

PharmaSUG 2024 - Paper ST-461
**Oncology ADaM Datasets Creation Using R Programming:
A Comprehensive Approach**
Ishwar Chouhan, Atorus Research

ABSTRACT:

Efficient analysis of clinical trial data is crucial in advancing cancer treatment strategies. This presentation showcases using R programming and packages, such as Admiralonco, as a comprehensive approach to creating ADaM datasets for oncology studies. This methodology spans initial data preparation to submission-ready datasets, ensuring compliance with industry standards and regulatory guidelines. We will cover pertinent areas such as data extraction and cleaning; variable mapping and standardization; ADaM dataset creation and variable derivation; meta-data documentation; quality control and validation, and version control and reproducibility. This systematic and transparent approach aims to enable you to create ADaM datasets, using R programming, consistently, accurately, and compliantly for the analysis of clinical trial data in the oncology domain.

INTRODUCTION:

In the relentless pursuit of effective cancer treatments, the analysis of clinical trial data stands as a pivotal beacon guiding researchers towards breakthroughs. However, amidst the complexity of oncology datasets lies a formidable challenge: how to efficiently process and analyze vast amounts of data while ensuring adherence to stringent industry standards and regulatory guidelines. In this presentation, we embark on a journey through the intricate landscape of oncology clinical trials, armed with a powerful ally: R programming.

Imagine a world where the creation of Analysis Data Model (ADaM) datasets for oncology studies transcends mere routine tasks to become a harmonious symphony of meticulous preparation and unwavering compliance. Enter Admiralonco, an innovative package within the R ecosystem, poised to revolutionize the way we approach data analysis in oncology research.

From the initial stages of data extraction and cleaning to the final submission-ready datasets, this comprehensive methodology serves as a guiding light, illuminating the path towards actionable insights. We traverse through the essential domains of variable mapping and standardization, ADaM dataset creation and variable derivation, meta-data documentation, quality control, and validation, culminating in version control and reproducibility.

Mission is clear: to empower the researcher with the tools and techniques necessary to navigate the complexities of oncology clinical trial data with confidence and precision. Through a systematic and transparent approach, we strive to arm you with the knowledge and skills to consistently, accurately, and compliantly create ADaM datasets using R programming. Together, let us embark on this journey towards unraveling the mysteries of cancer, one dataset at a time.

Overview of R Packages used in this presentation:

Package	Use
Admiral	Designed to facilitate the creation of Analysis Data Model (ADaM) datasets for clinical trials. It offers a set of functions to streamline the process of deriving

	variables, generating analysis-ready datasets, and performing common data manipulations encountered in clinical research
AdmiralOnco	specialized for oncology clinical trials, offering functions for deriving variables, handling time-to-event data, and creating analysis-ready datasets tailored to oncology research needs
haven	facilitates reading and writing of data files from other statistical software packages, such as SAS and SPSS. <code>read_sas()</code> to read SAS datasets, <code>read_xpt()</code> to read SAS .xpt into R.
pharmaversesdtm	Test data (ADaM) for the pharmaverse family of packages, to provide a one-stop-shop for ADaM test data in the pharmaverse family of packages.
pharmaversesdtm	Test data (SDTM) for the pharmaverse family of packages, to provide a one-stop-shop for SDTM test data in the pharmaverse family of packages. This includes datasets that are therapeutic area (TA)-agnostic (DM, VS, EG, etc.) as well TA-specific ones (RS, TR, OE, etc.).
Stringer	String manipulation is crucial for various data cleaning and preparation tasks, and the stringr package streamlines this process by offering a comprehensive suite of functions tailored for string operations.
lubridate	Handling date-time data in R can be challenging due to unintuitive commands and variations across different date-time objects. Additionally, ensuring robustness to time zones, leap days, and daylight savings time can be cumbersome, with R sometimes lacking necessary capabilities. Lubridate simplifies date-time operations in R, enabling tasks that R struggles with and enhancing overall functionality in managing date-time data.
dplyr	dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges, <code>mutate()</code> adds new variables that are functions of existing variable, <code>select()</code> picks variables based on their names, <code>filter()</code> picks cases based on their values, <code>summarise()</code> reduces multiple values down to a single summary, <code>arrange()</code> changes the ordering of the rows.
kableExtra	The kableExtra package in R extends the functionality of the <code>knitr::kable()</code> function, which is used to create nicely formatted tables in R Markdown documents or HTML output.

