PharmaSUG 2026 - Paper ET-179

Building ADaM Datasets from Scratch using R: A SAS programmer's Perspective

Mukesh Patel, Nileshkumar Patel, Merck & Co., Inc., Rahway, NJ, USA

ABSTRACT

In clinical trials, SAS has traditionally been the primary tool for statistical programming, analysis, and reporting. However, the industry is increasingly embracing a language-agnostic approach, encouraging statistical programmers to develop proficiency in multiple programming languages. This shift is driven by the growing adoption of open-source tools, with R emerging as a powerful alternative due to its flexibility, extensive package ecosystem, and cost-effectiveness. This paper documents a SAS programmer's transition to building ADaM datasets in R.

In this paper, we will share our experience as seasoned SAS programmers developing Analysis Data Model datasets entirely from scratch using R for the first time. We will provide a detailed, step-by-step walkthrough of variable derivation in R, highlighting the similarities and differences compared to traditional SAS programming methods. The discussion includes practical challenges encountered during the transition, strategies employed to overcome these obstacles, and insights into leveraging R's unique capabilities for clinical data analysis.

Through this comparative exploration, we aim to offer valuable guidance for statistical programmers navigating the evolving landscape of clinical trial programming, emphasizing the benefits and considerations of adopting a multilingual programming approach.

INTRODUCTION

For programmers with a predominantly SAS background, transitioning to R can introduce new concepts, workflows, and learning curves. This paper presents a SAS programmer's perspective on building ADaM datasets from scratch using R, highlighting parallels and key differences between the two languages to support programmers adapting to a multilingual environment and leveraging R's capabilities in modern clinical data pipelines. While the ADaM standards remain unchanged, what does change is the syntax, the data manipulation paradigm (from DATA steps and PROC steps to tidyverse pipelines), and the toolchain—so this paper maps each familiar SAS concept to its R equivalent throughout the ADaM build process.

INITIAL SETUP: PACKAGE LOADING & LIBNAME ASSIGNMENT

Before writing a single line of ADaM code, a properly configured working environment is essential. In SAS, this is typically achieved through an initial setup program that assigns library paths (LIBNAME statements), defines macro variables, sets autocall paths, and configures system options to establish a consistent and reproducible foundation. A directly analogous approach exists in R — using startup files such as .Rprofile or project-specific initialization scripts within an R Development Environment (RDE), users can automatically configure library paths, data source locations, output directories, and logging settings at the start of each session. Both approaches serve the same purpose: ensuring all required resources and dependencies are properly referenced before analysis begins, reducing manual setup steps, and promoting reproducibility across analyses.

OVERVIEW OF THE DEMONSTRATION DATASET

ADaM BDS structure dataset ADVS is being used for demonstration purposes. ADVS (Analysis Dataset for Vital Signs) is a dataset that holds analysis-ready vital signs measurements. It is derived primarily from the SDTM VS domain and the subject-level analysis dataset (ADSL) and follows ADaM Specification.

Key structural requirements for ADVS:

- One record per subject, parameter, analysis visit, and analysis date (USUBJID + PARAMCD + AVISIT + ADT is typically unique)
- AVAL must be a numeric analysis value
- Every variable must have a label; character variables must conform to XPT length constraints
- Traceability back to the VS SDTM domain must be maintained

1. READING INPUT DATASETS

The VS domain and ADSL dataset are read. In SAS, a DATA step with a SET statement is used to read the datasets. In R, the equivalent is provided by the read_xpt() function for XPT-format files, or by read_sas() for native SAS datasets. After the data are read, it is considered good practice for the structure to be reviewed. In SAS, PROC CONTENTS and PROC PRINT are used. In R, glimpse() from the tidyverse is used for a compact column-level summary, and head() is used to preview the first few rows.

SAS Code

```
data vs;
  set sdtm.vs;
run;


data adsl;
  set adam.adsl;
run;

/* Review contents */
proc contents data=vs; run;
proc print data=vs(obs=5); run;
```

R Code

```
vs   <- read_xpt('sdtm/vs.xpt')
adsl <- read_xpt('adam/adsl.xpt')


# Review structure


glimpse(vs)
head(vs, 5)
```

2. REQUIRED IDENTIFIER VARIABLES

These variables form the backbone of every ADaM BDS dataset. They are carried from SDTM or ADSL with no transformation. For example, few of them are listed here.

