

Gen AI-Assisted Conversion: from SAS to R for Pharmacokinetic Summary Analysis

Shiqi Lin, Runcheng Li, Yiyuan Zeng
Merck & Co., Inc., Rahway, NJ, USA

ABSTRACT

SAS has traditionally been the primary tool for data analysis and reporting in the pharmaceutical industry. However, the increasing popularity of R and its open-source nature have created a growing demand for program conversion. Manual conversion is time-consuming, error-prone, and requires significant expertise. This paper discusses a Gen AI-assisted approach aimed at streamlining the conversion from SAS to R, enabling SAS programmers with limited experience in R to initiate the conversion process. Using a pharmacokinetic (PK) summary table program as a case study, we decompose a SAS macro into key modules, convert it to R code with the assistance of a Gen AI tool, and compare the results between SAS and R to assess the accuracy of this approach.

INTRODUCTION

A pharmacokinetic (PK) summary table is essential for regulatory submissions and clinical development decision-making. It provides a tabular representation of key PK parameters, such as C_{max} (maximum concentration), T_{max} (time to maximum concentration), AUC (area under the curve), and t_{1/2} (half-life). By using summary statistics across different treatment groups and conducting statistical tests (e.g., t-tests, ANOVA), these tables enable the assessment of drug efficacy.

There is a growing trend in the industry to develop programs compatible with R. However, not all SAS programmers have experience or a background in using R. Gen AI tools provide a solution to this challenge by taking SAS code as input and returning the corresponding R code, along with an explanation of the thought process and reasoning behind each step. Additionally, these tools facilitate quick debugging of the programs, such as identifying missing delimiters or unclosed brackets. This allows beginners in R to successfully complete the program conversion.

The primary objective of this paper is to convert an existing SAS macro for generating early-phase, non-oncology PK summary tables to R. To achieve this, we will focus on key steps in the conversion process. The SAS dataset will be imported into R using the `read_sas()` function from the `haven` package to serve as the input data. The SAS macro will be divided into several sections to serve as input for the Gen AI tool. The accuracy of the R-based implementation will be assessed by comparing its output to the original SAS results. To streamline the conversion process, we will leverage GPTeal (an AI tool developed by Merck & Co., Inc., Rahway, NJ, USA, based on OpenAI's GPT-4o mini model) to facilitate the conversion of the SAS program. Finally, the post-Gen AI conversion process will be discussed to help quickly identify programming logic discrepancies between SAS and R.

Breaking Down SAS Macros Improves Gen AI-Assisted Conversion to R

Translating a complex SAS macro into an R function is challenging because of fundamental differences between the two languages. SAS macros often include nested loops, conditional statements, and dynamic variable handling, making it difficult for Gen AI to understand the complete logic in one go. Additionally, SAS procedures like PROC MIXED have no direct equivalent in R, requiring careful adjustments. When Gen AI attempts to convert the entire macro at once, it often misinterprets dependencies, skips important logic, or generates incorrect code. This is because SAS macros generate code dynamically, whereas R functions require clear function definitions and explicit parameter handling.

To improve accuracy, we divide the SAS macro into separate steps such as data extraction, transformation, model fitting, and output generation before conversion. This approach allows Gen AI to focus on one task at a time, ensuring that each function is correctly mapped to its R equivalent. For example, instead of translating a large SAS macro at once, we first convert data processing steps using `dplyr`, then handle statistical modeling using `lme4`, and finally format outputs with `emmeans`. By doing this, we can validate each step individually, reduce errors, and make debugging easier. Breaking the macro into parts also allows for manual refinements at each stage, improving the overall quality of Gen AI-assisted code conversion. This structured approach ensures that the final RStudio implementation closely matches the original SAS workflow, making the conversion smoother and more reliable.

PK Summary Table Macro Modules

This case study encompasses six significant steps: environment setup, defining macro parameters, conducting model-based analysis for PK data, final dataset cleaning, preparing the RTF output, and comparing the results between the original SAS code and the R code.

1. Environment Setup

The ADPP (Pharmacokinetic Parameters Analysis Dataset) is frequently used in PK analysis to understand how drugs behave in the body. In this case study, ADPP serves as the sole input dataset for the PK summary table and has been properly validated.

To make the SAS dataset (`.sas7bdat`) readable in R, the `haven` package is utilized to import the data. After installing and loading this package, the `read_sas()` function is employed to read the dataset directly into R as a data frame. Moreover, GPT-4o is capable of recognizing SAS statements such as `where`, `keep`, and `sort`, converting them into similar statements like `mutate`, `select`, and `arrange` in R.

