PharmaSUG 2025 - Paper OS-167

Advanced Programming with R: Leveraging Tidyverse and admiral for ADaM Data Set Creation with a Comparison to SAS®

Joshua J. Cook, University of West Florida;
Richann Jean Watson, DataRich Consulting

ABSTRACT

The transition from SDTM to ADaM data sets is a critical step in clinical trial data preparation, requiring precise programming solutions to ensure compliance with CDISC standards. This paper presents a two-stage approach to ADSL derivation: first, we manually derive key variables using base R and the tidyverse, demonstrating the detailed logic required for compliance and flexibility in handling real-world data challenges. Then, we explore the admiral package, showcasing how its specialized functions significantly simplify and streamline this process. By leveraging admiral, we automate derivations such as treatment start and end dates (TRTSDT, TRTEDT) and treatment duration (TRTDUR) while addressing duplicates, validating data set integrity, and enhancing reproducibility. We highlight the power of admiral in minimizing code redundancy, integrating custom logic, and ensuring seamless compliance with industry standards. To aid in the transition of code from SAS® to R, this paper includes a SAS version of the code for comparison purposes. Designed for intermediate to advanced programmers, this paper illustrates the benefits of transitioning to open-source tools and the pharmaverse ecosystem for robust clinical data workflows.

INTRODUCTION

The creation of Analysis Data Model (ADaM) data sets from Study Data Tabulation Model (SDTM) data sets is an essential process in clinical trials, enabling efficient statistical analysis and regulatory submission readiness. Precise and compliant programming is critical during this transformation to uphold Clinical Data Interchange Standards Consortium (CDISC) standards. Traditionally, SAS has dominated this landscape; however, the growing adoption of open-source tools presents new opportunities for streamlined, flexible, and reproducible clinical data workflows.

This paper explores an advanced two-stage approach for deriving key Subject-Level Analysis Dataset (ADSL) variables using the R programming language, with direct comparisons to SAS. Initially, we illustrate the manual derivation process leveraging base R and the tidyverse, providing insights into the detailed logic, flexibility, and challenges encountered when handling real-world clinical trial data. Subsequently, we demonstrate the automation, and efficiency gains achievable through the specialized R package, admiral, highlighting its ability to minimize redundancy, enhance reproducibility, and simplify complex data transformations.

By directly comparing these R-based methods with equivalent techniques implemented in SAS, we offer practical insights for clinical programmers and data scientists contemplating a transition to open-source tools. This comparison emphasizes both the technical feasibility and operational advantages of adopting R and the pharmaverse ecosystem in clinical trial data programming. Example data was obtained from CDISC. All code, data, and output can be found in our GitHub repository for this project.

SETUP

In R, we initiate setup by defining and installing required packages, such as tidyverse, safetyData, admiral, readxl, gt, haven, and waldo, ensuring that any missing packages are automatically installed. The concatenation function c() is used to make a group of packages that should be installed if not

already. Libraries are then explicitly loaded into the environment to facilitate subsequent data operations and analyses.

```r
# Clear workspace
rm(list = ls())

# Define required packages and install any that are missing
required_packages <- c("tidyverse", "safetyData", "admiral", "readxl", "gt", "haven",
"waldo")
install_if_missing <- function(packages) {
  installed_packages <- rownames(installed.packages())
  missing_packages <- packages[!packages %in% installed_packages]
  if (length(missing_packages) > 0) {
    install.packages(missing_packages, dependencies = TRUE)
  }
}
install_if_missing(required_packages)

# Load libraries
library(tidyverse)
library(safetyData)
library(admiral)
library(readxl)
library(gt)
library(haven)
library(waldo)
```

It is common to be presented with a warning about function conflicts when installing multiple libraries. However, the most recently loaded package will be used as the default function when called, and the alternative function can be called by explicitly specifying the package name such as “dplyr::lag() or stats::lag().”

```
✖ dplyr::lag()    masks stats::lag()
```

On the other hand, in SAS, setup begins with defining libraries and paths to data directories using LIBNAME statements, establishing clear references to input and output data sets**.** Additionally, global macro variables are defined to set study identifiers and paths to key directories, streamlining the workflow and maintaining consistency throughout the codebase.

```sas
/* clear out work directory */
proc datasets lib = work nolist memtype = data kill force;
quit;

options validvarname = v7;
libname sdtm 'C:\Users\gonza\OneDrive -
datarichconsulting.com\Desktop\GitHub\ADaM_R_vs_SAS-ADSL\Data\SDTM';
libname adam 'C:\Users\gonza\OneDrive -
datarichconsulting.com\Desktop\GitHub\ADaM_R_vs_SAS-ADSL\Data\ADaM';
libname cadam 'C:\Users\gonza\OneDrive -
datarichconsulting.com\Desktop\GitHub\ADaM_R_vs_SAS-ADSL\Data\ADaM\CDISC' access =
readonly;
```

SHELL CREATION

In R, shell creation involves reading data set specifications from external Excel files using read_xlsx() and mapping variable types appropriately. A data set shell is then generated using tidyverse functions such as tibble() and mutate(), explicitly assigning variable types like character, integer, numeric, or date. Labels from the specifications file are applied to each variable.

```r
# Load ADSL specifications and map types to R types
specs_path <- "data/adam_define.xlsx"
adsl_specs <- read_xlsx(specs_path, sheet = "ADSL")

type_mapping <- c(
  "text" = "character",
  "integer" = "integer",
  "float" = "double",
  "datetime" = "Date"
)
adsl_specs <- adsl_specs %>%
  mutate(Mapped_Type = type_mapping[Type] %>% replace_na("character"))

# Create an empty dataset (shell) with the specified variables and labels
adsl_shell <- tibble()
for (i in seq_len(nrow(adsl_specs))) {
  col_name <- adsl_specs$Variable[i]
  col_type <- adsl_specs$Type[i]
  adsl_shell[[col_name]] <- switch(
    col_type,
    "text" = character(),
    "integer" = integer(),
    "float" = numeric(),
    "datetime" = as.Date(character()),
    stop("Unsupported type")
  )
  attr(adsl_shell[[col_name]], "label") <- adsl_specs$Label[i]
}
str(adsl_shell)
```