DATA EXTRACTION AND CLEANING:

Data extraction and cleaning are essential preliminary steps in the analysis of clinical trial data. By leveraging R programming and a variety of functions and techniques, you can efficiently prepare your data for subsequent analysis, ensuring its accuracy and reliability. In the scenario presented, we demonstrated how to extract raw data from multiple files, combine them into a single dataset, and perform common data cleaning tasks to address missing values, duplicates, and errors. This systematic approach sets the stage for more in-depth analysis and insights into oncology research.

Imagine you're a Statistical Programmer working for a pharmaceutical company involved in oncology research. You've been tasked with analyzing data from a clinical trial evaluating the efficacy of a new cancer drug. The raw data for this trial is stored in multiple CSV files, each containing information on patient demographics, treatment regimens, adverse events, and clinical outcomes. Your goal is to extract and clean this data using R programming to prepare it for further analysis.

1. Data Extraction:

To begin, you'll need to load the raw data files into R and combine them into a single dataset for analysis. You can use the `read.csv()` function to read each CSV file and the `rbind()` function to concatenate them row-wise into a single data frame. Below, notice that the working path is set to 'C:/Users/User/Favorites'. When setting the working directory in R by copying and pasting the path from your computer, you may encounter an issue where it doesn't work as expected. This is because you need to replace backslashes (\) with forward slashes (/).

```
#Set working directory
setwd("C:/Users/User/Favorites")
#Read CSV files
Demo←read.csv("demographic.csv")
trt←read.csv("treatment.csv")
adevent←read.csv("adverse_event.csv")
clinical_outcome←read.csv("clinical_outcomes.csv")
#combine dataframes
combined_data←rbind(demo, trt, adevent, clinical_outcomes)
```

Now, you have a single data frame `combined_data` containing all the raw data from the clinical trial.

Using the `haven` package in R, you can read in files in formats such as `.xpt` and `.sas7bdat`. Here's an example of how to do it:

```
# Load the haven package if you haven't already
install.packages("haven")
library(haven)
# Read in an xpt file
my_data_xpt ← read_xpt("path/to/your/file.xpt")
# Read in a sas7bdat file
my_data_sas ← read_sas("path/to/your/file.sas7bdat")
```

Make sure to replace "path/to/your/file.xpt" and "path/to/your/file.sas7bdat" with the actual paths to your `.xpt` and `.sas7bdat` files, respectively.

2. Data Cleaning:

Next, you'll need to clean the data to address any inconsistencies, missing values, or errors that may impact the analysis. Let's walk through some common data cleaning tasks using R functions:

a) Handling Missing Values:

You can use functions like `is.na()` to identify missing values and `complete.cases()` to remove rows with missing data.

```
# Identify missing values
missing_values <- is.na(combined_data)

# Remove rows with missing values
cleaned_data <- combined_data[complete.cases(combined_data), ]
```

b) Removing Duplicates:

Use the `duplicated()` function to identify duplicate rows and `unique()` to remove them.

```
# Identify duplicate rows
duplicate_rows <- duplicated(cleaned_data)

# Remove duplicate rows
cleaned_data <- unique(cleaned_data)
```

c) Standardizing Variable Formats:

Use functions like `tolower()` or `toupper()` to convert character variables to lowercase or uppercase for consistency.

```
# Standardize variable formats
cleaned_data$gender <- tolower(cleaned_data$gender)
```

d) Correcting Data Errors:

If there are any data errors or inconsistencies, you can use conditional statements and data manipulation functions to correct them.

```
# Correct data errors
cleaned_data$age[cleaned_data$age < 0] <- NA
```

By applying these data cleaning techniques, you ensure that your dataset is free from errors and ready for further analysis.

VARIABLE MAPPING AND STANDARDIZATION:

Variable mapping and standardization are essential steps in the preparation of clinical trial data for analysis. By leveraging R programming and a variety of functions and techniques, you can map variables to standardized formats and ensure consistency and uniformity across the dataset. In the scenario presented below, we demonstrate how to map tumor response categories to standardized criteria and standardize the format of dates in the dataset using R functions and techniques. These standardized variables set the stage for more robust and reliable analysis of oncology research data.

By employing R programming and utilizing the `Admiral` package, researchers can effectively map clinical trial variables to ADaM domains, ensuring consistency and adherence to CDISC standards. This systematic approach facilitates efficient data analysis and contributes to the advancement of oncology research and treatment strategies.

In an oncology clinical trial, researchers aim to evaluate the efficacy of a new chemotherapy regimen for treating a specific type of cancer. The trial collects various data related to patient demographics, treatment regimens, adverse events, and clinical outcomes. To comply with CDISC (Clinical Data Interchange Standards Consortium) standards and facilitate efficient analysis, the trial data needs to be mapped to ADaM (Analysis Data Model) domains using R programming.

