PharmaSUG 2026 - Paper TT-156

SAS to R: A Practical Bridge for Programmers
Part 1 Data Manipulation and Workflow Transformation

Jyoti (Jo) Agarwal, Gilead Sciences

ABSTRACT

As the analytics landscape continues to evolve, R has emerged as a versatile and powerful tool for data analysis, visualization, and statistical modeling. While SAS remains a trusted platform in many industries, R offers distinct advantages such as flexibility, open-source accessibility, and a rich ecosystem of packages for advanced analytics. For SAS programmers, transitioning to R can present challenges due to differences in syntax, workflow, and programming approach.

Building on insights from the 2025 SI-294 paper on SAS programming efficiency, this paper provides a practical and hands-on bridge for SAS users entering the R environment. Part 1 focuses on foundational setup, basic syntax, and most importantly modern data manipulation using R and the tidyverse. It highlights parallels between SAS and R while demonstrating how familiar SAS concepts such as DATA step transformations and PROC based summaries can be translated into readable and efficient R workflows.

The paper emphasizes data wrangling using dplyr functions, the use of packages to extend R functionality, and piping for improved code readability. Through side-by-side SAS and R comparisons and real-world examples, SAS programmers gain actionable insights that allow them to build on their existing expertise while adopting R. This paper serves as a practical starting point for users seeking to expand their analytical capabilities beyond SAS.

DISCLAIMER

The views and opinions expressed in this paper are solely those of the author and do not reflect the views of the author's employer. The data and examples presented are derived from publicly available sources and do not include any proprietary or confidential data from Gilead programs.

INTRODUCTION

SAS has long been a foundation for statistical programming in clinical and business environments. Its structured workflows and reliability have made it a preferred tool, particularly in regulated settings. At the same time, the analytics landscape is evolving and there is increasing demand for tools that offer flexibility, scalability, and rapid innovation.

R has emerged as a strong complement to SAS. It is an open-source programming language designed specifically for data analysis and visualization. It offers a wide range of statistical techniques, an active global community, and the ability to produce high quality and reproducible outputs. These capabilities have contributed to its rapid adoption across research, healthcare, and data science domains.

For SAS programmers, learning R does not require starting over. Many core concepts such as working with datasets, transforming variables, and performing analysis can be translated into R with the right guidance. The key difference lies in how these tasks are expressed and executed.

The objective of this paper is to provide a practical bridge for SAS programmers transitioning to R. It focuses on:

- Familiar concepts translated into R syntax
- Hands-on examples aligned with real-world workflows
- Key differences that impact efficiency and thinking

The emphasis is on hands-on examples and real-world applications to support a smooth and confident transition.

By the end of this paper, readers will be equipped not only to use R, but to understand *when and why* to use it, enhancing their overall analytical capability.

Setting Up the R Environment

Getting Started with R

Before working with R, it is necessary to install the R programming language. R is freely available and can be downloaded from the Comprehensive R Archive Network (CRAN). Users can download the appropriate version for their operating system (Windows, macOS, or Linux) and follow standard installation procedures.

The installation process is straightforward and similar to installing any standard software application.

Choosing an Interface: Beyond Base R

While R includes a native interface, most users prefer enhanced development environments that improve usability and productivity. Several options are available:

R Native Application

- Basic interface bundled with R
- Limited usability for complex workflows

Posit Cloud (formerly RStudio Cloud)

- Browser-based environment
- No installation required
- Ideal for quick experimentation and collaboration

Jupyter / Google Colab

- Notebook-based interface
- Supports R alongside Python
- Useful for documentation and reproducible workflows

Jamovi

- GUI-based statistical tool built on R
- Allows transition from point-and-click to coding
- Can generate R syntax for analysis