Conversely, in SAS, shell creation typically involves creating an empty data set structure through the TRANSPOSE procedure (**PROC TRANSPOSE**) or DATA steps. Variables are explicitly defined using ATTRIB statements, specifying variable types (character or numeric), lengths, and labels. This structured approach in SAS facilitates alignment with predefined metadata specifications and ensures consistency in data set creation. The code illustrates two techniques and both yield the same results. It is matter of preference which is used. The author is partial to **CALL EXECUTE** however if the user is not familiar with the **CALL EXECUTE** subroutine, the first option would be ideal and some companies may have a standard process or macro that will build a data set shell.

SAS

```
libname specs 'C:\Users\gonza\OneDrive -
datarichconsulting.com\Desktop\GitHub\ADaM_R_vs_SAS-ADSL\adam_define.xlsx';

data adslspec;
   length allvars $2000;
   set specs.'ADSL$'n end = eof;
   retain allvars;
   length dc $200;
   call missing(dc, dn);
   allvars = catx(' ', allvars, variable);
   if eof then call symputx('allvars', allvars);
run;

libname specs clear;

/* OPTION 1 - CREATING SHELL DATA SET */
/* create all numeric variables */
proc transpose data = adslspec out = tspec_n;
   var dn;
   id variable;
   idlabel label;
   where lowcase(type) in ('integer' 'float');
run;

/* create all character variables */
proc transpose data = adslspec out = tspec_c;
   var dc;
   id variable;
   idlabel label;
   where lowcase(type) not in ('integer' 'float');
run;

/* create master shell */
data adslskel;
   retain &allvars;
   set tspec_: (drop = _:);
run;

/* OPTION 2 - CREATING SHELL DATA SET */
data _null_;
   set adslspec end = eof;
   if _n_ = 1 then do;
      call execute ('data adslskel;');
      call execute ('   attrib');
   end;
   if lowcase(type) not in ('integer' 'float') then __length = cats('$', length, '.');
   else __length = cats(length, '.');
   if not missing(DISPLAY_FORMAT) then __format = catx(' ', 'format =', DISPLAY_FORMAT);
   attrbstmt = catx(' ', VARIABLE, 'label = ', quote(strip(LABEL)),
                    'length = ', __length, __format);
   call execute (attrbstmt);
   if eof then do;
      call execute('; call missing(of _all_); stop; run;');
   end;
run;
```

DATA RETRIEVAL

Data retrieval is a critical step involving reading, filtering, and transforming source data into structured formats suitable for subsequent analysis. In our example, data retrieval was completed by first defining the file paths for each data set, and then combining the data (such as questionnaires) into a single data set. Both R and SAS approaches leverage domain-specific data sets and apply logical rules to extract meaningful subsets of data.

R

```
# Define file paths for QS (questionnaire) xpt files and load them into a list
qs_files <- list(
  qsco = "data/qsco.xpt",
  qsda = "data/qsda.xpt",
  qsgi = "data/qsgi.xpt",
  qshi = "data/qshi.xpt",
  qsmm = "data/qsmm.xpt",
  qsni = "data/qsni.xpt"
)
qs_list <- lapply(qs_files, read_xpt)

# Combine all QS files into one data set and write out as an xpt file
qs_all <- bind_rows(qs_list)
write_xpt(qs_all, "data/qs_all.xpt")

# Define file paths for SDTM domains (with questionnaire variables removed)
sdtm_paths <- list(
  dm = "data/dm.xpt",  # Demographics
  ae = "data/ae.xpt",  # Adverse Events
  se = "data/se.xpt",  # Subject Elements
  ex = "data/ex.xpt",  # Exposure
  vs = "data/vs.xpt",  # Vital Signs
  sc = "data/sc.xpt",  # Subject Characteristics
  mh = "data/mh.xpt",  # Medical History
  ds = "data/ds.xpt",  # Disposition
  sv = "data/sv.xpt",  # Subject Visits
  qs = "data/qs_all.xpt"  # Combined Questionnaire Data
)
sdtm_data <- lapply(sdtm_paths, haven::read_xpt)
names(sdtm_data) <- names(sdtm_paths)
str(sdtm_data)  # Check structure
```

DEMOGRAPHIC DATA

In R, data was first read in from xpt files and loaded into lists. Demographic data retrieval involves loading the DM data set using functions like read_xpt(), renaming variables (rename()), and filtering subjects (filter()). Key demographic variables such as age groups, race, ethnicity, and treatment assignments are derived using mutate() and conditional logic functions like case_when().

R

```
src_dm <- sdtm_data$dm %>%
  rename(`__origageu` = AGEU,
         `__origrace` = RACE,
         `__origethnic` = ETHNIC) %>%
  filter(ARMCD != "Scrnfail") %>%
  mutate(
    AGEU   = str_trim(`__origageu`),
    RACE   = str_trim(`__origrace`),
    ETHNIC = str_trim(`__origethnic`),
    SITEGR1 = if_else(as.numeric(SITEID) %in% c(702, 706, 707, 711, 714, 715, 717),
"900", SITEID),
    TRT01P = ARM,
    TRT01PN = case_when(
      str_starts(ARM, "Placebo") ~ 0,
      str_detect(ARM, regex("Low", ignore_case = TRUE)) ~ 54,
      str_detect(ARM, regex("High", ignore_case = TRUE)) ~ 81,
      TRUE ~ NA_integer_
    ),
    AGEGR1 = case_when(
      AGE > 0 & AGE < 65 ~ "<65",
      AGE >= 65 & AGE <= 80 ~ "65-80",
      AGE > 80 ~ ">80",
      TRUE ~ NA_character_
    ),
    AGEGR1N = case_when(
      AGEGR1 == "<65" ~ 1,
      AGEGR1 == "65-80" ~ 2,
      AGEGR1 == ">80" ~ 3,
      TRUE ~ NA_integer_
    ),
    RACEN = case_when(
      !is.na(RACE) & str_sub(RACE, 1, 2) == "WH" ~ 1,
      !is.na(RACE) & str_sub(RACE, 1, 2) == "BL" ~ 2,
      !is.na(RACE) ~ 6,
      TRUE ~ NA_integer_
    ),
    ITTFL = if_else(!is.na(ARMCD), "Y", "N"),
    RFENDT = if_else(!is.na(RFENDTC), ymd(RFENDTC), as.Date(NA))
  )
```