Variable	Type	Required	Source	Derivation / Notes
STUDYID	Char	Required	VS.STUDYID	Study identifier; carried directly from VS
USUBJID	Char	Required	VS.USUBJID	Unique subject identifier; primary merge key
SUBJID	Char	Expected	ADSL.SUBJID	Subject identifier within study
SITEID	Char	Expected	ADSL.SITEID	Study site identifier
DOMAIN	Char	Required	Hardcoded	Always 'VS' — indicates SDTM source domain
VSSEQ	Num	Expected	VS.VSSEQ	Sequence number from SDTM VS
TRT01P	Char	Expected	ADSL	Planned treatment for Period 1

Variable	Type	Required	Source	Derivation / Notes
TRT01PN	Num	Expected	ADSL	Planned treatment numeric for Period 1
TRT01A	Char	Expected	ADSL	Actual treatment for Period 1
TRT01AN	Num	Expected	ADSL	Actual treatment numeric for Period 1

In **R**, subject-level identifiers and treatment variables are merged from ADSL into the VS-based ADaM dataset (ADVS) using left_join(by = "USUBJID"). Before the join, ADSL is trimmed with select() so that only the necessary variables—such as SUBJID, SITEID, treatment assignments, and treatment dates—are brought into ADVS. Because left_join() retains all rows from VS, the full set of VS records is preserved while adding the corresponding subject-level information from ADSL.

In **SAS**, the equivalent operation is typically performed with a MERGE … BY USUBJID; statement, often combined with IF inv; (or a similar condition) to keep only the VS records. Conceptually, this mirrors an R left_join(), where ADSL variables are appended onto the VS dataset while keeping all VS observations.

SAS Code

```
data advs;
  set adsl(in=ina keep=usubjid subjid
           siteid trt01p trt01pn
           trt01a trt01an trtsdt
           trt01sdt trt02sdt);
  by usubjid;
  if inv;
 run;
```

R Code

```
advs <- vs %>%
  left_join(
    adsl %>% select(
      USUBJID, SUBJID, SITEID,
      TRT01P, TRT01PN,
      TRT01A, TRT01AN, TRTSDT, TRT01SDT,
      TRT02SDT),
    by = 'USUBJID')
```

3. PARAMETER VARIABLES

Parameter variables identify what is being measured. PARAMCD is the short code (≤ 8 chars), PARAM is the full label, and PARAMN is an optional numeric code for sorting and by-group processing. PARCAT1 is Category for parameters and PARCAT1N optional numeric code for PARCAT1.

Variable	Type	Required	Source	Derivation / Notes
PARAMCD	Char	Required	VS.VSTESTCD	ADaM parameter code; mapped 1:1 from VSTESTCD. Max 8 chars.
PARAM	Char	Required	VS.VSTEST	Full parameter description; typically VSTEST + unit appended
PARAMN	Num	Permissible	Lookup	Numeric version of PARAMCD for sorting
PARCAT1	Char	Permissible	Protocol	Parameter category (e.g., 'VITAL SIGNS')
PARCAT1N	Num	Permissible	Protocol	Numeric version of PARCAT1 for sorting

In **R**, new parameter-related variables are created using mutate(), which allows multiple variable derivations within a single step.

- **Trimming whitespace:** stringr::str_trim() is used to remove leading and trailing spaces—similar to strip() in SAS—such as when deriving PARAMCD from VSTESTCD.
- **Conditional logic:** dplyr::if_else() supports row-wise conditional assignments, for example when constructing PARAM differently depending on whether VSSTRESU is available.
- **Concatenation:** paste0() (or paste()) is used to build display strings like "VSTEST (VSSTRESU)".
- **Multi-branch mapping:** dplyr::case_when() is used to derive variables like PARAMN based on VSTESTCD, offering a clean and readable alternative to deeply nested if statements.
- **Missing values:** R uses typed missing values; for example, TRUE ~ NA_real_ assigns a numeric missing value, while TRUE ~ NA_character_ would be used for character variables.

In **SAS,** corresponding operations are typically performed using strip() for trimming, if/then/else for conditional logic, catx() for concatenation, select/when for branching logic, and . or ' ' to represent numeric or character missing values.