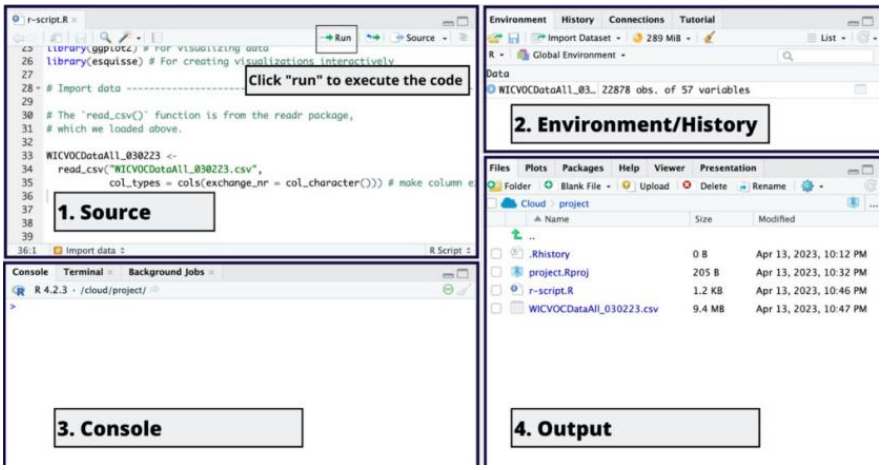
These platforms can be useful for learning and quick analysis, especially for users transitioning from GUI-driven workflows (common in SAS and SPSS) to code-driven analytics, while still maintaining accessibility.

RStudio: The Preferred Environment

RStudio, now developed by Posit, is the most widely used integrated development environment (IDE) for R and is recommended for most users. It provides a clean and organized interface that supports efficient programming and analysis.

RStudio organizes work into a structured, multi-pane interface:

- Script Editor for writing and saving code
- Console for executing commands and viewing output
- Environment and History for tracking objects and workspace activity
- Files, Plots, Packages, and Help for managing outputs and resources



This layout allows users to write code, run analysis, visualize results, and debug code within a single workspace. For SAS programmers, RStudio offers a familiar sense of structure while introducing a more interactive and flexible approach to data analysis.

KEYBOARD SHORTCUTS FOR EFFICIENCY

Efficient use of keyboard shortcuts is essential for improving productivity in RStudio. For SAS programmers who are used to structured workflows and repeated execution of code, shortcuts provide a faster and more interactive way to write, test, and debug programs.

RStudio supports a wide range of shortcuts that allow users to run code, navigate between panes, manage the environment, and format scripts without interrupting workflow. This interactive approach enables users to execute code line by line, which is particularly useful during development and troubleshooting.

Users can also customize shortcuts through the Global Options menu. This flexibility allows programmers to align their workflow with personal preferences and improve efficiency over time.

The following table summarizes commonly used shortcuts and commands.

Action	Command or Shortcut	Description
Open settings	Tools → Global Options	Configure environment and preferences
Keyboard shortcuts	Global Options → Code	Customize keyboard shortcuts
Clear console	Ctrl + L	Clears console output
Clear environment	rm(list = ls())	Removes all objects from memory
Run code	Ctrl + Enter	Executes current line or selected code
Comment code	# , ## , ###	Used to add comments and create section headers
Help	?function_name	Opens help documentation for a function or dataset
Access variable	df$variable	Accesses a variable within a dataset
Load package	library(package_name)	Loads a package into R
Load data	read_csv("file.csv")	Imports data from a csv file
Dataset help	?dataset_name	Displays dataset documentation
Save dataset	df <- dataset_name	Assigns dataset to a variable
View data	head(df)	Displays first few rows

Action	Command or Shortcut	Description
Create histogram	hist(df$variable)	Creates a histogram for a variable

These commands represent commonly used operations that form the foundation for working efficiently in R.

HELP SYSTEM AND COMMENTING CONVENTIONS

R provides a built-in help system that allows users to quickly access documentation for functions, datasets, and packages. Help can be accessed using the question mark operator followed by the function or dataset name, which displays usage, arguments, and examples directly within RStudio.

In addition to built-in help, R is supported by extensive community documentation, making it easier to explore functions, troubleshoot issues, and adopt best practices.

Commenting is essential for writing clear and maintainable code. In R, comments are created using the hash symbol, and any text following it is not executed. Comments are used to explain code, document logic, and improve readability for collaboration and review. Multiple levels of comments can also be used to organize scripts into logical sections, making code easier to navigate and understand.

Basic Operations And Syntax

Once the R environment is set up, the next step is to understand how basic operations are performed. For SAS programmers, this is often the first point where the differences in style become visible. SAS typically follows a step-based structure using DATA and PROC blocks, while R allows users to work interactively by entering and executing commands line by line.