In SAS, demographic data retrieval is performed through **DATA** steps with the WHERE statement to filter records and conditional statements (**IF-THEN-ELSE**) to derive demographic variables. **Various SAS techniques** may also be utilized to categorize and format demographic information.

```sas
/* demographic */
data src_dm (drop = __orig:);
   set SDTM.DM (rename = (AGEU = __origageu RACE = __origrace ETHNIC = __origethnic));
   where ARMCD ne 'Scrnfail';
   length AGEU $5 RACE $32 ETHNIC $22 AGEGR1 $5;
   AGEU = strip(__origageu);
   RACE = strip(__origrace);
   ETHNIC = strip(__origethnic);

   if input(SITEID, best.) in (702 706 707 711 714 715 717) then SITEGR1 = '900';
   else SITEGR1 = SITEID;
   TRT01P = ARM;
   if ARM =: 'Placebo' then TRT01PN = 0;
   else if find(ARM, 'Low', 'i') then TRT01PN = 54;
   else if find(ARM, 'High', 'i') then TRT01PN = 81;

   if . < AGE < 65 then AGEGR1 = '<65';
   else if 65 <= AGE <= 80 then AGEGR1 = '65-80';
   else if AGE > 80 then AGEGR1 = '>80';
   if not missing(AGEGR1) then AGEGR1N = whichc(first(AGEGR1), '<', '6', '>');

   if not missing(RACE) then
      RACEN = whichc(substr(RACE, 1, 2), 'WH', 'BL', 'xx', 'xx', 'xx', 'AM', 'AS');

   if not missing(ARMCD) then ITTFL = 'Y';
   else ITTFL = 'N';

   format RFENDT date9.;
   if not missing(RFENDTC) then RFENDT = input(RFENDTC, e8601da.);
run;
```

DISPOSITION DATA

In R, disposition data retrieval utilizes the DS data set loaded through read_xpt(). Relevant disposition events are filtered (filter()), sorted (arrange()), and selected (slice()) to retain the most significant records. Reasons for discontinuation are derived using conditional logic (case_when()).

```r
src_ds <- sdtm_data$ds %>%
  filter(DSCAT == "DISPOSITION EVENT") %>%
  arrange(USUBJID, desc(VISITNUM)) %>%
  group_by(USUBJID) %>%
  slice(1) %>%  # Keep the record with the highest VISITNUM per subject
  ungroup() %>%
  mutate(
    VISNUMEN = if_else(VISITNUM != 13, VISITNUM, 12),
    DCDECOD = DSDECOD,
    DCREASCD = case_when(
      DSDECOD == "PROTOCOL DEVIATION" & DSTERM == "PROTOCOL VIOLATION" ~ "PROTOCOL
VIOLATION",
      DSDECOD == "PROTOCOL DEVIATION" & DSTERM == "PROTOCOL ENTRY CRITERIA NOT MET" ~
"I/E NOT MET",
      !str_detect(DSDECOD, regex("^(STUDY|WITH|SCRE|LOST)", ignore_case = TRUE)) ~
toupper(DSDECOD),
      str_starts(DSDECOD, regex("STUDY", ignore_case = TRUE)) ~ "STUDY TERMINATED BY
SPONSOR",
      str_starts(DSDECOD, regex("WITH", ignore_case = TRUE)) ~ "WITHDREW CONSENT",
      str_starts(DSDECOD, regex("SCRE", ignore_case = TRUE)) ~ "I/E NOT MET",
      str_starts(DSDECOD, regex("LOST", ignore_case = TRUE)) ~ "LOST TO FOLLOW-UP",
      TRUE ~ NA_character_
    ),
    DISCONFL = if_else(DCREASCD != "COMPLETED", "Y", ""),
    DSRAEFL  = if_else(DCREASCD == "ADVERSE EVENT", "Y", "")
  ) %>%
  select(USUBJID, VISNUMEN, DCDECOD, DCREASCD, DISCONFL, DSRAEFL)
```

By the nature of SDTM data sets, they are sorted by key variables (e.g., USUBJID). Thus, SAS takes advantage of this to process the data one record at a time for each subject while utilizing the RETAIN statement to carry the value forward to the next record. **IF-THEN-ELSE** conditions are used to select appropriate records and standardize disposition reasons. Combined with the LAST. temporary variable that is automatically created in a DATA step that has a BY statement, helps to create a data set that is one record per subject.