1. Variable Mapping:

Variable mapping involves linking variables in the dataset to standardized formats or codes. Let's consider an example where you need to map tumor response categories to a common set of criteria (e.g., RECIST criteria). You can create a mapping table and use functions like `merge()` or `match()` in R to perform the mapping.

```
# Define mapping table for tumor response categories
response_mapping <- data.frame(
  original_category = c("Complete Response", "Partial Response", "Stable Disease", "Progressive Disease"),
  standardized_category = c("CR", "PR", "SD", "PD")
)

# Merge mapping table with dataset
mapped_data <- merge(cleaned_data, response_mapping, by.x = "tumor_response", by.y = "original_category", all.x = TRUE)

# Remove original tumor_response column
mapped_data$tumor_response <- NULL
```

Now, the tumor response categories in your dataset are mapped to standardized codes based on the RECIST criteria.

Examples: Mapping Clinical Trial Variables to ADaM Domains

a) Demographics:

Suppose the trial collects demographic information such as age, gender, and race/ethnicity. These variables need to be mapped to the appropriate ADaM domains, such as DM (Demographics), to ensure consistency.

```
# Mapping demographics variables to ADaM DM domain
library(admiralonco)

# Original data containing demographics variables
demographics_data <- read.csv("demographics.csv")

# Mapping age to ADaM DM domain (AGE)
age_adam <- map_to_adam(demographics_data$age, domain = "DM", variable = "AGE")

# Mapping gender to ADaM DM domain (SEX)
gender_adam <- map_to_adam(demographics_data$gender, domain = "DM", variable = "SEX")

# Mapping race/ethnicity to ADaM DM domain (RACE)
race_adam <- map_to_adam(demographics_data$race_ethnicity, domain = "DM", variable = "RACE")
```

b) Treatment Regimens:

The trial records information about the treatment regimens received by each patient, including the type of chemotherapy drugs and dosages administered. These variables need to be mapped to the appropriate ADaM domains, such as CM (Concomitant Medications) or EX (Exposure), depending on the context.

```
# Mapping treatment regimen variables to ADaM CM domain
library(admiralonco)

# Original data containing treatment regimen variables
treatment_data <- read.csv("treatment_regimens.csv")

# Mapping chemotherapy drugs to ADaM CM domain (CMTRT)
chemo_drugs_adam <- map_to_adam(treatment_data$chemotherapy_drugs, domain = "CM", variable = "CMTRT")

# Mapping dosages to ADaM CM domain (CMDOS)
dosages_adam <- map_to_adam(treatment_data$dosages, domain = "CM", variable = "CMDOS")
```

c) Adverse Events:

Adverse events experienced by patients during the trial, such as nausea, fatigue, or allergic reactions, need to be mapped to the appropriate ADaM domains, such as AE (Adverse Events), using standardized terms.

```
# Mapping adverse events variables to ADaM AE domain
library(admiralonco)

# Original data containing adverse events variables
adverse_events_data <- read.csv("adverse_events.csv")

# Mapping adverse events to ADaM AE domain (AEDECOD)
adverse_events_adam <- map_to_adam(adverse_events_data$adverse_event_terms, domain = "AE", variable = "AEDECOD")
```

d) Clinical Outcomes:

Clinical outcomes assessed during the trial, such as tumor response rates, progression-free survival, or overall survival, need to be mapped to the appropriate ADaM domains, such as LB (Laboratory), or TS (Trial Summary), depending on the type of outcome measured.

```
# Mapping clinical outcomes variables to ADaM LB domain
library(admiralmonco)

# Original data containing clinical outcomes variables
outcomes_data <- read.csv("clinical_outcomes.csv")

# Mapping tumor response rates to ADaM LB domain (LBTEST)
tumor_response_adam <- map_to_adam(outcomes_data$tumor_response_rates, domain = "LB", variable = "LBTEST")

# Mapping progression-free survival to ADaM TS domain (TSVAL)
pfs_adam <- map_to_adam(outcomes_data$progression_free_survival, domain = "TS", variable = "TSVAL")
```

2. Standardization:

Standardization involves ensuring consistency and uniformity in the format or coding of variables across the dataset.

Standardization of variables in R for analysis datasets using Admiralmonco involves ensuring that variables adhere to predefined formats, coding systems, and terminologies. This process is crucial for ensuring consistency and comparability across datasets, which is essential for reliable analysis. Below, I'll outline how standardization can be done using the Admiralmonco package in R.