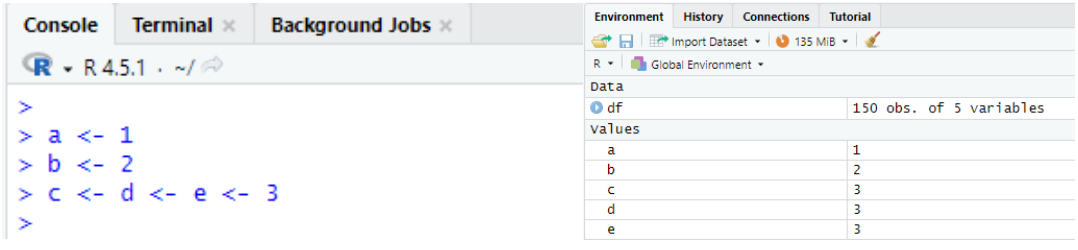
This interactive style makes R especially useful for testing logic, exploring data, and building code incrementally. Basic operations in R include variable assignment, arithmetic calculations, generation of sequences, and use of built-in functions. Although the syntax differs from SAS, the underlying concepts remain familiar and transferable.

Variable Assignment and Multiple Assignments

In R, values are assigned to variables using the assignment operator <-. This is one of the most recognizable features of R syntax. Unlike SAS, users do not need to explicitly declare variable types before assigning values. R determines the type automatically based on the assigned value.

A single value can be assigned to one variable, or the same value can be assigned to multiple variables in a single statement. This makes R concise and efficient for interactive work.

Example 1: Variable assignment in R



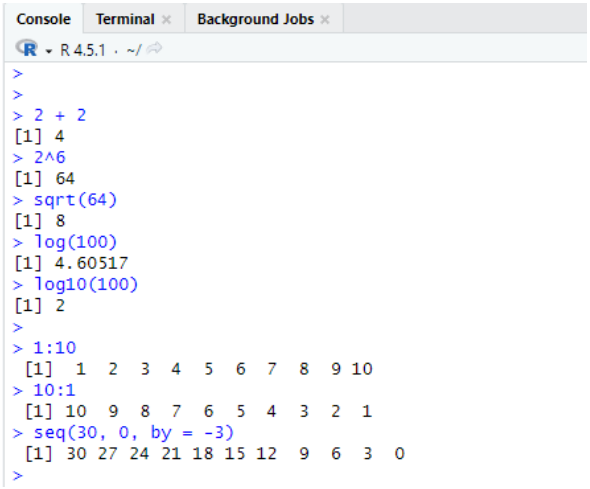
In this example, a and b are assigned individual values, while c, d, and e are all assigned the value 3. This differs from SAS, where assignments are typically performed within a DATA step.

Arithmetic and Sequences

R can perform arithmetic operations directly in the console or in a script. Addition, subtraction, multiplication, division, exponents, square roots, and logarithms are all supported through simple expressions and built-in functions.

R also provides convenient ways to generate numeric sequences. The colon operator creates consecutive values, while the seq() function allows more control over starting point, ending point, and interval size.

Example 2: Arithmetic operations and sequences in R



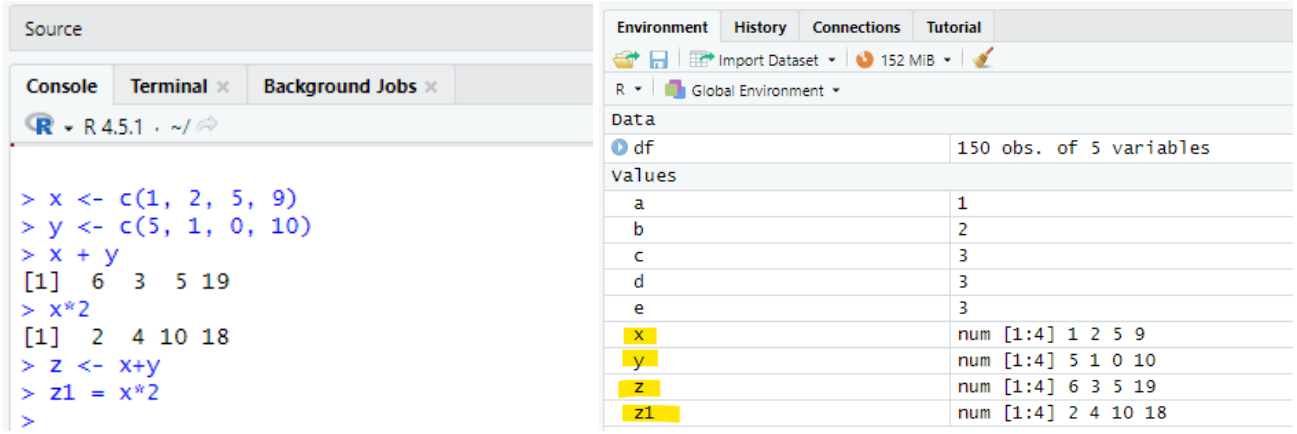
These commands illustrate both direct calculations and sequence generation. Such features are especially useful when creating test data, indexing positions, or preparing values for analysis.