SAS

```
data src_ds (keep = USUBJID VISNUMEN DC: DISCONFL DSRAEFL);
   set SDTM.DS;
   by USUBJID;
   retain VISNUMEN DCDECOD DCREASCD;
   length DCDECOD $27 DCREASCD $18;
   if first.USUBJID then call missing(VISNUMEN, DCDECOD, DCREASCD);

   if DSCAT = 'DISPOSITION EVENT' then do;
      if VISITNUM ^= 13 then VISNUMEN = VISITNUM;
      else VISNUMEN = 12;

      DCDECOD = DSDECOD;
      if DSDECOD not in: ('STUDY' 'WITH' 'SCRE' 'LOST') then
         DCREASCD = tranwrd(propcase(DSDECOD), ' Of ', ' of ');
      else if DSDECOD =: 'STUDY' then DCREASCD = 'Sponsor Decision';
      else if DSDECOD =: 'WITH' then DCREASCD = 'Withdrew Consent';
      else if DSDECOD =: 'SCRE' then DCREASCD = 'I/E Not Met';
      else if DSDECOD =: 'LOST' then DCREASCD = 'Lost to Follow-up';

      if DSTERM = 'PROTOCOL ENTRY CRITERIA NOT MET' then DCREASCD = 'I/E Not Met';
   end;
   if last.USUBJID;
   if DCDECOD ^= 'COMPLETED' then DISCONFL = 'Y';
   if DCDECOD = 'ADVERSE EVENT' then DSRAEFL = 'Y';

run;
```

BASELINE VITALS DATA

In R, baseline vitals data retrieval involves selecting relevant baseline measurements from the VS data set using filter() and reshaping the data with pivot_wider(). Height and weight are explicitly selected for further analysis.

R

```
src_vs <- sdtm_data$vs %>%
  filter((VSTESTCD == "HEIGHT" & VISITNUM == 1) |
           (VSTESTCD == "WEIGHT" & VISITNUM == 3)) %>%
  select(USUBJID, VSTESTCD, VSSTRESN) %>%
  pivot_wider(names_from = VSTESTCD, values_from = VSSTRESN) %>%
  rename(HEIGHTBL = HEIGHT,
         WEIGHTBL = WEIGHT)
```

SAS achieves similar results through **PROC TRANSPOSE**, which reshapes the vital signs data into a wide format after sorting and filtering for baseline measurements using PROC SORT.

SAS

```
proc sort data = SDTM.VS out = vs;
   by USUBJID VSTESTCD;
   where (VSTESTCD = 'HEIGHT' and VISITNUM = 1) or
         (VSTESTCD = 'WEIGHT' and VISITNUM = 3);
run;

proc transpose data = vs
               out = src_vs (drop = _:)
               suffix = BL;
   by USUBJID;
   id VSTESTCD;
   var VSSTRESN;
run;
```

TREATMENT START AND SUBJECT VISIT DATA

In R, treatment start dates and visit data are derived from the SV data set using conditional filtering (filter()), date transformations (ymd()), and summarizing relevant visit dates (summarise()).

```r
src_sv <- sdtm_data$sv %>%
  arrange(USUBJID, VISITNUM) %>%
  group_by(USUBJID) %>%
  mutate(
    VISIT1DT = if_else(VISITNUM == 1, ymd(SVSTDTC), as.Date(NA)),
    TRTSDT   = if_else(VISITNUM == 3, ymd(SVSTDTC), as.Date(NA)),
    `__vis4dt` = if_else(VISITNUM == 4, ymd(SVSTDTC), as.Date(NA)),
    `__vis8dt` = if_else(VISITNUM == 8, ymd(SVSTDTC), as.Date(NA)),
    `__vis10dt` = if_else(VISITNUM == 10, ymd(SVSTDTC), as.Date(NA)),
    `__vis12dt` = if_else(VISITNUM == 12, ymd(SVSTDTC), as.Date(NA))
  ) %>%
  summarise(
    VISIT1DT = last(na.omit(VISIT1DT)),
    TRTSDT   = last(na.omit(TRTSDT)),
    `__vis4dt` = last(na.omit(`__vis4dt`)),
    `__vis8dt` = last(na.omit(`__vis8dt`)),
    `__vis10dt` = last(na.omit(`__vis10dt`)),
    `__vis12dt` = last(na.omit(`__vis12dt`))
  ) %>%
  ungroup()
```

R

In SAS, similar derivations are conducted using **DATA** steps with conditional logic to identify and retain essential visit dates and treatment start information.

```sas
data src_sv (keep = USUBJID VISIT1DT TRTSDT __vis:);
   set SDTM.SV;
   by USUBJID;
   format VISIT1DT TRTSDT __vis4dt __vis8dt __vis10dt __vis12dt date9.;
   retain VISIT1DT TRTSDT __vis4dt __vis8dt __vis10dt __vis12dt;
   if first.USUBJID then call missing(VISIT1DT, TRTSDT, of __vis:);
   if VISITNUM = 1 then VISIT1DT = input(SVSTDTC, e8601da.);
   else if VISITNUM = 3 then TRTSDT = input(SVSTDTC, e8601da.);
   else if VISITNUM = 4 then __vis4dt = input(SVSTDTC, e8601da.);
   else if VISITNUM = 8 then __vis8dt = input(SVSTDTC, e8601da.);
   else if VISITNUM = 10 then __vis10dt = input(SVSTDTC, e8601da.);
   else if VISITNUM = 12 then __vis12dt = input(SVSTDTC, e8601da.);
   if last.USUBJID;
run;
```

SAS

PRIMARY DIAGNOSIS DATA

Primary diagnosis data retrieval in R selects relevant records from the MH data set through filtering (filter()) and extracts diagnosis onset dates using date transformations (ymd()).

```r
src_mh <- sdtm_data$mh %>%
  filter(MHCAT == "PRIMARY DIAGNOSIS") %>%
  mutate(DISONSDT = if_else(!is.na(MHSTDTC), ymd(MHSTDTC), as.Date(NA))) %>%
  select(USUBJID, DISONSDT)
```

R

SAS utilizes **DATA** steps with logical filters to consistently identify primary diagnosis records, extracting and formatting onset dates accordingly.

```sas
data src_mh;
   set SDTM.MH;
   where MHCAT = 'PRIMARY DIAGNOSIS';
   format DISONSDT date9.;
   if not missing(MHSTDTC) then DISONSDT = input(MHSTDTC, e8601da.);
   keep USUBJID DISONSDT;
run;
```

SAS

QUESTIONNAIRE DATA

In R, questionnaire data retrieval aggregates multiple questionnaire data sets (QS domain) using functions such as group_by() and summarise() to compute totals and derive indicators based on questionnaire responses.

