ...);
```

R Function

```
xxx <- function(analysis_data, analysis_vars, ...) {
  res <- lapply(analysis_vars, function(x) {
    analysis_data |>
      dplyr::group_by(dplyr::across(
        dplyr::all_of(c("TRT01P", x)))) |>
      dplyr::summarise(n = dplyr::n_distinct(USUBJID),
                       .groups = "drop")
  })
}

adsl$TRT01P <- factor(adsl$TRT01P,
  levels =
unique(adsl$TRT01P[order(adsl$TRT01PN)]))
...

xxx(analysis_data = adsl,
    analysis_vars = c("AGEGR1", "SEX", "RACE"),
    ...)
```

No macro-parameter parsing, and no separate display and ordering variables to carry through the workflow.

Public

Strengths & Aspects Requiring Careful Consideration

Lesson 2: Input Checks and Default Values

%if validation blocks collapse into a few built-in R calls

SAS Macro

```
%macro xxx(..., time_unit = M, font_size = 9);
%* Assign default when blank;
%if %nrbquote(&time_unit) = %nrbquote( ) %then
   %let time_unit = %str(M);

%* Check inputs;
%if %sysfunc(compress(&time_unit,%str(DMYW))) ne %str()
   %then %do;
  %put &err&orr: Valid values for time_unit: D, W, M or Y.;
  %goto exit;
%end;

%if %sysfunc(verify(%qcmpres(&font_size),0123456789+.))
   %then %do;
  %put &err&orr: Invalid values: font_size must be numeric.;
  %goto exit;
%end;
```

R Function

```
xxx <- function(
   ...,
   time_unit = c("M", "D", "W", "Y"),
   font_size = 9
) {

  time_unit <- match.arg(time_unit,
                         c("M", "D", "W", "Y"))

  if (!is.numeric(font_size)) {
    stop("Invalid values: font_size must be numeric.")
  }
}
```

Built-in functions express parameter validation and default handling more clearly and concisely

Public

Strengths & Aspects Requiring Careful Consideration

Lesson 3: Variable Names and Expressions

SAS macros pass parameters as text; R functions evaluate arguments

R Function

```
xxx <- function(...) {

# For expression-style arguments
if (!is.null(substitute(analysis_extract))) {
  ...
}

# For character-string arguments
if (!is.null(analysis_var)) {
  ...
}

xxx(
  analysis_data = advs
 ,analysis_extract = TRTFL == "Y" & ANL01FL == "Y"
 ,analysis_var = "CHG"
 ,...
)
```

Variable names as character strings

- Easier manipulation
  e.g., data[[arg]], apply(arg, ...)
- Directly use `is.null()` for input validation

Conditions kept as expressions

- More intuitive and readable inputs
- Capture an expression argument with substitute() before checking whether it was supplied, since the argument must remain unevaluated

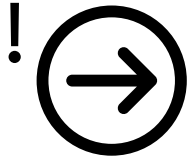
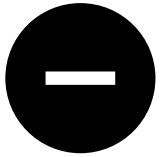
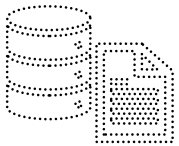
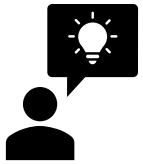
Unlike SAS macros, different handling approaches were required depending on the arguments.

Public

Strengths & Aspects Requiring Careful Consideration

Lesson 4: Error Handling and Stability

An unhandled error in R stops the whole function

	SAS Macro	R Function
On Error	 Continues past failures	 Terminates immediately
Intermediate Results	 Logs and data remain available	 Objects are generally local to a function
Debugging	 Start from what was produced	 Errors raised outside own checks may be hard to interpret



Early input validation and explicit error handling are important especially in R.

Public



Case Study of MMRM

Overview: PROC MIXED to mmrm + emmeans

mmrm (0.3.17)

emmeans (2.0.1)

Providing functionality commonly used in clinical-trial MMRM analyses:

- Flexible covariance structures
- Kenward-Roger adjustments
- Least-squares means estimation

Corresponding implementations are readily available and reported to be highly consistent with PROC MIXED.

Example Code	SAS: PROC MIXED	R: mmrm + emmeans
Model & Covariance	`model / repeated type=un`	mmrm: `formula = ...` `covariance = cov_struct("us", ...)`

Estimation and DF	`method = reml`
	`ddfm = KR`
		mmrm: `reml = TRUE`
		`method = "Kenward-Roger"`
		`vcov = "Kenward-Roger-Linear"`

LS means	`lsmeans`	emmeans: `emmeans(fit, ~ TRTP | AVISIT)`
Comparison	`lsmeans / pdiff cl`	emmeans: `contrast()`

Several aspects requiring careful attention were found in the R implementation

Public

Case Study of MMRM

Point 1: Reference Level Setting

`mmrm::mmrm()` has no class(*ref* =) equivalent

```
adeff$TRTP <- relevel(factor(adeff$TRTP), ref = "Control")
# If the numeric variable is used for analysis
adeff$TRTPN <- relevel(factor(adeff$TRTPN), ref = "2")
...