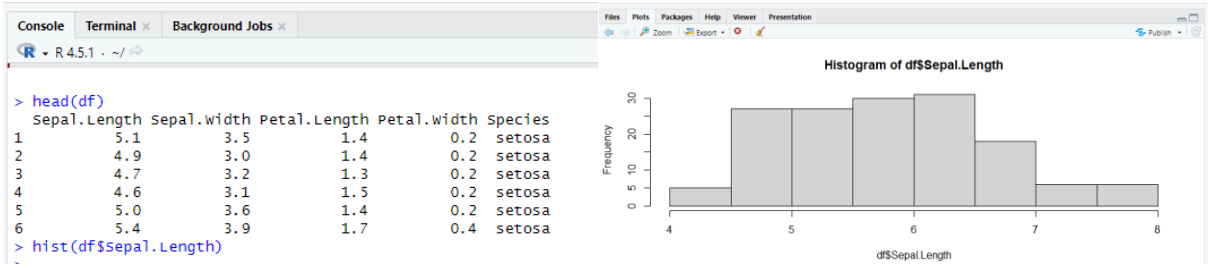
OPERATORS AND BASIC FUNCTIONS

R includes a variety of operators and built-in functions that simplify data handling and computation. For example, the dollar sign operator is commonly used to access a variable within a data frame. Functions such as head() allow quick inspection of data, while graphical functions such as hist() provide immediate visual summaries.

Vectors can be created using the c() function, which combines values into a single object. This supports element wise operations, which are central to how R works.

Example 3: Operators and basic functions in R





In this example, R adds corresponding elements of two vectors and multiplies each element of x by 2. The final two commands display the first few rows of a dataset and generate a histogram for a selected variable.

SAS And R Comparison For Basic Operations

The following table highlights a few common parallels between SAS and R for basic syntax and operations.

Task	SAS	R
Assign a value	x = 5; (within a DATA step)	x <- 5
Assign multiple variables	Separate assignment statements	a <- b <- c <- 3
Create a sequence	Use do loop	1:10 or seq(1, 10)
Square root	sqrt(x)	sqrt(x)
Logarithm	log(x)	log(x) for natural log
View first rows	proc print data=mydata(obs=5); run;	head(df)
Create histogram	proc sgplot; histogram x; run;	hist(df$x)
Access variable in dataset	dataset.variable in some contexts	df$variable

This comparison helps SAS programmers see that the transition is less about learning entirely new concepts and more about adapting to a different style of expression.

While the previous sections introduce foundational concepts, the true strength of R becomes evident in data manipulation workflows, which form the core of most analytical tasks.

Data Manipulation

Data manipulation is one of the most critical steps in any analytical workflow. For SAS programmers, this section closely aligns with DATA step processing, where data is filtered, transformed, and prepared for analysis.

In R, data manipulation is commonly performed using functions from the tidyverse, particularly the dplyr package. These functions provide a readable and efficient approach to working with data frames.

Core Data Manipulation Functions

The following functions form the foundation of data manipulation in R:

- select() for choosing columns
- filter() for subsetting rows
- mutate() for creating or modifying variables
- arrange() for sorting data
- summarise() for aggregation
- group_by() for grouped analysis

These functions allow users to perform operations in a structured and readable way.