R

```
src_qs <- sdtm_data$qs %>%
  group_by(USUBJID) %>%
  summarise(
    MMSETOT = sum(
      if_else(
        str_starts(str_trim(QSCAT), regex("^MMSE", ignore_case = TRUE)),
        QSSTRESN,
        0
      ),
      na.rm = TRUE
    ),
    `__effalz` = if_else(any(str_starts(str_trim(QSCAT), regex("^ALZ", ignore_case =
TRUE)) & VISITNUM > 3), 1, 0),
    `__effcli` = if_else(any(str_starts(str_trim(QSCAT), regex("^CLI", ignore_case =
TRUE)) & VISITNUM > 3), 1, 0)
  ) %>%
  ungroup()
```

SAS similarly employs the **DATA** step to aggregate and summarize questionnaire data for a specific questionnaire, accurately representing subject responses as well as deriving indicators that identify whether other types of questionnaires were completed.

SAS

```
data src_qs;
   set SDTM.QS;
   by USUBJID;
   retain MMSETOT __effalz __effcli;
   if first.USUBJID then do;
      MMSETOT = 0;
      __effalz = 0;
      __effcli = 0;
   end;
   if QSCAT =: 'MINI' then MMSETOT = sum(MMSETOT, input(QSORRES, best.));
   if QSCAT =: 'ALZ' and VISITNUM > 3 then __effalz = 1;
   if QSCAT =: 'CLI' and VISITNUM > 3 then __effcli = 1;
   if last.USUBJID;
   keep USUBJID MMSETOT __eff:;
run;
```

EXPOSURE DATA

Exposure data processing in R involves reshaping and summarizing exposure domain data (EX data set) through custom functions and transformations (pivot_wider(), mutate()) to derive treatment start and end dates as well as dosing information.

R

```
ex <- sdtm_data$ex %>%
  mutate(
    exstdt = ymd(EXSTDTC),
    exendt = ymd(EXENDTC),
    `__visc` = sprintf("%02d", VISITNUM)
  )

# Helper function to pivot exposure variables using names_glue
ex_transpose <- function(data, var, suff) {
  name_template <- paste0("EX{`__visc`}", suff)
  data %>%
    select(USUBJID, `__visc`, !!sym(var)) %>%
    pivot_wider(names_from = `__visc`, values_from = !!sym(var), names_glue =
name_template)
}

ex_st <- ex_transpose(ex, var = "exstdt", suff = "ST")
ex_en <- ex_transpose(ex, var = "exendt",  suff = "EN")
ex_ds <- ex_transpose(ex, var = "EXDOSE",  suff = "DS")

# Merge exposure data; do not calculate cumulative dose here
src_ex <- ex_st %>%
  left_join(ex_en, by = "USUBJID") %>%
  left_join(ex_ds, by = "USUBJID") %>%
  mutate(
    TRTEDT = case_when(
      !is.na(`EX12ST`) ~ `EX12EN`,
      !is.na(`EX04ST`) ~ `EX04EN`,
      !is.na(`EX03ST`) ~ `EX03EN`,
      TRUE ~ as.Date(NA)
    )
  )
```

In SAS, the EX data is pre-processed to convert dates to a numeric value. With the use of **PROC TRANSPOSE** within a macro definition (EXTRANS), the pre-processed data is transposed for three different variables and then combined within a **DATA** step with conditional logic is used to determine the end of treatment.

SAS

```
data ex;
   set SDTM.EX;
   format exstdt exendt date9.;
   if not missing(EXSTDTC) then exstdt = input(EXSTDTC, e8601da.);
   if not missing(EXENDTC) then exendt = input(EXENDTC, e8601da.);
   __visc = put(VISITNUM, Z2.);
run;

%macro extrans(suff = , var = );
   proc transpose data = ex
                  out = ex_&suff (drop = _:)
                  prefix = EX
                  suffix = &suff;
      by USUBJID;
      var &var;
      id __visc;
   run;
%mend extrans;

%extrans(suff = ST, var = exstdt)
%extrans(suff = EN, var = exendt)
%extrans(suff = DS, var = EXDOSE)

data src_ex;
   merge ex_:;
   by USUBJID;

   format TRTEDT date9.;
   if not missing(EX12ST) then TRTEDT = EX12EN;
   else if not missing(EX04ST) then TRTEDT = EX04EN;
   else if not missing(EX03ST) then TRTEDT = EX03EN;
run;
```

DATA COMBINATION AND VARIABLE CREATION

In R, combining data sets involves sequentially merging data sets using functions like left_join() from the tidyverse, ensuring data sets are matched accurately by subject identifiers (USUBJID). This approach preserves all necessary information from each data set. Variable creation in R utilizes extensive use of the mutate() function, leveraging conditional logic (if_else(), case_when()) and mathematical operations to derive new analytical variables such as BMI, treatment duration, and cumulative dose.

R

```
calculate_cumdose <- function(TRT01AN, TRTDUR, TRTSDT, TRTEDT, RFENDT, vis4, vis12) {
  if (!is.na(TRT01AN) && TRT01AN %in% c(0, 54)) {
    return(TRTDUR * TRT01AN)
  } else if (!is.na(TRT01AN) && TRT01AN == 81) {
    doseint1 <- if (!is.na(vis4) && !is.na(TRTSDT)) {
      as.numeric(min(TRTEDT, vis4) - TRTSDT + 1)
    } else if (is.na(vis4) && !is.na(TRTSDT) && !is.na(RFENDT) && (RFENDT > TRTSDT)) {
      as.numeric(TRTEDT - TRTSDT + 1)
    } else {
      NA_real_
    }
    ds_doseint1 <- if (!is.na(doseint1)) doseint1 * 54 else NA_real_

    doseint2 <- if (!is.na(vis4) && !is.na(vis12)) {
      as.numeric(min(TRTEDT, vis12) - vis4)
    } else if (is.na(vis12) && !is.na(vis4) && !is.na(RFENDT) && (RFENDT > vis4)) {
      as.numeric(TRTEDT - vis4)
    } else {
      NA_real_
    }
    ds_doseint2 <- if (!is.na(doseint2)) doseint2 * 81 else NA_real_

    doseint3 <- if (!is.na(vis12) && !is.na(RFENDT) && (RFENDT > vis12)) {
      as.numeric(TRTEDT - vis12)
    } else {
      NA_real_
    }
    ds_doseint3 <- if (!is.na(doseint3)) doseint3 * 54 else NA_real_

    return(sum(c(ds_doseint1, ds_doseint2, ds_doseint3), na.rm = TRUE))
  } else {
    return(NA_real_)
  }
```