When creating PK tables, the SAS program generates not only the RTF table but also `.log` and `.lst` files, which are saved in different subfolders. The similar folder structure is expected for clinical trials analysis and reporting (A&R) when using R programs. In this exploratory, we save the output in `.rtf` format by using the `r2rtf` package, placing it in the same location as the ADPP.

Sample prompt:

```
Please create a template to set up file path, read SAS dataset ADPP, and subset the ADPP dataset.
```

R template (excerpt) by GPT-4o:

```
# install.packages("haven")
library(haven) # For reading SAS datasets

# Define the file path
file_path <- "//your_dataset_path/adpp.sas7bdat"

# Read the SAS file
adpp_data <- read_sas(file_path)

# Create the adpp dataframe and manipulate it
adpp_filtered <- adpp_data %>%
  mutate(
    # Add your mutation logic here
  ) %>%
  filter(# Add your mutation logic here)
print(adpp_filtered)
```

2. Define Macro Parameters

Similar to SAS macros, R allows for the definition of parameters that can be passed into the function. The code conversion can be achieved by copying the SAS syntax into the chat box and asking GPT-4o to “convert this SAS macro call to R”.

The sample SAS macro call (excerpt) below is designed for PK summary tables. It includes parameters for selecting and subsetting input dataset, specifying variables such as treatment arm, subject ID, and analysis value as covariates, as well as choosing specific parameter values into different types of analysis.

This step is not necessarily required when defining an R function, as R handles parameters differently compared to SAS macros. In SAS, parameters are defined within the macro, while in R, parameters are typically passed as arguments to the function. The main purpose of this discussion is to examine whether Gen AI can understand the differences in dataset structure and parameter handling between SAS and R.

```
esdr0pp0summary0sd(  
/* Input dataset name for observation dataset */  
  observation_from = adpp  
/* Valid SAS where statement to subset observation_from dataset */  
  ,observation_where=  
/* Variable name for treatment group code */  
  ,therapy_cd_var   = TRTA  
/* Variable name for treatment group description */  
  ,therapy_des_var  = TRTAN  
/* PK parameter to be used in the model */  
  ,back_trans_param = 'AUCIFP' 'CMAX' 'CONC24'  
/* PARAMCD to be calculated for Median, Min, Max */  
  ,median_minmax_param = 'TMAX'  
/* PARAMCD to be calculated for geometric mean */  
  ,gm_cv_param        = 'LAMZHL'  
/* Concentration variable default= AVAL */  
  ,aval_var           = aval  
/* Subject variable (required) default= USUBJID */  
  ,subject_var        = usubjid  
...  
)
```

R function syntax by GPT-4o:

```
esdr0pp0summary0sd <- function(observation_from = adpp,  
                                observation_where = NULL,  
                                therapy_cd_var = "TRTA",  
                                therapy_des_var = "TRTAN",  
                                back_trans_param = c('AUCIFP', 'CMAX', 'CONC24'),  
                                median_minmax_param = c('TMAX'),  
                                gm_cv_param = c('LAMZHL'),  
                                aval_var = "aval",  
                                subject_var = "usubjid") {  
...  
}
```

In this attempt, we observed that GPT-4o has gradually improved its programming capabilities in the background. In the recent attempt, it has demonstrated the ability to identify the programming logic differences between SAS and R, particularly in how missing parameters are assigned as 'NULL'. It also recognizes the variable PARAMCD as a vector, converting the parameters such as 'TMAX' to c('TMAX'). However, if the input parameter is in lowercase, such as 'usubjid', it still requires a manual

adjustment to change the variable name to uppercase 'USUBJID' for compatibility with R, as R is case-sensitive.

3. Model Based Analysis for PK Data

PK modeling is a major component of PK analysis, and the model to be used is typically defined in the study protocol. In single-dose studies, for each PK parameter, individual values of PARAMCD at each dose level is natural log-transformed and evaluated with a linear mixed effects model containing a fixed effect for treatment and a random effect for participant. The primary output values include subject counts (n), geometric means (GM), 95% confidence intervals (95% CI), and the square root of the conditional mean squared error (RMSE).

In SAS, this approach is divided into two steps: first, calculating the log-transformed PK parameters using the PROC MIXED function, and second, calculating the necessary variables based on the model outputs.