Let's consider an example where you need to standardize the format of dates to a common format (e.g., YYYY-MM-DD).

```
# Standardize date format
cleaned_data$date_of_birth <- as.Date(cleaned_data$date_of_birth, format = "%m/%d/%Y")
```

By applying the **standardize_variable()** function from the Admiralmonco package, researchers can easily standardize variables in the analysis dataset to adhere to predefined formats and terminologies. This ensures consistency and comparability, enabling more reliable analysis and interpretation of oncology research data.

Example: Standardization of Variables in Oncology Analysis Dataset

Let's consider a scenario where you have an oncology analysis dataset containing variables related to tumor response rates, adverse events, and clinical outcomes. These variables may be encoded differently or have inconsistent formats, making analysis challenging. We'll demonstrate how to standardize these variables using the Admiralmonco package in R.

a) Standardization of Tumor Response Rates:

Suppose the tumor response rates in the dataset are currently encoded using various terminology standards, such as RECIST, WHO, or modified RECIST. We want to standardize these rates to adhere to the RECIST criteria.

```
# Load the admiralonco package
library(admiralonco)

# Original dataset containing tumor response rates
dataset <- read.csv("oncology_analysis_dataset.csv")

# Standardize tumor response rates to RECIST criteria
standardized_response <- standardize_variable(dataset$tumor_response, standard = "RECIST")
```

b) Standardization of Adverse Events:

Adverse events recorded in the dataset may be coded using different terminology or severity scales. We want to standardize these events to a common terminology, such as MedDRA (Medical Dictionary for Regulatory Activities).

```
# Standardize adverse events to MedDRA terminology
standardized_adverse_events <- standardize_variable(dataset$adverse_events, standard = "MedDRA")
```

c) Standardization of Clinical Outcomes:

Clinical outcomes, such as progression-free survival (PFS) or overall survival (OS), may be recorded using different units or time intervals. We want to standardize these outcomes to a common format for consistent analysis.

```
# Standardize clinical outcomes to a common format
standardized_pfs <- standardize_variable(dataset$progression_free_survival, standard = "Days")
standardized_os <- standardize_variable(dataset$overall_survival, standard = "Months")
```

DATASET CREATION AND DERIVATION:

Creating ADaM (Analysis Data Model) datasets in R involves several steps, including deriving variables, creating analysis-ready datasets, and handling missing data appropriately.

By leveraging R capabilities, you can efficiently create ADaM datasets, including analysis-ready datasets, for oncology clinical trials. This involves deriving relevant variables, preparing datasets for analysis, and handling missing data appropriately. This systematic approach ensures that the data is suitable for further analysis and contributes to the advancement of cancer treatment strategies. Below, I'll provide examples and a scenario to illustrate these processes.

Suppose you're a data scientist working on an oncology clinical trial project. The trial collects data from patients receiving a new cancer treatment, including demographics, treatment information, adverse events, and clinical outcomes. Your task is to create ADaM datasets, including analysis datasets, using R programming. This involves deriving variables, preparing datasets for analysis, and handling missing data effectively.

Example: Creating ADaM Datasets in R

a) Deriving Variables:

In the oncology trial dataset, you may need to derive variables based on existing data. For example, you might calculate the duration of progression-free survival (PFS) based on the start and end dates of treatment.

```
# Load necessary packages
library(dplyr)

# Calculate PFS duration (in days)
dataset <- dataset %>%
  mutate(PFS_duration = as.numeric(difftime(end_date, start_date, units = "days")))
```

b) Creating Analysis-Ready Datasets:

Once variables are derived, you can create analysis-ready datasets by selecting relevant variables and filtering out unnecessary data.

```
# Select relevant variables for analysis
analysis_dataset <- dataset %>%
  select(patient_id, age, gender, treatment_group, PFS_duration, adverse_event_severity)

# Filter out missing data
analysis_dataset <- na.omit(analysis_dataset)
```

c) Handling Missing Data Appropriately:

Missing data is common in clinical trials and must be handled appropriately. You can use various techniques such as imputation or exclusion based on the nature of the missingness.

```
# Impute missing values for age using mean imputation
analysis_dataset$age[is.na(analysis_dataset$age)] <- mean(analysis_dataset$age, na.rm = TRUE)

# Exclude observations with missing PFS duration
analysis_dataset <- analysis_dataset[!is.na(analysis_dataset$PFS_duration), ]
```

Example: Creating Oncology Specific ADaM Datasets in R

ADRS, ADTTE, and ADTR are commonly used ADaM (Analysis Data Model) domains in oncology clinical trials. ADRS represents Adverse Events, ADTTE represents Time-to-Event (e.g., time to progression), and ADTR represents Treatment (e.g., exposure to treatment). We'll create these domains using the Admiralonco package in R, along with examples and a scenario.