R

```
# Process subject characteristics: extract EDLEVEL for education
sc_edulevel <- sdtm_data$sc %>%
  filter(SCTESTCD == "EDLEVEL") %>%
  mutate(EDUCLVL = as.numeric(SCSTRESN)) %>%
  select(USUBJID, EDUCLVL)

# Merge all processed domains (order mirrors SAS merge logic)
# Note: BMIBL is calculated before it is used in EFFFL.
adsl <- src_dm %>%
  left_join(src_ds, by = "USUBJID") %>%
  left_join(src_vs, by = "USUBJID") %>%
  left_join(src_sv, by = "USUBJID") %>%
  left_join(src_mh, by = "USUBJID") %>%
  left_join(src_qs, by = "USUBJID") %>%
  left_join(src_ex, by = "USUBJID") %>%
  left_join(sc_edulevel, by = "USUBJID") %>%
  filter(!is.na(STUDYID)) %>%
  mutate(
    TRT01A = TRT01P,
    TRT01AN = TRT01PN,
    SAFFL = if_else(!is.na(TRTSDT), "Y", "N"),
    # Calculate BMIBL before using it in EFFFL
    HEIGHTBL = round(HEIGHTBL, 1),
    WEIGHTBL = round(WEIGHTBL, 1),
    BMIBL = if_else(!is.na(HEIGHTBL) & !is.na(WEIGHTBL),
                    round(WEIGHTBL / ((HEIGHTBL/100)^2), 1),
                    NA_real_),
    EFFFL = if_else(SAFFL == "Y" & !is.na(BMIBL) & `__effalz` == 1 & `__effcli` == 1,
"Y", "N"),
    BMIBLGR1 = case_when(
      !is.na(BMIBL) & BMIBL < 25 ~ "<25",
      !is.na(BMIBL) & BMIBL >= 25 & BMIBL < 30 ~ "25-<30",
      !is.na(BMIBL) & BMIBL >= 30 ~ ">=30",
      TRUE ~ "<25"  # default group if BMIBL is missing
    ),
    DURDIS = if_else(!is.na(VISIT1DT) & !is.na(DISONSDT),
                     round((as.numeric(VISIT1DT - DISONSDT) + 1) / (365.25/12), 1),
                     NA_real_),
    DURDSGR1 = case_when(
      !is.na(DURDIS) & DURDIS < 12 ~ "<12",
      !is.na(DURDIS) & DURDIS >= 12 ~ ">=12",
      TRUE ~ NA_character_
    ),
    COMP8FL = if_else(!is.na(`__vis8dt`) & !is.na(RFENDT) & (RFENDT >= `__vis8dt`), "Y",
"N"),
    COMP16FL = if_else(!is.na(`__vis10dt`) & !is.na(RFENDT) & (RFENDT >= `__vis10dt`),
"Y", "N"),
    COMP24FL = if_else(!is.na(`__vis12dt`) & !is.na(RFENDT) & (RFENDT >= `__vis12dt`),
"Y", "N"),
    TRTEDT = if_else(is.na(TRTEDT) & !is.na(RFENDT) & !is.na(TRTSDT) & (RFENDT > TRTSDT),
                     RFENDT, TRTEDT),
    TRTDUR = if_else(!is.na(TRTSDT) & !is.na(TRTEDT),
                     as.numeric(TRTEDT - TRTSDT) + 1, NA_real_),
    # Calculate cumulative dose and average daily dose
    CUMDOSE = pmap_dbl(
      list(TRT01AN, TRTDUR, TRTSDT, TRTEDT, RFENDT, `__vis4dt`, `__vis12dt`),
      ~ calculate_cumdose(..1, ..2, ..3, ..4, ..5, ..6, ..7)
    ),
    AVGDD = if_else(!is.na(TRTDUR) & TRTDUR == 0,
                    TRT01PN,
                    if_else(!is.na(TRTDUR) & TRTDUR > 0,
                            round(CUMDOSE / TRTDUR, 1),
                            NA_real_)
    )
  )
```

```
# Merge with the shell data set (if needed) and restrict final_adsl to variables in the
ADSL specs
allvars <- adsl_specs$Variable
final_adsl <- if (nrow(adsl_shell) == 0) {
  adsl
} else {
  adsl_shell <- adsl_shell %>%
    mutate(
      TRTSDT = as.Date(TRTSDT, origin = "1970-01-01"),
      TRTEDT = as.Date(TRTEDT, origin = "1970-01-01"),
      VISIT1DT = as.Date(VISIT1DT, origin = "1970-01-01"),
      RFENDT = as.Date(RFENDT, origin = "1970-01-01")
    )
  bind_rows(adsl_shell, adsl)
}
final_adsl <- final_adsl %>% select(all_of(allvars))
attr(final_adsl, "label") <- "Subject-Level Analysis Data"

# Save final ADaM ADSL data set (simulate ADAM.ADSL)
ADAM_ADSL <- final_adsl
```

SAS achieves data set combination using **DATA** steps and **MERGE** statements, carefully aligning data sets by key identifiers to ensure a comprehensive, subject-level data set. In SAS, variable creation is performed through **DATA** steps employing extensive **IF-THEN-ELSE** logic, calculated fields, and formatted assignment statements to generate new analysis-ready variables effectively.