SAS Code

```
/* Create a parameter lookup format */

data advs;
 set vs;
 length PARCAT1 $20 PARAMCD PARAM $200;
 PARCAT1 = "Vital Sign";
 PARCAT1N = 1;

 if not missing(VSTESTCD) then
       PARAMCD =  strip(VSTESTCD);
 else PARAMCD = "";
if not missing(VSSTRESU) then
  PARAM = catx(' ', strip(VSTEST), '('
          || strip(VSSTRESU) || ')');
else PARAM = strip(VSTEST);

select (strip(VSTESTCD));
    when ("SYSBP") PARAMN = 1;
    when ("DIABP") PARAMN = 2;
    when ("PULSE") PARAMN = 3;
    when ("TEMP") PARAMN = 4;
    when ("RESP") PARAMN = 5;
    when ("WEIGHT") PARAMN = 6;
    when ("HR") PARAMN = 7;
    when ("HEIGHT") PARAMN = 8;
    when ("BSA") PARAMN = 9;
    otherwise PARAMN = .;
   end;
run;
```

R Code

```
advs <- vs %>%
  mutate(
    PARCAT1 = "Vital Sign",
    PARCAT1N = 1,
    PARAMCD = str_trim(VSTESTCD),
    PARAM =  if_else(!is.na(VSSTRESU) &
VSSTRESU != "", paste0(str_trim(VSTEST),
" (", str_trim(VSSTRESU), ")"),
str_trim(VSTEST),

    PARAMN = case_when(
      VSTESTCD == "SYSBP" ~ 1,
      VSTESTCD == "DIABP" ~ 2,
      VSTESTCD == "PULSE" ~ 3,
      VSTESTCD == "TEMP"  ~ 4,
      VSTESTCD == "RESP"  ~ 5,
      VSTESTCD == "WEIGHT" ~ 6,
      VSTESTCD == "HR" ~ 7,
      VSTESTCD == "HEIGHT" ~ 8,
      VSTESTCD == "BSA" ~ 9,
      TRUE ~ NA_real_)

))
```

4. ANALYSIS TIMEPOINT VARIABLES

Accurate date derivation is critical for correct baseline assignment, change from baseline computation, and visit windowing. ADaMIG v1.3 requires ADT (analysis date) for all BDS datasets.

Variable	Type	Required	Source	Derivation / Notes
VISIT	Char	Permissible	VS.VISIT	Visit Name from SDTM VS
VISITNUM	Num	Permissible	VS.VISITNUM	Visit Number from SDTM VS
VSTPT	Char	Permissible	VS.VSTPT	Planned Timepoint Name from SDTM VS
VSTPTNUM	Num	Permissible	VS.VSTPTNUM	Planned Timepoint Number from SDTM VS
ADT	Num	Required	VS.VSDTC	Analysis date; numeric (days since 1960-01-01). Derived from VSDTC.
ADTM	Num	Permissible	VS.VSDTC	Analysis datetime (if time collected). Numeric datetime in SAS/XPT.
ADY	Num	Expected	ADT, TRTSDT	Analysis relative day: ADT - TRTSDT (pre); ADT - TRTSDT + 1 (post)
AVISIT	Char	Expected	VS.VISIT	Analysis Visit
AVISITN	Char	Permissible	VS.VISITNUM	Analysis Visit Number
APERIOD	Num	Expected	ADT, TR02SDT	Analysis relative day: 1) ADSL.TR02SDT is non-missing; and 2) ADT > ADSL.TR02SDT; then APERIOD = 2 Otherwise APERIOD=1.
APERDY	Char	Expected	ADT, TR01SDT, TR02SDT	Day Relative to Start of Period: For APERIOD=1: APERDY = ADT-TR01SDT+(ADT>=TR01SDT); For APERIOD=2: APERDY = ADT-TR02SDT+(ADT>=TR02SDT);

In **R**, ISO 8601 date/time character values (e.g., VSDTC) are converted into ADaM-compliant date and datetime variables using standard parsing functions.

- **Date derivation (ADT):** The date portion is extracted and converted using as.Date(), producing an R Date object appropriate for ADaM date variables.
- **Datetime derivation (ADTM):** ADTM is derived conditionally based on whether a time component is present. When nchar(VSDTC) > 10, the value is parsed into a POSIXct datetime using lubridate::ymd_hms(VSDTC); otherwise, ADTM is set to NA, preventing partial datetimes from being created when only a date is supplied.

- **Relative day (ADY) and visit-period variables (APERIOD, APERDY):** These are derived using the same mutate() patterns as other variables—if_else() for conditional assignment, as.numeric() for computing day differences, and case_when() for multi-branch logic used in period classification.

In **SAS,** equivalent processing is typically performed with input() and the appropriate date/datetime informats, often combined with substr() to separate date and time components. Subsequent derivations for day counts and period variables are implemented using standard if/then/else and select/when statements.