AdmiralOnco

Admiralonco is an oncology extension for ADaM in R Asset Library (Admiral) to offer a supplementary toolkit, complementing the functionality of admiral, that empowers users to tailor their approach to oncology-specific needs. Admiralonco develops a versatile set of functions and utilities to construct ADaM datasets tailored to oncology in R, designed for modularity. These functions adhere closely to the ADaM Implementation Guide, streamlining the process of adhering to ADaM dataset standards.

Initially, the package prioritizes addressing prevalent efficacy endpoints for solid tumors (utilizing RECIST v1.1 response criteria). However, future updates will expand its scope to include additional areas such as endpoints for targeted indications, baseline disease characteristics, common sensitivity analyses, and questionnaires. This package can be installed as below:

```
#> install.packages("admiralongo")
```

To acquire the most recent developmental iteration of the package directly from GitHub, employ the following code snippet:

```
if (!requireNamespace("remotes", quietly = TRUE)) {  
  install.packages("remotes")  
}  
  
remotes::install_github("pharmaverse/admiraldev") # This is a required dependency of {admiralongo}  
remotes::install_github("pharmaverse/admiral") # This is a required dependency of {admiralongo}  
remotes::install_github("pharmaverse/admiralongo")
```

A list of all available templates can be obtained by `list_all_templates()` from `admiral`, as below:

```
list_all_templates(package = "admiralongo")  
#> Existing ADaM templates in package 'admiralongo':  
#> • ADRS_BASIC  
#> • ADRS  
#> • ADTR  
#> • ADTTE
```

If you want to utilize an existing template for an ADRS dataset, you can use the following code:

```
library(admiral)  
use_ad_template(adam_name = "adrs", save_path = "./ad_adrs.R", package = "admiralongo")
```

Creating Oncology ADaM Datasets

Creating Basic ADRS

To commence, it is essential to import all requisite data frames into the environment for the generation of ADRS. This procedure may vary according to company-specific protocols. Among the necessary data frames are ADSL, RS, and TU. For illustrative purposes, we utilize SDTM and ADaM datasets (derived from CDISC Pilot test data) included within the 'pharmaversesdtm' and 'pharmaverseadam' repositories.

Steps to create ADRS:

- ❖ Read in Data
- ❖ Pre-processing of Input Records
- ❖ Derive Progressive Disease Parameter
- ❖ Derive Response Parameter
- ❖ Derive Clinical Benefit Parameter
- ❖ Derive Best Overall Response Parameter
- ❖ Derive Best Overall Response of CR/PR Parameter
- ❖ Derive Response Parameters requiring Confirmation
- ❖ Derive Parameters using Independent Review Facility (IRF)/ Blinded Independent Central Review (BICR) responses
- ❖ Derive Death Parameter
- ❖ Derive Last Disease Assessment Parameters
- ❖ Derive Measurable Disease at Baseline Parameter
- ❖ Derive AVAL for New Parameters
- ❖ Assign ASEQ
- ❖ Add ADSL variables

➤ Read in Data

To initiate the process, it is imperative to import all requisite data frames essential for generating the Analysis Dataset for Regulatory Submission (ADRS) into the working environment. This procedure is tailored to each company's specifications. Among the crucial data frames required are ADSL, RS, and TU. For illustrative purposes, we utilize SDTM and ADaM datasets, sourced from the CDISC Pilot test data, available within Pharmaversesdtm and Pharmaverseadam repositories.

```
library(admiral)
library(admiralonco)
library(dplyr)
library(pharmaversesdtm)
library(pharmaverseadam)
library(lubridate)
library(stringr)
data("adsl")
data("rs_onco_recist")
data("tu_onco_recist")
rs <- rs_onco_recist
tu <- tu_onco_recist
rs <- convert_blanks_to_na(rs)
tu <- convert_blanks_to_na(tu)
```

During this phase, it can prove advantageous to merge the ADSL (Analysis Dataset for Safety Reporting) with your RS (Raw Data Structure) domain. Specifically, only the ADSL variables pertinent to the derivations are cherry-picked at this stage. The remaining pertinent ADSL variables will be incorporated subsequently.

```
adsl_vars <- exprs(RANDDT)
adrs <- derive_vars_merged( rs, dataset_add = adsl, new_vars = adsl_vars,
  by_vars = exprs(STUDYID, USUBJID))
```

➤ Pre-processing of Input Records

The initial phase involves company-specific preprocessing of records to prepare the required input for subsequent parameter functions. It's worth noting that this preprocessing might be necessary multiple times, such as once for investigator records and once for Independent Review Facility (IRF)/Blinded Independent Central Review (BICR) records. Additionally, it might entail integrating input data from sources beyond RS, like ADTR.