SAS

```
data adsl;
  merge src_:
        SDTM.SC (keep = USUBJID SCSTRESN SCTESTCD
                 where = (SCTESTCD = 'EDLEVEL')
                 rename = (SCSTRESN = EDUCLVL));
  by USUBJID;
  length SAFFL EFFFL $1 BMIBLGR1 $6  DURDSGR1 $4;
  if not missing(STUDYID);  /* STUDYID is only in src_dm */

  TRT01A = TRT01P;
  TRT01AN = TRT01PN;

  /* set population flags */
  SAFFL = ifc(not missing(TRTSDT), 'Y', 'N');
  EFFFL = ifc(__effalz = 1 and __effcli = 1, 'Y', 'N');

  /* CDISC has weight and height rounded to one decimal */
  if not missing(HEIGHTBL) then HEIGHTBL = round(HEIGHTBL, .1);
  if not missing(WEIGHTBL) then WEIGHTBL = round(WEIGHTBL, .1);
  if nmiss(HEIGHTBL, WEIGHTBL) = 0 then do;
     BMIBL = round(WEIGHTBL / ((HEIGHTBL / 100)**2), .1);
     if . < BMIBL < 25 then BMIBLGR1 = '<25';
     else if 25 <= BMIBL < 30 then BMIBLGR1 = '25-<30';
     else if BMIBL >= 30 then BMIBLGR1= '>=30';
  end;

  /* determine the number of months between diagnosis and visit 1 */

  DURDIS = round((VISIT1DT - DISONSDT + 1) / (365.25/12), .1);
  if not missing(DURDIS) and DURDIS < 12 then DURDSGR1 = '<12';
  else if DURDIS >= 12 then DURDSGR1 = '>=12';

  if nmiss(__vis8dt, RFENDT) = 0 and RFENDT >= __vis8dt then COMP8FL = 'Y';
  else COMP8FL = 'N';
  if nmiss(__vis10dt, RFENDT) = 0 and RFENDT >= __vis10dt then COMP16FL = 'Y';
  else COMP16FL = 'N';
  if nmiss(__vis12dt, RFENDT) = 0 and RFENDT >= __vis12dt then COMP24FL = 'Y';
  else COMP24FL = 'N';

  /* if missing treatment end date and subject discontinued after visit 3 then set end
     of treatment to discontinuation */
  if missing(TRTEDT) and nmiss(RFENDT, TRTSDT) = 0 and RFENDT >= TRTSDT then
     TRTEDT = RFENDT;

  if nmiss(TRTSDT, TRTEDT) = 0 then TRTDUR = TRTEDT - TRTSDT + 1;

  /* calculate cumulative dose */
  if TRT01AN in (0 54) then CUMDOSE = TRTDUR * TRT01AN;
  else if TRT01AN = 81 then do;
     /* dosing interval 1 on 54 mg */
     if nmiss(__vis4dt, TRTSDT) = 0 then _doseint1 = min(TRTEDT, __vis4dt) - TRTSDT + 1;
     else if missing(__vis4dt) and nmiss(TRTSDT, RFENDT) = 0 and RFENDT > TRTSDT then
          _doseint1 = TRTEDT - TRTSDT + 1;
     if not missing(_doseint1) then ds_doseint1 = _doseint1 * 54;

     /* dosing interval 2 on 81 mg */
     if nmiss(__vis4dt, __vis12dt) = 0 then
        _doseint2 = min(TRTEDT, __vis12dt) - __vis4dt;
     else if missing(__vis12dt) and nmiss(__vis4dt, RFENDT) = 0 and RFENDT > __vis4dt
          then _doseint2 = TRTEDT - __vis4dt;
     if not missing(_doseint2) then ds_doseint2 = _doseint2 * 81;

     /* dosing interval 3 on 54 mg */
     if nmiss(__vis12dt, RFENDT) = 0 and RFENDT > __vis12dt then
        _doseint3 = TRTEDT - __vis12dt;
     if not missing(_doseint3) then ds_doseint3 = _doseint3 * 54;
```

```
      CUMDOSE = sum(of ds_doseint:);
   end;

   if nmiss(TRTDUR, CUMDOSE) = 0 then AVGDD = round(CUMDOSE / TRTDUR, .1);
run;
```

THE FINAL DATA SETS & CDISC COMPARISON

In R, the final ADSL data set is generated using a combination of functions such as left_join(), mutate(), and filter() to combine previously processed data sets. The resulting data set is structured to match CDISC standards and specifications precisely. To validate the accuracy and compliance of the final data set, comparisons are conducted using the waldo package, specifically the compare() function, against a CDISC reference data set. Small differences identified during comparison are typically due to rounding errors inherent in numerical calculations.

R

```
# (Assumes the reference data set is loaded as 'adam_adsl')
CADAM_ADSL <- adam_adsl
str(CADAM_ADSL)

comp <- compare(
  CADAM_ADSL %>% arrange(USUBJID),
  ADAM_ADSL %>% arrange(USUBJID),
  x_arg = "CADAM.ADSL", y_arg = "ADAM.ADSL",
  ignore_attr = TRUE,
  max_diffs = Inf
)

# Remove ANSI escape sequences and write output
clean_comp <- gsub("\033\\[[0-9;]*m", "", comp)
writeLines(clean_comp, con = "waldo_diffs_clean.txt")
print(comp)

str(ADAM_ADSL)
str(CADAM_ADSL)
```

In SAS, the final data set is similarly created through careful setting of the shell (ADSLSKEL) that contains all the variable metadata with the transactional data (ADSL) within a **DATA** step, ensuring adherence to CDISC specifications. The validation process involves COMPARE procedure (**PROC COMPARE)**, which compares the resulting data set against a CDISC-compliant reference data set, highlighting any minor discrepancies such as rounding differences between SAS and R

SAS

```
data ADAM.ADSL (label = 'Subject-Level Analysis Data');
   set adslskel adsl;
   keep &allvars;
run;

/* compare to the CDISC version */
proc compare base = CADAM.ADSL compare = ADAM.ADSL listall;
   id USUBJID;
run;
```

Note that both R and SAS had noted a difference when compared to the CDISC-compliant reference data set.with BMIBLGR1. Per specifications, BMIBLGR1 should only be populated if BMI is non-missing.