# Model Fitting
fit <- mmrm::mmrm(...)

# Obtain difference at each visit
diff <- emmeans::emmeans(...) |>
  emmeans::contrast(
    method = "trt.vs.ctrl",
    ref = "Control",
    adjust = "none"
  )
```

SAS

- class (*ref* =) option can set reference level at once

R

- mmrm::mmrm() has no such argument
- emmeans::contrast() has a ref argument, but it only affects the comparisons

Set the factor levels, so the reference category can be controlled consistently throughout the analysis.

Public

Case Study of MMRM

Point 2: Convergence Status

Failed convergence raises an error and stops the run

Model fitting with `mmrm::mmrm()`

- **Converged**: Convergence status is stored in the fitted object
- **Failed**: An error is raised, and no convergence status is available

```
fit <- tryCatch(
  mmrm::mmrm(...),
  error = function(e) NULL    # record and continue
)
```

Capturing errors allows convergence issues to be detected and managed without interrupting the overall workflow.

Public

Point 3: Impact of Data and Analysis (1)

Limited data during an ongoing study produced different confidence intervals

Scenario

- Limited data like an ongoing study
- 6 participants: Drug X vs. Control (1:1)
- Time points: Baseline, Week 4, 8
- The Control participants had not yet reached Week 8 at the time of analysis
- Same MMRM code as shown in the previous example

LS means

TRTP	AVISIT	SAS Estimate	SAS Lower	SAS Upper	R Estimate	R Lower	R Upper
Drug X	Week 4	1.77	-4.28	7.81	1.77	-4.31	7.85
Drug X	Week 8	3.42	-15.34	22.19	3.42	-1.41e+12	1.41e+12
Control	Week 4	-0.58	-10.07	8.90	-0.58	-10.15	8.98

Public



Case Study of MMRM

Point 3: Impact of Data and Analysis (2)

Model specification changed the agreement

Scenario

- 180 participants: Drug X vs. Control (1:1), three strata levels
- Time points: Baseline, Week 4, 8, 12, and 24
- cLDA method using essentially the same framework as MMRM
  - Assume a common mean across treatment groups at baseline and a different mean for each treatment at each of the post-baseline time points

Week 24 LS means

TRTP	SAS Estimate	SAS Lower	SAS Upper	R Estimate	R Lower	R Upper
Model 1						
Drug X	2355.4	1728.4	2982.4	2355.4	1728.4	2982.4
Control	2659.5	2080.3	3238.6	2659.5	2080.3	3238.6
Model 2						
Drug X	2270.9	1625.8	2916.1	2310.8	1668.2	2953.4
Control	2606.1	1984.2	3227.9	2645.9	2052.4	3239.4

Model 1: with treatment, visit, and treatment * visit

Model 2: with treatment, visit, strata, treatment * visit, and strata * visit

Case Study of MMRM

Point 3: Impact of Data and Analysis (2)

Model specification changed the agreement

Handling discrepancies that can arise

Scenario

- ...trol
- ...,
- ...he

Assume a common mean across treatment groups at baseline and a different mean for each treatment at each of the post-baseline time points