This stage encompasses selecting/deriving ADT, applying partial date imputations if needed, updating AVAL (e.g., arranging from best to worst response), and establishing the analysis flag ANL01FL. Regarding ANL01FL, typical options include nullifying invalid assessments or those post new anti-cancer therapy, or flagging assessments on or after the date of first treatment/randomization. Rules might also cover instances of multiple observations per visit (e.g., selecting the worst value). Furthermore, there might be additional protocol-specific sources of Progressive Disease, like radiological assessments, which could undergo preprocessing here to generate a PD record for downstream derivations.

For parameter derivation, it's anticipated that subject identifier variables (typically STUDYID and USUBJID) and ADT form a unique key. This uniqueness can be achieved by deriving an analysis flag (ANLzzFL). Refer to Derive ANL01FL for an illustrative example.

The below shows an example of a possible company-specific implementation of this step.

➤ Select Overall Response Records and Set Parameter Details

In this scenario, our initial point of reference is the overall response records sourced from the investigator within the RS. While the specifics of parameters such as PARAMCD and PARAM are invariably tailored to each company's requirements, an illustrative example is provided below. This example aids in understanding how these records serve as inputs for subsequent parameter derivations.

```
adrs <- adrs %>%
  filter(RSEVAL == "INVESTIGATOR" & RSTESTCD == "OVLRESP") %>%
  mutate( PARAMCD = "OVR", PARAM = "Overall Response by Investigator", PARCAT1 = "Tumor Response",
    PARCAT2 = "Investigator", PARCAT3 = "RECIST 1.1" )
```

➤ Partial Date Imputation and Deriving ADT, ADTF, AVISIT etc

If your data collection permits partial dates, you have the opportunity to implement a company-specific imputation rule during the derivation of ADT. In this instance, we demonstrate an approach where missing day values are imputed with the last possible date.

```

adrs ← adrs %>%
  derive_vars_dt( dtc = RSDTC, new_vars_prefix = "A", highest_imputation = "D", date_imputation = "last"
) %>% mutate(AVISIT = VISIT)

```

➤ Derive AVALC and AVAL

In this step, we fill in AVALC and generate its numerical counterpart, AVAL, ensuring it's arranged from the best to the worst response. Notably, this ordering aligns with our RECIST v1.1 parameter derivation functions, meaning altering AVAL at this stage won't affect the outcome of those derivations.

```

adrs ← adrs %>%
  mutate( AVALC = RSSTRESC, AVAL = aval_resp(AVALC) )

```

➤ Flag Worst Assessment at Each Date (ANL01FL)

During the derivation of ANL01FL, it presents an opportunity to omit any records that shouldn't influence downstream parameter derivations. In the provided example, this involves exclusively choosing valid assessments and those happening on or after the randomization date. In cases where there are multiple assessments on a single date, the most unfavorable one is flagged.

```

worst_resp ← function(arg) {
  case_when(
    arg == "NE" ~ 1, arg == "CR" ~ 2, arg == "PR" ~ 3, arg == "SD" ~ 4, arg == "NON-CR/NON-PD" ~ 5, arg == "PD" ~ 6,
    TRUE ~ 0 )}

adrs ← adrs %>%
  restrict_derivation(
    derivation = derive_var_extreme_flag, args = params( by_vars = exprs(STUDYID, USUBJID, ADT),
    order = exprs(worst_resp(AVALC), RSSEQ), new_var = ANL01FL, mode = "last" ),
    filter = !is.na(AVAL) & ADT ≥ RANDDT )

```

Here's an alternative example where records occurring after the initiation of new anti-cancer therapy are also excluded. In this scenario, NACTDT would be pre-derived as the first date of new anti-cancer therapy. For details on deriving this variable, please refer to the documentation on "Creating and Using New Anti-Cancer Start Date" provided by Admiralonco.

```

adrs ← adrs %>%
  mutate( ANL01FL = case_when( !is.na(AVAL) & ADT ≥ RANDDT & ADT < NACTDT ~ "Y", TRUE ~ NA_character_ ) )

```

Note here that we don't filter out records after first PD at this stage, as that is specifically catered for in the Admiralonco parameter derivation functions in the below steps, via `source_pd` arguments.