SAS Code

```
data advs;
  set vs;
  if not missing(VSDTC) then
  ADT = input(substr(VSDTC, 1, 10),
  yymmdd10.);
  else   ADT = .;
  if not missing(ADT) and not
  missing(TRTSDT) then
  ADY = (ADT - TRTSDT) + (ADT >=
  TRTSDT);
  else  ADY = .;
  if length(strip(VSDTC)) > 10 then do;
_vsdtc_norm = tranwrd(VSDTC, 'T', ' ');

  ADTM = input(substr(_vsdtc_norm, 1,
16), ymddttm16.);

  end;
     else   ADTM = .;
     AVISITN = VISITNUM;
     AVISIT  = strip(VISIT);
  if not missing(TR02SDT) and not
missing(ADT) and ADT > TR02SDT then
        APERIOD = 2;
else
        APERIOD = 1;
    select (APERIOD);
      when (1) do;
            if not missing(ADT) and not
missing(TR01SDT) then  APERDY = (ADT -
TR01SDT) + (ADT >= TR01SDT);
            else  APERDY = .;
        end;
        when (2) do;
         if not missing(ADT) and not
missing(TR02SDT) then APERDY = (ADT -
TR02SDT) + (ADT >= TR02SDT);
            else  APERDY = .;
        end;
        otherwise APERDY = .;
    end;
    format ADT  yymmdd10.  ADTM
datetime16.;
run;
```

R Code

```
  advs <- vs %>%

  mutate(

    ADT = as.Date(substr(VSDTC, 1, 10),
format = "%Y-%m-%d"),

    ADY = if_else(!is.na(ADT) &
!is.na(TRTSDT), as.numeric(ADT -
TRTSDT)) + as.numeric(ADT >= TRTSDT),

                  NA_real_),

    ADTM = if_else(nchar(VSDTC) > 10,
ymd_hms(VSDTC), NA_POSIXct_),

    AVISITN = VISITNUM,

    AVISIT  = str_trim(VISIT),

    APERIOD = if_else(!is.na(TR02SDT) &
(ADT > TR02SDT),2, 1),

    APERDY  = case_when(

      APERIOD == 1 ~ as.numeric(ADT -
TR01SDT) + as.numeric(ADT >= TR01SDT),

      APERIOD == 2 ~ as.numeric(ADT -
TR02SDT) + as.numeric(ADT >= TR02SDT),

      TRUE ~ NA_real_)

    )
```

5. ANALYSIS VALUE, BASELINE, CHANGE FROM BASELINE VARIABLES

In ADaM, **AVAL** provides the standardized analysis-ready measurement, **BASE** captures the subject's pre-treatment reference value, and **CHG/PCHG** quantify how the value changes from baseline over time. These variables ensure consistent, traceable inputs for summaries and statistical models.

Variable	Type	Required	Source	Derivation / Notes
AVAL	Num	Required	VS.VSSTRESN	Analysis Value; primary analysis variable
AVALU	Char	Expected	VS.VSSTRESU	Analysis Value Units (e.g., 'mmHg', 'kg', 'C')
ABLFL	Char	Expected	Derived	Baseline record flag: Set to Y if it is the last non-missing record (according to latest date and largest VSSEQ) prior to the first study medication day for each subject, each parameter of each period.
BASE	Num	Expected	Derived	Baseline value of AVAL (merged from ABLFL='Y' record)
BASETYPE	Char	Permissible	Protocol	Concatenate "Period" with character value of APERIOD variable for applicable records.
CHG	Num	Expected	AVAL − BASE	Change from baseline; missing if AVAL or BASE is missing
PCHG	Num	Expected	(CHG / BASE) × 100	Percent change from baseline; missing if BASE = 0 or missing

In **R**, baseline derivation is expressed as a clear and reproducible pipeline:

1. **Filter to baseline-eligible records**
   Begin with the VS dataset and keep only observations with non-missing ADT, limited to records occurring before treatment start (e.g., ADT < TRT01SDT when baseline is defined as the last pre-dose value). The exact rule follows the study's baseline definition.
2. **Order within subject and parameter**
   Sort the filtered data by STUDYID, USUBJID, PARCAT1N, and PARAMN to establish the groups within which baseline will be selected.
3. **Select the baseline record (last eligible observation)**
   For each subject/parameter group, use slice_max() to choose the record with the latest ADT.
   If multiple records share the same date, apply a second slice_max() on VSSEQ to break ties.
   This produces a single baseline record per subject per parameter.
4. **Create the baseline lookup dataset**
   In the resulting baseline subset, rename AVAL to BASE and VSSEQ to NVSSEQ.
   Retaining the original sequence number enables accurate baseline-flagging later.
5. **Merge baseline information back into the analysis dataset**
   Join the baseline dataset back to the full VS-derived analysis dataset using left_join().
   Then derive ABLFL by flagging the row where VSSEQ == NVSSEQ—the selected baseline record.
6. **Finalize baseline-related variables**
   After dropping the helper variable NVSSEQ, derive:
   - **CHG** and **PCHG** using mutate() and if_else() to properly handle missing values and protect against divide-by-zero.