SAS syntax:

```
* Proc Mixed analysis of log-transformed pk-parameters;
option nodate nonumber;
title 'Mixed model analysis';
proc mixed data=Model_data order = data;
  by paramcd ;
  class therapy_des subject_var;
  model lnpkval=therapy_des /ddfm=kr outpred=pred;
  random subject_var ;
  lsmeans therapy_des / alpha=0.05 cl ;
  ods output lsmeans=lsmeans_1
             covparms=covparms_1;
run;
```

In this instance, the SAS syntax is provided as input to GPT-4o, which is tasked with generating a function that groups by PARAMCD and calculates n, GM, 95% CI, and RMSE. It can return a model template that meets these requirements. Furthermore, it can generate alternative code that breaks down each calculation step, allowing for the validation of results. However, manual checking is still necessary at this stage to ensure that GPT-4o has included all the necessary covariables in its output.

Sample prompt:

```
Please convert this SAS macro into R, using a single function to calculate the geometric mean (GM), count (n), 95% confidence interval (CI), and root mean square error (RMSE). Remember to group the calculations by PARAMCD.
```

R Syntax (excerpt) for single dose linear model by GPT-4o:

```
# Fit the mixed model by paramcd and calculate GM, CI, and RMSE
# Define a function for fitting the model and obtaining EMMS

fit_model_and_get_emms <- function(data) {
  model <- lmer(lnpkval ~ therapy_des_var +(1 | subject_var), data = data)
  # Fit the mixed-effects model
  emm <- emmeans(model, ~ therapy_des_var) # Estimated marginal means
  emm_summary <- summary(emm, infer = TRUE) # Get the summaries with
  confidence intervals

  # Get residual variance from the model
  var_corr <- VarCorr(model) # Get variance components
  residual_variance <- attr(var_corr, "sc")^2 # Extract the residual
  standard deviation and square it to get variance
  rmse <- sqrt(residual_variance) # Calculate RMSE
```

R Syntax (excerpt) breakdown by GPT-4o:

```
# Back transformation to get geometric means and 95% CI
ls_means <- model_results %>%
  mutate(Estimate1 = exp(estimate),
         Lower1 = exp(lower.CL),
         Upper1 = exp(upper.CL)) %>%
  mutate(gm = round(Estimate1, 3),
         ci95 = paste0("(", round(Lower1, 3), ", ", round(Upper1, 3), ")"))
%>%
  select(paramcd, therapy_des, gm, ci95)

# RMSE calculation
rmse <- covparms_1 %>%
  filter(CovParm == 'Residual') %>%
  left_join(parammeta, by = "paramcd") %>%
  mutate(rmse = sqrt(estimate)) %>%
  select(paramcd, rmse)
```

In addition to single-dose studies, linear model used for multiple-dose studies includes an additional fixed effect: day, as well as the interaction of dose and day.

```
class subject_var therapy_des day_var;
model lnPKval=therapy_des day_var therapy_des*day_var/DDOFM=KR
OUTPRED=PRED;
```

By providing this new model as an input, it becomes straightforward to ask GPT-4o to generate a new R function for multiple-dose model based on the previous defined single-dose model.

Sample prompt:

```
There is another SAS linear model, with day_var as the additional
covariate. I will provide the new SAS model. Please compare it with the
previous SAS model, update the R function you generated, and do not change
the structure too much.
```

R syntax for multiple dose linear model (excerpt) by GPT-4o:

```
fit_model_and_get_emms <- function(data) {
  model <- lmer(lnPKval ~ therapy_des_var * day_var + (1 | subject_var),
    data = data) # Fit the mixed-effects model with interaction
  emm <- emmeans(model, ~ therapy_des_var * day_var) # Calculate estimated
  marginal means
}
```

With the assistance of GPT-4o, R functions for both single-dose and multiple-dose study designs have been developed. This encompasses the majority of study designs for PK summary tables.

4. Final Dataset

PK parameters and descriptive statistics derived from these concentrations should in general be reported to three significant digits. Note that rounding to three significant digits will result in at most a 0.5% difference from the unrounded exact value, a difference which can generally be considered negligible. The `round()` function does not meet the requirement. When describing the demand as input, GPT-4o is able to return the command in the form of a specification, along with a function that can be used to format these values accordingly. Moreover, GPT-4o generates random numbers to verify the function, allowing the user to validate the function and request adjustments if it doesn't meet the expectation in the initial attempt. This also indicates that Gen AI tools like GPT-4o can recognize specification details and then create utility functions.