Filtering and Selecting Data

Filtering allows users to subset rows based on conditions, similar to WHERE clauses in SAS.

```
df %>%
  filter(Age > 25) %>%
  select(Name, Age)
```

This example filters rows where Age is greater than 25 and selects specific columns.

Creating and Modifying Variables

The mutate() function is used to create new variables or modify existing ones.

```
df %>%
  mutate(Age_months = Age * 12,
         Age_group = ifelse(Age > 30, "Senior", "Junior"))
```

This is equivalent to variable derivation in SAS DATA steps.

Sorting and Arranging Data

```
df %>%
  arrange(Age)

df %>%
  arrange(desc(Age))
```

This sorts data in ascending or descending order.

Grouping and Summarizing Data

```
df %>%
  group_by(Gender) %>%
  summarise(Avg_Age = mean(Age),
            Count = n())
```

Grouping allows aggregation similar to PROC MEANS or PROC SUMMARY in SAS.

Packages and Functions

R is highly extensible through packages, which provide additional functionality for data manipulation, visualization, statistical analysis, and data import and export. Packages allow users to extend the base capabilities of R and perform complex tasks efficiently.

Packages are freely available through the CRAN, which hosts thousands of user contributed libraries across various domains.

Installing and Loading Packages

Packages are installed once and loaded in each session.

```r
install.packages("tidyverse")
library(tidyverse)
```

To simplify loading multiple packages, tools such as pacman can be used.

```r
if (!require("pacman")) install.packages("pacman")
pacman::p_load(tidyverse, rio, psych)
```

Some packages are included with R but must be explicitly loaded.

```r
library(datasets)
```

These packages enable additional functionality such as reading data, transforming variables, and performing analysis.

The Tidyverse Ecosystem

One of the most widely used collections of packages in R is the tidyverse. It provides a consistent framework for working with data and includes tools for data manipulation, visualization, and functional programming.

Key packages include:

- dplyr for data manipulation
- ggplot2 for visualization
- tidyr for data reshaping
- readr for data import
- stringr for string operations

These packages are designed to work together and promote readable and efficient workflows.

SAS and R Comparison Using DPLYR

Task	SAS	R (dplyr)
Select variables	keep var1 var2;	select(var1, var2)
Drop variables	drop var1;	select(-var1)
Filter rows	where Age > 25;	filter(Age > 25)
Multiple conditions	if Age > 25 and Gender='M';	filter(Age > 25 & Gender == "M")
Create variable	new = Age*12;	mutate(new = Age*12)
Conditional logic	if Age > 30 then Group='Senior';	mutate(Group = ifelse(Age > 30, "Senior", "Junior"))
Sort data	proc sort; by Age;	arrange(Age)
Descending sort	by descending Age;	arrange(desc(Age))
Summary stats	proc means;	summarise(mean(Age))
Grouped analysis	class Gender;	group_by(Gender)
Frequency	proc freq;	count(Gender)

This table directly bridges SAS thinking to R workflows and highlights how tidyverse functions support concise and readable data manipulation.

PIPING OPERATOR FOR READABILITY

One of the key advantages of modern R workflows is the use of piping, which allows users to chain multiple operations in a clear and sequential manner.

Without piping, nested functions can become difficult to read and interpret, especially when multiple transformations are applied to a dataset. Piping simplifies this by passing the output of one function directly into the next.

The most commonly used pipe operator in R is %>%, which is part of the tidyverse. It improves readability by structuring code in a top to bottom flow rather than inside nested expressions.

For example, a complex nested command can be rewritten using pipes to clearly show each step of the transformation.

Example 4: Without pipe (difficult to read)

```
round(prop.table(margin.table(UCBAdmissions, 3)), 2) * 100
```

Example 5: With pipe (clear and readable)

```
UCBAdmissions %>%
  margin.table(3) %>%
  prop.table() %>%
  round(2) %>%
  `*`(100)
```

Using pipes makes the workflow easier to follow, debug, and maintain. Each step is clearly separated and executed in sequence. Piping transforms code from a nested and difficult to follow structure into a logical sequence where each operation builds on the previous one.

This approach is conceptually similar to sequential DATA step processing in SAS, but provides greater flexibility and readability.

CONCLUSION

The transition from SAS to R is not a shift in analytical concepts, but a shift in approach. While SAS relies on structured DATA and PROC steps, R enables interactive, flexible, and function driven workflows.