- **BASETYPE** using paste0() and str_trim() to generate a clear label identifying the baseline definition used.

In **SAS,** equivalent workflow typically relies on where filters, PROC SORT with appropriate BY groups, and first./last. logic to select the correct record (often using date and sequence ordering). The selected baseline values are then merged back into the main dataset, followed by the derivation of ABLFL, CHG, and PCHG within a DATA step.

SAS Code	R Code

`data base1;`
` set vs;`
`  where not missing(ADT) and ADT <`
`  TR01SDT;`
`run;`
 	
`proc sort data=base1;`
`  by STUDYID USUBJID PARCAT1N PARAMN`
`  descending ADTM descending ADT`
`  descending VSSEQ;`
`run;`
 	
`data base2;`
`  set base1;`
`  by STUDYID USUBJID PARCAT1N PARAMN;`
`  if first.PARAMN;`
`  rename AVAL  = BASE`
`  VSSEQ = NVSSEQ;`
`  keep STUDYID USUBJID PARCAT1N PARAMN`
`  AVAL VSSEQ;`
`run;`
 	
`proc sort data=vs;`
`  by STUDYID USUBJID PARCAT1N PARAMN;`
`run;`
 	
`proc sort data=base2;`
`  by STUDYID USUBJID PARCAT1N PARAMN;`
`run;`
 	
`data vs1;`
`  merge vs(in=a) base2(in=b);`
`  by STUDYID USUBJID PARCAT1N PARAMN;`
`  if a;`
`  if VSSEQ = NVSSEQ then ABLFL = "Y";`
`  else ABLFL = "";`
`  drop NVSSEQ;`
`run;`
 	
`data vs2;`
`  set vs1;`
`  if APERDY > 1 and not missing(BASE)`
`  then`
`  CHG = AVAL - BASE;`
`  else`
`  CHG = .;`
 	
`  if APERDY > 1 and BASE > 0 then`
`  PCHG = (CHG / BASE) * 100;`
	`base1 <- vs %>%`
	
	`  filter((!is.na(ADT) & ADT < TR01SDT))`
	
	`base2 <- base1 %>%`
	
	`  group_by(STUDYID, USUBJID, PARCAT1N,`
	`PARAMN) %>%`
	
	`  slice_max(order_by = ADTM, n = 1) %>%`
	
	`  slice_max(order_by = ADT, n = 1) %>%`
	
	`  slice_max(order_by = VSSEQ, n = 1) %>%`
	`# to get max VSSEQ if still multiple obs`
	`present #`
	
	`  ungroup() %>%`
	
	`  rename(BASE = AVAL, NVSSEQ = VSSEQ)`
	`%>%`
	
	`  select(STUDYID, USUBJID, PARCAT1N,`
	`PARAMN, BASE, NVSSEQ)`
	
	`vs1 <- vs %>%`
	
	`  left_join(base2, by = c("STUDYID",`
	`"USUBJID", "PARCAT1N", "PARAMN")) %>%`
	
	`  mutate(`
	
	`  ABLFL = if_else(  VSSEQ == NVSSEQ,`
	`"Y", NA_character_)`
	
	`  ) %>% select(-NVSSEQ)`
	
	`vs2 <- vs1 %>%`
	
	`  mutate(`
	
	`  CHG = if_else(APERDY > 1 &`
	`!is.na(BASE), AVAL - BASE, NA_real_),`
	
	`  PCHG = if_else(APERDY > 1 & BASE >`
	`0, (CHG / BASE) * 100, NA_real_),`
	
	`  BASETYPE = paste0( "Period ",`

SAS Code	R Code

else
PCHG = .;
BASETYPE = cats("Period ",
strip(put(APERIOD, best.)));
 	
run;
	str_trim(APERIOD))
	
	)

6. ANALYSIS FLAG: ANL01FL

Analysis flags are used to identify the primary set of records to be included in a specific analysis (e.g., the main endpoint/time window/selection rule). It ensures tables and models use the same, pre-defined subset of data consistently (e.g., one record per subject per visit/parameter after applying selection criteria).