Sample prompt:

Please create a function, to keep 3 scientific number. For value ≥ 100 , round it to first three digits, i.e. 1988 show as 1990, 150.6 show as 151, don't keep decimal place; for value $10 < x < 100$, round it to xx.x, keep 1 decimal place; for value $1 < x < 10$ round it to x.xx, keep 2 decimal places; for value $x < 1$ round it to 0.xxx, keep 3 decimal places.

Sample specification by GPT-4o:

Rounding Values:
For values ≥ 100 : Computes the order of magnitude and rounds to the nearest significant figure (e.g., 1988 becomes 1990).
For values between 10 and 100: Rounds to one decimal place (e.g., 15.06 becomes 15.1).
For values between 1 and 10: Rounds to two decimal places (e.g., 9.999 becomes 10.00).
For values less than 1: Rounds to three decimal places (e.g., 0.999 becomes 0.999).

R syntax by GPT-4o:

```
# Custom function to format values to three significant digits with
detailed rules
format_value <- function(x) {
  if (is.na(x)) {
    return(NA) # Handle NA values
  }

  x_signif <- signif(x, 3) # Get three significant digits

  # Handle zero explicitly
  if (x_signif == 0) {
    return(format(0, nsmall = 2)) # Return '0.00'
  }

  # Different formatting based on the value
  if (abs(x_signif) < 1) {
    # For numbers less than 1, keep three decimal places
    return(format(round(x_signif, 3), nsmall = 3))
  } else if (abs(x_signif) >= 1 && abs(x_signif) < 10) {
    # For numbers between 1 and 10, keep two decimal place
    return(format(round(x_signif, 2), nsmall = 2))
  } else if (abs(x_signif) >= 10 && abs(x_signif) < 100) {
    # For numbers between 10 and 100, keep one decimal place
    return(format(round(x_signif, 1), nsmall = 1))
  } else if (abs(x_signif) >= 100) {
    # For numbers greater than or equal to 100, no decimal places
    return(format(round(x_signif, 0), nsmall = 0))
  }
}
```

Expected Output by GPT-4o:

	original	rounded
1	1988.0	1990
2	150.6	151
3	99.9	100

4	9.999	10.0
5	0.999	0.999
6	10.5	10.5
7	0.1	0.100
8	1234.6	1230
9	5000.0	5000
10	0.003456	0.003

By reviewing the sample output numbers above, we can confidently conclude that the rounding function works as expected.

5. Prepare RTF

In most situations, there is a powerful company-standard SAS macro to generate RTF tables for various structures. Several sub-macros are built into the main macro that automatically select the column variables, adjust column widths, and manage page breaks, among other functions. The logic for RTF table function is far more complex than that for a single PK summary macro, and converting the SAS RTF macro into R with robust functions would require a separate discussion.

GPT-4o allows users to upload screenshots of mockups to help it understand the scope of the task. It provides RTF template code based on R packages. However, multiple iterations of communication with GPT-4o are needed to adjust the details like columns, title, and footnotes. Even when provided with the final dataset structure, the more complex the mockup, the more prone the returned code is to errors. Moreover, GPT-4o sometimes provides R functions that do not exist, which causes difficulty and makes it more time-consuming for users to modify the program. Therefore, a more effective approach, for now, is to reference existing R code from similar table structures and then modify it to meet specific table needs with GPT-4o's assistance. At this stage, this section of code is study-specific and requires modifications to be compatible with other table structures.

Sample prompt:

Please provide a sample R function using the r2rtf package based on the dataset created in the previous step, and the uploaded mockup screenshot.

R Syntax (excerpt) for RTF by GPT-4o:

```
library(r2rtf)

# Define the function for generating the summary statistics table
generate_summary_table <- function(data) {
  # Create an RTF document
  rtf_doc <- rdoc_new()
  # Add title
  rtf_doc <- rtf_add_paragraph(rtf_doc, "xxxxx", bold = TRUE, size = 12)

  # Add the first section header
  rtf_doc <- rtf_add_paragraph(rtf_doc, "PK Statistics", bold = TRUE)

  # Create a table from the dataset
  # For example, expecting 'data' to have columns: Parameter, GM, CI, etc.
  table_data <- data.frame(
    Parameter = data$Parameter, # Assuming this column exists
    Panel_A_GM = data$Panel_A_GM, # Update column names based on dataset
    Panel_A_CI = data$Panel_A_CI,
    N = data$N # Ensure this column exists as well
  )

  # Add the table to the RTF document
```

```

rtf_doc <- rtf_add_table(rtf_doc, table_data, style = "Table Grid")

# Add footnotes or additional comments
rtf_doc <- rtf_add_paragraph(rtf_doc, "xxxx")

# Save the RTF document
rtf_file <- "summary_statistics_table.rtf"
rdoc_save(rtf_doc, rtf_file)

# Return the file path
return(rtf_file)
}