➤ Flag Assessments up to First PD (ANL02FL)

If you opt not to rely on the `source_pd` arguments, an alternative approach allows users to filter out records after the first occurrence of progressive disease at this stage, utilizing an ANLzzFL flag. Subsequently,

you could keep the `source_pd` as null in all downstream parameter derivation function calls. For instance, users could establish an `ANL02FL` flag to subset the post-baseline response data, inclusive of the first reported progressive disease. This method serves as an alternative and transparent means to create ADRS parameters below. Utilizing the `admiral::derive_var_relative_flag()`, `ANL02FL` could be created as outlined below.

```
adrs <- adrs %>%
  derive_var_relative_flag( by_vars = exprs(STUDYID, USUBJID), order = exprs(ADT, RSSEQ), new_var = ANL02FL,
    condition = AVALC == "PD", mode = "first", selection = "before", inclusive = TRUE )
```

➤ Derive Progressive Disease Parameter

With the input records prepared as outlined above, incorporating any company-specific requirements, we proceed to derive new parameter records. During parameter derivations, all values are retained except those overwritten by the `set_values_to` argument, sourced from the earliest occurring input record that fulfills the specified criteria.

The `admiral::derive_extreme_records()` function proves invaluable for identifying the date of the first progressive disease.

```
adrs <- adrs %>%
  derive_extreme_records(
    dataset_ref = adsl, dataset_add = adrs, by_vars = exprs(STUDYID, USUBJID),
    filter_add = PARAMCD == "OVR" & AVALC == "PD" & ANL01FL == "Y", order = exprs(ADT, RSSEQ),
    mode = "first", exist_flag = AVALC, false_value = "N",
    set_values_to = exprs( PARAMCD = "PD", PARAM = "Disease Progression by Investigator",
      PARCAT1 = "Tumor Response", PARCAT2 = "Investigator", PARCAT3 = "RECIST 1.1",
      AVAL = yn_to_numeric(AVALC), ANL01FL = "Y"
    )
  )
```

For progressive disease, response, and death parameters, as demonstrated in the steps provided here and below, our examples illustrate them as ADRS parameters. However, they could just as easily be attained using ADSL dates or ADEVENT parameters. If you opt to store them as an ADSL date, you can utilize the function `admiral::derive_var_extreme_dt()` to identify the date of the first progressive disease as a variable, rather than as a new parameter record. All parameter derivation functions that utilize these dates offer flexibility to source them from any input source using `admiral::date_source()`. Examples showcasing this flexibility are provided below.

➤ Derive Response Parameter

The next required step is to define the source location for this newly derived `PD` date.

```
pd <- date_source( dataset_name = "adrs", date = ADT, filter = PARAMCD == "PD" & AVALC == "Y")
```

An equivalent example if using ADSL instead could be as follows (where `PDDT` would be pre-derived as first date of progressive disease).

```
pd ← date_source( dataset_name = "adsl", date = PDDT)
```

To identify the date of the first response, you can employ the **derive_param_response()** function. Notably, this function differs from **admiral::derive_extreme_records()** as it solely examines events occurring before the first progressive disease. In the example below, the response condition has been specified as complete response (CR) or partial response (PR).

```
adrs ← adrs %>%
  derive_param_response( dataset_adsl = adsl, filter_source = PARAMCD == "OVR" & AVALC %in% c("CR", "PR") & ANL01FL == "Y",
    source_pd = pd, source_datasets = list(adrs = adrs), set_values_to = exprs(
      PARAMCD = "RSP", PARAM = "Response by Investigator (confirmation not required)",
      PARCAT1 = "Tumor Response", PARCAT2 = "Investigator",
      PARCAT3 = "RECIST 1.1", AVAL = yn_to_numeric(AVALC), ANL01FL = "Y"
    )
  )
```

➤ Derive Clinical Benefit Parameter

Similarly, we now define the source location for this newly derived first response date.

```
resp ← date_source( dataset_name = "adrs", date = ADT, filter = PARAMCD == "RSP" & AVALC == "Y")
```

To derive the clinical benefit parameter, you can utilize the **derive_param_clinbenefit()** function. In our context, we define clinical benefit as either a patient exhibiting a response or experiencing a sustained period of time before the first progressive disease, which could also be termed as disease control. In this example, the "sustained period" is defined as 42 days after the randomization date, leveraging the **ref_start_window** argument.

```
adrs ← adrs %>%
  derive_param_clinbenefit( dataset_adsl = adsl, filter_source = PARAMCD == "OVR" & ANL01FL == "Y", source_resp = resp,
    source_pd = pd, source_datasets = list(adrs = adrs), reference_date = RANDDT, ref_start_window = 42,
    set_values_to = exprs(
      PARAMCD = "CB", PARAM = "Clinical Benefit by Investigator (confirmation for response not required)",
      PARCAT1 = "Tumor Response", PARCAT2 = "Investigator", PARCAT3 = "RECIST 1.1", AVAL = yn_to_numeric(AVALC),
      ANL01FL = "Y" ) )
```