Variable	Type	Required	Derivation / Notes
ANL01FL	Char	Expected	"Y" for the last record of each USUBJID, PARAMCD, AVISITN. For multiple records with same date, consider the one with the largest VSSEQ.

In **R**, ANL01FL is derived by creating a helper dataset that identifies the single analysis record within each grouping, based on a defined ordering rule.

1. **Create a flagging dataset**
   Start by selecting only the variables required for flagging—typically the grouping variables along with ADT and VSSEQ.
2. **Sort to bring the "best" record to the top**
   Arrange the data so the desired record appears first within each group. This is usually done by sorting by descending ADT and then descending VSSEQ, ensuring that the most recent assessment—and, in the case of ties, the record with the highest sequence number—is prioritized.
3. **Derive ANL01FL based on row position**
   Within each grouping, use row_number() to identify the first record after sorting. Then assign the flag using mutate() and if_else(), setting ANL01FL = "Y" when row_number() == 1, and leaving it missing otherwise.
4. **Merge the flag back into the full analysis dataset**
   Use left_join() to merge the flagging dataset back onto the original analysis dataset so that ANL01FL appears alongside all other analysis variables.

In **SAS,** a similar approach is typically implemented by sorting with PROC SORT, using BY-group FIRST. logic to identify the first record, and then merging the resulting flag back into the main dataset in a subsequent DATA step.

SAS Code	R Code
proc sort data=vs out=fl1;	fl1 <- vs %>%

SAS Code	R Code

by USUBJID PARAMCD AVISITN
descending ADT descending VSSEQ;
run;
 	
data fl2;
set fl1;
by USUBJID PARAMCD AVISITN;
if first.AVISITN then ANL01FL = "Y";
else ANL01FL = "";
keep USUBJID PARAMCD AVISITN ADT
VSSEQ ANL01FL;
run;
 	
proc sort data=vs;
by USUBJID PARAMCD AVISITN ADT
VSSEQ;
run;
 	
proc sort data=fl2;
by USUBJID PARAMCD AVISITN ADT
VSSEQ;
run;
 	
data vs;
merge vs (in=a) fl2 (in=b);
by USUBJID PARAMCD AVISITN ADT
VSSEQ;
if a;
run;
	select(USUBJID, PARAMCD, AVISITN, ADT,
	VSSEQ, AVAL) %>%
	arrange(USUBJID, PARAMCD, AVISITN,
	desc(ADT), desc(VSSEQ))
	
	fl2 <- fl1 %>%
	group_by(USUBJID, PARAMCD, AVISITN)
	%>%
	mutate(ANL01FL = if_else(row_number()
	== 1, "Y", NA_character_)) %>%
	ungroup()
	
	vs <- vs %>%
	left_join(select(fl2, USUBJID,
	PARAMCD, AVISITN, ADT, VSSEQ, ANL01FL),
	by = c("USUBJID", "PARAMCD",
	"AVISITN", "ADT", "VSSEQ"))

7. APPLYING METADATA, VARIABLE ORDER, AND XPT EXPORT

In **R**, once all variables specified in the ADaM metadata have been derived, the dataset is finalized by applying our organization's in-built standard function to the completed analysis dataframe. This step standardizes the final deliverable by:

- **Applying variable attributes** such as labels, lengths/types, and formats where applicable.
- **Enforcing the required variable order** defined in the ADaM specification.
- **Producing transport-ready output**, including creation of the XPT file as part of the finalization workflow.

In **SAS,** a similar process in SAS is performed at the end of dataset creation using ATTRIB, FORMAT, and LABEL statements, applying the required variable order (e.g., via RETAIN), and exporting to XPT using SAS transport procedures.

As an alternative approach in R, XPT creation can also be performed using the xportr package, which supports metadata-driven application of CDISC-compliant attributes and writes XPT files.

SAS Code	R Code
`%add0attribute (input_dataset_specdata = advs0spec ,Input_dataset = vs ,domain = advs ,output_dataset = advs);`	`ADVS <- gsu0add0attribute(path, "ADVS", vs, spec_ls, out_path=file.path(proj, "rout"))`

CHALLENGES & STRATEGIES

The following are key challenges encountered during the transition from SAS to R for ADaM dataset development, along with practical strategies employed to overcome each one.

CHALLENGE 1: SHIFT FROM PROCEDURAL TO FUNCTIONAL PROGRAMMING MINDSET

SAS programmers are accustomed to the DATA step's row-by-row, sequential processing model — reading one observation at a time, retaining variables across iterations, and using FIRST. / LAST. logic for by-group processing. R's tidyverse paradigm operates on entire datasets at once through vectorized, declarative pipelines, which feels counterintuitive at first.