```

R Syntax (excerpt) for RTF created by user:

```

table_rtf <- filtered_data%>%
  rtf_page(orientation = landscape) %>%
  rtf_title(title = title_txt,
            subtitle = subtitle ) %>%
  rtf_colheader(colheader = "PK Parameters | Treatment Group | n | GM |95%
CI",
  col_rel_width = c(2.5,7,2,2, 5)) %>%
  rtf_body(col_rel_width = c(2.5,7,2,2,5),
  text_justification = c("c", "c", "c", "c", "c"),
  group_by="PARAMCD2") %>%
  rtf_footnote(footnote = c("XXX", "XXX")) %>%
  rtf_source(source = data_source_txt,
  text_justification = "l",
  as_table=FALSE)

```

6. Result Compares: SAS vs. R

As shown in Table 1, the R program converted by GPT-4o successfully reproduces the values generated by the SAS program. The AUC results include back-transformed least squares mean and confidence interval, the Tmax (h) reports median (min, max), and the t1/2 (h) reports geometric mean and percent geometric coefficient of variation (CV).

Dummy values are presented here for illustrative purposes.

Pharmacokinetic Parameters	Treatment Group	SAS Results			R Results		
		n	GM	95% CI	n	GM	95% CI
AUC0-inf (h*nM)	Treatment A	6	3070	(2360, 3990)	6	3070	(2360, 3990)
	Treatment B	12	67.7	(49.2, 93.0)	12	67.7	(49.2, 93.0)
	...						
Tmax (h)	Treatment C	6	2.50	(1.00, 6.00)	6	2.50	(1.00, 6.00)
	Treatment D	12	5.00	(2.00, 6.00)	12	5.00	(2.00, 6.00)
	...						
t1/2 (h)	Treatment E	6	8.20	67.0	6	8.20	67.0
	Treatment F	12	5.92	30.7	12	5.92	30.7
...	...						

Table 1. Dummy PK summary table result: SAS vs. R

DISCUSSION AND LIMITATIONS

The primary objective of this approach is to convert the existing SAS PK summary macro into a format compatible with R. In this case study, we successfully reproduced PK summary table results with the assistance of GPT-4o. This tool automatically selects the necessary library packages, sets up R functions, and provides essential comments, thus enabling programmers with limited R coding experience to initiate their work effectively.

However, the current capabilities of GPT-4o are not sufficiently robust to seamlessly convert the entire SAS code into R. There is insufficient attention to the structural differences between the two programming languages, necessitating manual visual checks for error messages and subsequent corrections. Moreover, the functionality of the R code is inherently limited by the original design of the SAS macro, which may restrict the potential for more advanced analyses.

Additionally, this case study examines a limited number of study designs, highlighting the need for further updates and testing to ensure successful conversion across a broader range of studies. Future work should focus on investigating the GPT-4o tool's conversion capabilities, assessing its adaptability to various analytical contexts, and developing a package for RTF output.

REFERENCE

- [1] Hao Meng et. al. 2020. "R for Clinical Reporting, Yes - Let's Explore It!". PharmaSUG 2020 - Paper DV-057.
- [2] Perphy ZHAO et. al. 2023. "Application of R language in clinical data". PharmaSUG China 2023 - Paper DV-105.
- [3] Jing Su and Jiannan (Jane) Kang, 2018. Challenges and Strategies in PKPD Programming, PharmaSUG 2018 - Paper AA-13
- [4] JHuei-Ling Chen, Heng Zhou, Madhusudhan Ginnaram, Yilong Zhang, 2021. Making Customized RTF Output with R Package r2rtf, PharmaSUG 2021 – Paper EP-085

ACKNOWLEDGEMENTS

The authors would like to take this opportunity to thank all Merck colleagues, especially Dan, Jing, and Jeff who provided their valuable input for facilitating and supporting this paper.

CONTACT INFORMATION

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

Shiqi Lin
Merck & Co., Inc., Rahway, NJ, USA
e-mail: shiqi.lin@merck.com

Runcheng Li
Merck & Co., Inc., Rahway, NJ, USA
e-mail: runcheng.li@merck.com

Yiyuan Zeng
Merck & Co., Inc., Rahway, NJ, USA
e-mail: yiyuan.zeng@merck.com