LEVERAGING ADMIRAL FUNCTIONS FOR STREAMLINED DERIVATIONS

In our workflow, several custom data processing steps for creating an ADaM ADSL data set could be replaced by standardized functions provided by the admiral package (see admiral ADSL Tutorial). By using these functions, we can improve code readability, reproducibility, and consistency across derivation steps. The following subsections detail key transformations that could be refactored using admiral functions.

CONSISTENT DATE CONVERSIONS

Traditionally, date conversion was handled manually using functions such as ymd() from the lubridate package. admiral's derive_vars_dtm() function standardizes this conversion across domains. For example, converting the end-of-treatment date variable is now achieved as follows:

R

```
src_dm <- src_dm %>%
       derive_vars_dtm(
          source_vars = vars(RFENDTC),
          new_vars_suffix = "DT"
       )
```

This approach ensures that all date/time variables are consistently processed, reducing the risk of errors in manual conversions.

RESHAPING BASELINE VITALS DATA

Our original code used custom pivoting logic with pivot_wider() to reshape baseline vital sign data. admiral's derive_vars_pivot_wider() provides a reproducible alternative. For instance, baseline height and weight can be derived by filtering for relevant tests and reshaping as shown below:

R

```
src_vs <- derive_vars_pivot_wider(
  data = sdtm_data$vs,
  key = VSTESTCD,
  value = VSSTRESN,
  filter = VSTESTCD %in% c("HEIGHT", "WEIGHT") & VISITNUM %in% c(1, 3) ) %>%
  rename(
    HEIGHTBL =
    HEIGHT, WEIGHTBL = WEIGHT
  )
```

This approach ensures that all date/time variables are consistently processed, reducing the risk of errors in manual conversions.

SUMMARIZING QUESTIONNAIRE DATA

For the summarization of questionnaire data, specifically aggregating MMSE scores and deriving binary flags, admiral's derive_summary_vars() function provides a clean solution. Instead of manually grouping and summing the scores, we can compute subject-level summaries directly:

R

```
src_qs <- derive_summary_vars(
  data = sdtm_data$qs,
  by_vars = vars(USUBJID),
  analysis_var = QSSTRESN,
  summary_fun = sum,
  filter = str_starts(str_trim(QSCAT), regex("^MMSE", ignore_case = TRUE)),
  new_var = MMSETOT
)
```

This method reduces boilerplate code and improves transparency in how summary metrics are calculated.

MERGING DATA SETS WITH A STANDARDIZED APPROACH

Merging data from multiple SDTM data sets was previously accomplished with a series of custom left joins. admiral's derive_vars_merged_join() function standardizes the merging process. For example, merging the demographic and disposition domains is performed as follows:

```
adsl <- derive_vars_merged_join(
  new_dataset = src_dm,
  dataset_add = src_ds,
  by_vars = vars(USUBJID)
)
```
R

This approach not only simplifies the code but also ensures that the merge logic is applied consistently across studies.

CALCULATING TREATMENT DURATION

Manual arithmetic to calculate treatment duration can be error prone. admiral's derive_vars_duration() function abstracts this calculation by automatically computing the duration (in days) between two dates, with an option to add one day. The following code demonstrates this for deriving the treatment duration:

```
adsl <- adsl %>% derive_vars_duration(
  new_var = TRTDUR,
  start_date = TRTSDT,
  end_date = TRTEDT,
  out_unit = "days",
  add_one = TRUE
  )
```
R

By using this function, the code becomes more intuitive and the duration calculation more transparent to reviewers.

CONCLUSION

The comparative analysis presented in this paper underscores the strengths and benefits of transitioning to R for clinical trial data management, specifically leveraging the tidyverse and admiral packages. The demonstrated efficiency, flexibility, and compliance with CDISC standards make R a compelling alternative to traditional SAS workflows. As the clinical research industry continues to evolve, embracing open-source technologies such as R provides significant opportunities for innovation and improved data processing practices. Future directions should include extending these methodologies to additional ADaM domains and exploring deeper integration with regulatory submission processes.

REFERENCES

Wickham, H., Cetinkaya-Rundel, M., Grolemund, G. (2016). R for Data Science Second Edition. O'Reilly Media. Retrieved from https://r4ds.hadley.nz/

Clinical Data Interchange Standards Consortium. (2021). ADaM Implementation Guide Version 1.3. Retrieved from https://www.cdisc.org/standards/foundational/adam

pharmaverse.org. admiral R Package Documentation. Retrieved from https://pharmaverse.github.io/admiral/

ACKNOWLEDGMENTS

The authors thank PharmaSUG for their support of this project and for previous scholarships. Joshua J. Cook thanks his mentors, Richann Jean Watson, Kirk Paul Lafler (@sasNerd), and Dr. Achraf Cohen, for their ongoing mentorship and support.

RECOMMENDED READING

- *Project GitHub Repository*
- *CDISC*
- *Project GitHub Repository*
- *R for Data Science Second Edition*
- *Get Started with admiral*
- *ADSL with admiral*

CONTACT INFORMATION

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

Joshua J. Cook

University of West Florida

Jcook0312@outlook.com

https://joshua-j-cook-portfolios.netlify.app/

Richann Jean Watson

DataRich Consulting

Richann.watson@datarichconsulting.com

http://www.datarichconsulting.com/