STRATEGY

Systematically map SAS constructs to their R equivalents before writing a single line of code. The dplyr verb family — mutate(), filter(), group_by(), summarize(), and arrange() — is the closest analog to the SAS DATA step and PROC SQL combined. Building a personal cheat sheet accelerates this transition significantly.

CHALLENGE 2: REPLICATING SAS MACRO FUNCTIONALITY

Experienced SAS programmers rely heavily on macro variables (%let, %macro, %do %while) to parameterize and reuse derivation logic across multiple ADaM programs. R has no macro preprocessor, and this gap can feel limiting when first attempting to build generalizable code.

STRATEGY

Embrace R functions as the direct — and in fact more powerful — replacement for SAS macros. Unlike SAS macros which perform text substitution, R functions are true first-class objects that accept typed inputs, return values, and can be tested independently. For ADaM-specific patterns, some R packages (admiral) provide a library of pre-built, CDISC-compliant derivation functions

CHALLENGE 3: DATE AND TIME ARITHMETIC

SAS represents dates as integers (days since January 1, 1960), making study day calculations and interval derivations straightforward with INTCK and INTNX. R's date ecosystem is richer but more fragmented — Date, POSIXct, and POSIXlt classes coexist, and origin differences can silently introduce errors in ADSL date variables and BDS parameters like ADY, ASTDY, and AENDY.

STRATEGY

Use as.Date() consistently and avoid mixing date classes within a dataset. For study day derivation, implement a single reusable function and apply it uniformly across all ADaM programs. All date operations can be standardized using the lubridate package.

CHALLENGE 4: VARIABLE ORDERING, ATTRIBUTES, AND XPT COMPLIANCE

CDISC ADaM submissions require datasets in SAS Transport (.xpt) format with strict rules: variable names limited to 8 characters, labels up to 200 characters, specific variable ordering, and defined character lengths. In SAS, these attributes are managed naturally through dataset metadata. In R, data frames carry no inherent metadata unless explicitly assigned.

STRATEGY

An internally developed company package has been used to address this challenge. As additional options, the **haven** package may be employed for importing existing SAS datasets—where format attributes are preserved—and haven::write_xpt() may be used for exporting XPT files. The **xportr** package can also be adopted, as it is designed specifically to support pre-submission compliance by enforcing variable order, type, length, label, and format based on a metadata specification. It is recommended that **xportr** checks be integrated into the final step of every ADaM program, where they should serve as a required quality gate prior to dataset lock.

CHALLENGE 5: REPLICATING BY-GROUP PROCESSING

One of the most frequently used SAS techniques in ADaM programming — BY-group processing with FIRST./LAST. flags — have no direct syntactic equivalent in R and require a different way of thinking about data transformation.

STRATEGY

The table below maps the most common SAS patterns to their R equivalents. Practice translating five to ten ADaM DATA step patterns before beginning the full dataset build:

SAS Pattern	R Equivalent
IF FIRST.USUBJID	group_by(USUBJID) %>% slice(1)
IF LAST.USUBJID	group_by(USUBJID) %>% slice_tail(n=1)

CHALLENGE 6: LEARNING CURVE AND PRODUCTIVITY LOSS

For a seasoned SAS programmer, the initial transition to R involves a real and unavoidable productivity dip. Syntax that takes seconds in SAS may require research and iteration in R, and debugging error messages in R can be less intuitive than SAS's log-based feedback system.

STRATEGY

Invest in structured upskilling before production work begins. The following resources are specifically tailored for clinical programmers making this transition:

- General training materials and domain-specific ADaM walkthroughs
- Pair programming — where an experienced R user works alongside the transitioning programmer on the first one or two ADaM datasets

TIPS TO LEVERAGE R'S CAPABILITIES

This section provides a practical SAS-to-R translation reference and a curated R package ecosystem to support ADaM dataset development. These tools enable efficient, CDISC-compliant, and regulatory-ready clinical data programming in R.