Core tasks such as filtering, transformation, summarization, and visualization remain consistent across both environments. However, R enhances these processes through readable pipelines, extensible packages, and efficient syntax.

For SAS programmers, adopting R allows greater flexibility, faster iteration, and improved workflow efficiency. By leveraging familiar concepts in a new framework, programmers can expand their analytical capabilities while maintaining strong foundational principles.

This paper represents Part 1 of a broader practical series designed to support SAS programmers as they build confidence in R through real world applications. While this paper focuses on data manipulation and workflow transformation, subsequent papers will extend this foundation to data structures and data handling, advanced workflows and clinical programming applications, and visualization and statistical analysis.

Together, this series provides a structured and approachable pathway for expanding analytical capabilities from SAS to R.

References

Agarwal, J. 2025. “SI-294 SAS Programming Efficiency.” PharmaSUG 2025 Proceedings.
R Core Team. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation.
Wickham, H. et al. *Welcome to the tidyverse*. Journal of Open-Source Software.
Comprehensive R Archive Network. Available at CRAN.
Posit. RStudio Documentation and Tidyverse Resources. cran.r-project.org/doc/manuals/r-release/R-lang.html.
The SAS to R Cheatsheet. https://posit.co/wp-content/uploads/2023/04/sas-r.pdf

Acknowledgments

The author acknowledges the availability of publicly accessible documentation, open-source R resources, and the broader analytics community that continues to support learning and knowledge sharing across programming platforms.

Appendix A: SAS vs R for 20 Common Clinical Derivations

The examples shown are simplified for illustration purposes and may require adaptation for production use in clinical programming environments.

Clinical Derivation	SAS Code	R Code
Treatment Start Date	TRTSDT = input(TRTSDT_raw, yymmdd10.);	TRTSDT <- ymd(TRTSDT_raw)
Treatment End Date	TRTEDT = input(TRTEDT_raw, yymmdd10.);	TRTEDT <- ymd(TRTEDT_raw)
Treatment Duration	TRTDUR = TRTEDT - TRTSDT + 1;	TRTDUR <- as.numeric(TRTEDT - TRTSDT + 1)
Age	AGE = floor((RFSTDTC - BRTHDTC)/365.25);	AGE <- floor(time_length(interval(BRTHDTC, RFSTDTC), "years"))
FAS Flag	if TRTSDT ne . then FASFL='Y';	FASFL <- if_else(!is.na(TRTSDT), "Y", "N")
Safety Flag	if TRTSDT ne . then SAFFL='Y';	SAFFL <- if_else(!is.na(TRTSDT), "Y", "N")
AE Duration	AEDUR = AEENDTC - AESTDTC + 1;	AEDUR <- as.numeric(AEENDTC - AESTDTC + 1)
Max Severity	proc sql; select max(AESEV)	adae %>% group_by(USUBJID) %>% summarise(max_sev = max(AESEV_NUM))
Baseline Lab	if AVISIT='Baseline' then BASE=AVAL;	BASE <- adlbc %>% filter(AVISIT=="Baseline") %>% pull(AVAL)
Change from Baseline	CHG = AVAL - BASE;	CHG <- AVAL - BASE
Percent Change	PCHG = (AVAL - BASE)/BASE * 100;	PCHG <- (AVAL - BASE)/BASE * 100
Visit Number	VISITNUM = input(VISIT, best.);	VISITNUM <- as.numeric(VISIT)
Merge ADSL	merge adae adsl; by USUBJID;	adae <- left_join(adae, adsl, by="USUBJID")
Count AEs	proc freq data=adae; tables USUBJID;	adae %>% count(USUBJID)
Analysis Flag	if PARAMCD='ALT' then ANL01FL='Y';	ANL01FL <- if_else(PARAMCD=="ALT", "Y", "N")
Censoring Flag	if EVENT=1 then CNSR=0;	CNSR <- if_else(EVENT==1, 0, 1)
ADY	ADY = AVALDT - TRTSDT + 1;	ADY <- as.numeric(AVALDT - TRTSDT + 1)

Contact Information

Comments and questions are valued and encouraged. Contact the author at:

Jyoti (Jo) Agarwal
www.linkedin.com/in/joagarwal/