Week 24 LS means

TRTP	SAS	SA	SAS	R	R	R
			32			6
Model 2						
Drug X	2270.9	1625.8	2916.1	2310.8	1668.2	2953.4
Control	[illegible]	[illegible]	3227.9	2645.9	2052.4	3239.4

Model 1: with treatment, visit, and treatment * visit

Model 2: with treatment, visit, strata, treatment * visit, and strata * visit

Important!

Set the principles

Establish clear principles for the extent of investigation and appropriate handling strategies

Example

Assess the impact

Consider how far each difference affects the interpretation of the results.

Decide the action

A difference may be considered acceptable, or **additional action may be required**.

e.g.,

- Search another package
- Implement a custom function

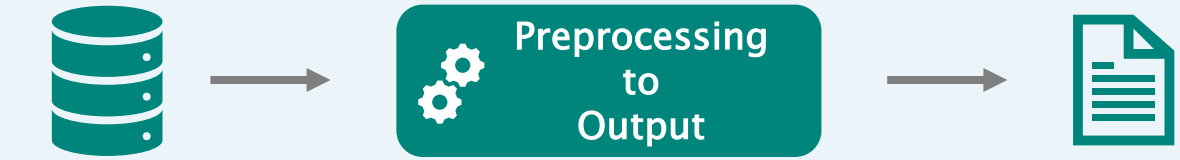
Public

ARS: Analysis Results Standard
ARD: Analysis Results Data

Considerations for Function Design

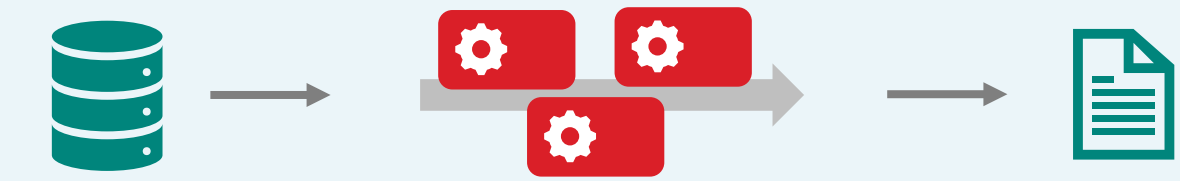
A Modular Design Would Work Better

Implemented



Suggested

More modular approach



e.g., Preprocessing, Analysis, Output

Easier troubleshooting

R stops at the first error, so splitting the workflow into steps makes the cause much easier to identify.

Inspectable intermediates

Checking the dataset at each step shows whether a difference came from processing or from a package or function.

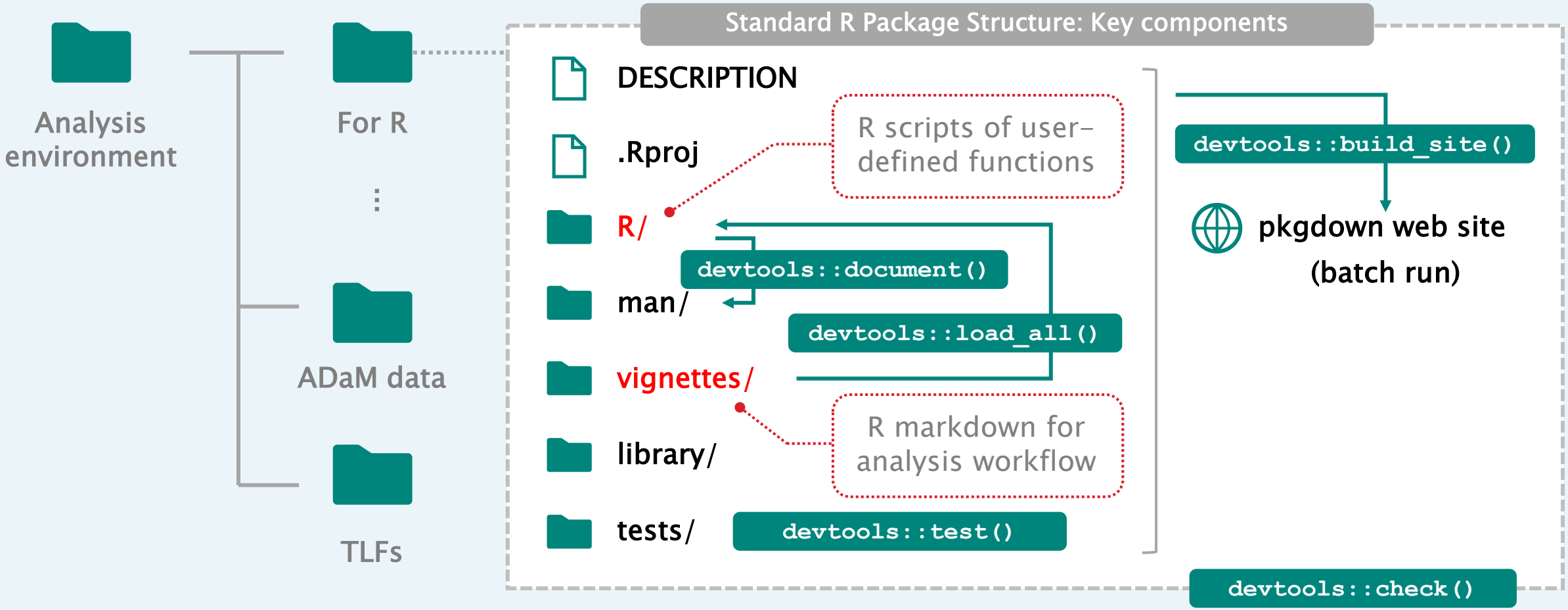
Extensibility with ARS

Keeping analysis results data separate from output generation makes future integration with the CDISC ARS, particularly its ARD representation, straightforward.

Public

Practical Workflow to Generate TLFs

R Package as Analysis Environment



Essential processes are not overlooked, since validation, documentation, and testing run systematically.

Public



Summary

- **Shared the practical experience in translating SAS macros into R functions**

  Implementing the R way, rather than mirroring SAS syntax, worked well and let the same logic be written more simply.
- **Write robust code that anticipates errors**

  Since an unhandled error stops the whole function in R, it is important to validate inputs and handle exceptional situations explicitly.
- **Evaluate results with the expected data and consider how to handle differences**

  Analysis results are better assessed using data that reflects the situations expected in the study. In addition, when results differ from SAS, decide the response beforehand.
- **A modular implementation may be better**

  Based on this implementation experience, splitting the workflow into several modules is easier to work with than a single function.

Hopefully, these findings and the implementation shared here serve as a practical reference for moving to an R environment.