1. SAS TO R TRANSLATION REFERENCE

The following table provides a practical reference mapping frequently used SAS constructs to their R equivalents for ADaM development:

SAS Construct	SAS Description	R Equivalent	Notes
DATA step	Row-by-row transformation logic	**dplyr::mutate() pipeline**	Vectorized column derivation
WHERE Statement	Subset records	**dplyr::filter()**	dplyr filtering function
PROC SQL (joins/merge)	Dataset joins by key variables	**dplyr::left_join() / full_join()**	Full tidyverse join family
PROC SORT + BY group	Sort and group processing	**arrange() + group_by()**	dplyr verbs, chainable
PROC SORT DESC + FIRST. Variable	Group tiebreaking	**group_by() + slice_max()**	dplyr package functions
PROC TRANSPOSE	Wide ↔ long reshape	**tidyr::pivot_wider/longer()**	Controlled by names_from / values_from
KEEP statement	Keep variables	**select()**	dplyr package function
LEFT JOIN	Join the records	**left_join()**	dplyr package function
IF/ELSE data step	Conditional Logic	**if_else() / mutate()**	dplyr package functions
STRIP() Function	Removal of Leading and Trailing Blanks	**str_trim()**	dplyr package function
Cats() function	Concatenating strings	**paste0()**	dplyr package function
PUT(..., best.) numeric format	Numeric Format	**as.numeric()**	dplyr package function
%macro / %let	Reusable parameterized code blocks	**R functions + admiral package**	derive_*() pattern
PROC EXPORT (xpt)	SAS Transport for FDA submission	**haven::write_xpt()**	+ xportr package for validation
Missing value (.)	Sorts lowest,	**NA (propagates in**	Use na.rm / is.na()

SAS Construct	SAS Description	R Equivalent	Notes
	numeric	**operations)**	guards
SAS formats (DATE9.)	Variable display attributes	**haven + attr(var,'format.sas')**	Apply at output/export time
PROC MEANS/SUMMARY	Descriptive statistics	**dplyr::summarize()**	Group-wise summary
RENAME statement	Rename variable logic	**rename() function**	dplyr function
RETAIN statement	Carry-forward variable logic	**lag() or cumulative functions**	tidyverse window functions

2. RECOMMENDED R ECOSYSTEM FOR ADAM

A carefully selected set of R packages forms the foundation of a productive ADaM development environment:

Package	Purpose
admiral	CDISC-compliant ADaM derivation functions (derive_*, filter_*, compute_*)
dplyr	Core data manipulation: mutate, filter, select, join, group_by, summarize
tidyr	Reshaping: pivot_wider(), pivot_longer()
lubridate	Date arithmetic: study day, duration, interval handling
haven	Import/export SAS datasets and XPT files
xportr	Validate and write regulatory-compliant XPT files
metatools	Apply metadata to ADaM datasets
testthat	Unit testing framework for derivation functions
diffdf / arsenal	Dataset comparison for QC and parallel validation

CONCLUSION

Moving from SAS to R for ADaM dataset development involves a significant but achievable learning curve for seasoned clinical programmers. Although SAS's procedural, row-oriented style contrasts with R's vectorized operations and pipeline-based workflows, the underlying ADaM standards and derivation principles remain the same. By deliberately translating common SAS patterns into tidyverse equivalents—and by using specialized tools such as admiral, xportr, and lubridate—programmers can produce compliant, submission-ready ADaM datasets in R with confidence. Common hurdles like date calculations, metadata handling, by-group logic, and XPT export requirements all have established R solutions that, once mastered, can match or even surpass the productivity of traditional SAS approaches. In the end, building capability across multiple languages helps statistical programmers succeed in a more open-source clinical trial ecosystem, expanding both personal skill sets and organizational adaptability as regulatory expectations continue to evolve.

ACKNOWLEDGEMENT

The authors would like to thank our management team Jeffrey Lavenberg, Sapan Patel, Amy Gillespie and Ruchitbhai Patel for their time in reviewing the paper and providing valuable comments.

REFERENCES

1. CDISC. Analysis Data Model Implementation Guide (ADaMIG) Version 1.3. CDISC, 2021.
2. R Core Team. R: A Language and Environment for Statistical Computing. R Foundation, 2024.
3. R for Data Science by Hadley Wickhan & Garrett Grolemund

CONTACT INFORMATION

Name: Mukesh Patel
Enterprise: Merck & Co., Inc.
Address: 126 E. Lincoln Ave
City, State & Zip: Rahway, NJ 07065-4607
Phone: (732) 594-3057
E-mail: mukesh.patel1@merck.com
Web: www.merck.com

Name: Nilesh Patel
Enterprise: Merck & Co., Inc.
Address: 126 E. Lincoln Ave
City, State & Zip: Rahway, NJ 07065-4607
Phone: (732) 594-6312
E-mail: nileshkumar.patel@merck.com
Web: www.merck.com