➤ Derive Best Overall Response Parameter

You can utilize the **derive_param_bor()** function to derive the best overall response parameter (without requiring confirmation). Similar to the previous function, you have the option to specify the period from which you consider a stable disease (SD) or non-complete response/non-progressive disease (NON-

CR/NON-PD) as eligible. In this example, we've once again employed a period of 42 days after the randomization date.

```
adrs <- adrs %>%
  derive_param_bor( dataset_adsl = adsl, filter_source = PARAMCD == "OVR" & ANL01FL == "Y",
    source_pd = pd,
    source_datasets = list(adrs = adrs), reference_date = RANDDT, ref_start_window = 42,
    set_values_to = exprs(
      PARAMCD = "BOR", PARAM = "Best Overall Response by Investigator (confirmation not required)",
      PARCAT1 = "Tumor Response", PARCAT2 = "Investigator", PARCAT3 = "RECIST 1.1", AVAL = aval_resp(AVALC),
      ANL01FL = "Y" ))
```

Note that the above gives pre-defined AVAL values (defined by `aval_resp()`) of: "CR" ~ 1, "PR" ~ 2, "SD" ~ 3, "NON-CR/NON-PD" ~ 4, "PD" ~ 5, "NE" ~ 6, "MISSING" ~ 7.

If you'd like to provide your own company-specific ordering here you could do this as follows:

```
aval_resp_new <- function(arg) {
  case_when(
    arg == "CR" ~ 7, arg == "PR" ~ 6, arg == "SD" ~ 5, arg == "NON-CR/NON-PD" ~ 4, arg == "PD" ~ 3, arg == "NE" ~ 2,
    arg == "MISSING" ~ 1, TRUE ~ NA_real_ )}
```

Next, revise the definition of AVAL within the `set_values_to` argument of the aforementioned **`derive_param_bor()`** call. It's important to note that this adjustment will solely affect the mapping of AVAL and not impact the derivation of BOR in any manner, as the function's derivation solely depends on AVALC.

➤ Derive Best Overall Response of CR/PR Parameter

The function `admiral::derive_extreme_records()` can be used to check if a patient had a response for BOR.

```
adrs <- adrs %>%
  derive_extreme_records( dataset_ref = adsl, dataset_add = adrs, by_vars = exprs(STUDYID, USUBJID),
    filter_add = PARAMCD == "BOR" & AVALC %in% c("CR", "PR"), order = exprs(ADT, RSSEQ), mode = "first",
    exist_flag = AVALC, false_value = "N",
    set_values_to = exprs(
      PARAMCD = "BCP", PARAM = "Best Overall Response of CR/PR by Investigator (confirmation not required)",
      PARCAT1 = "Tumor Response", PARCAT2 = "Investigator", PARCAT3 = "RECIST 1.1", AVAL = yn_to_numeric(AVALC),
      ANL01FL = "Y" ) )
```

➤ Derive Response Parameters requiring Confirmation

You have the option to repeat any of the aforementioned response parameters specifically for "confirmed" responses. To achieve this, you can utilize the **`derive_param_confirmed_resp()`** and **`derive_param_confirmed_bor()`** functions. Following the derivation of confirmed response records, some of the other functions mentioned earlier can be reused by passing in these confirmed response

records. Below are examples showcasing derived parameters requiring confirmation. It's essential to note that both the initial assessment and the confirmatory assessment must occur at least 28 days apart, without applying any additional +1 to calculate the days between visits, as specified by the `ref_confirm` argument.

```
adrs ← adrs %>%
  derive_param_confirmed_resp( dataset_adsl = adsl, filter_source = PARAMCD == "OVR" & ANL01FL == "Y", source_pd = pd,
    source_datasets = list(adrs = adrs), ref_confirm = 28,
    set_values_to = exprs( PARAMCD = "CRSP", PARAM = "Confirmed Response by Investigator", PARCAT1 = "Tumor Response",
      PARCAT2 = "Investigator", PARCAT3 = "RECIST 1.1", AVAL = yn_to_numeric(AVALC), ANL01FL = "Y" )
  )
confirmed_resp ← date_source( dataset_name = "adrs", date = ADT, filter = PARAMCD == "CRSP" & AVALC == "Y")

adrs ← adrs %>%
  derive_param_clinbenefit( dataset_adsl = adsl, filter_source = PARAMCD == "OVR" & ANL01FL == "Y", source_resp = confirmed_resp,
    source_pd = pd, source_datasets = list(adrs = adrs), reference_date = RANDDT, ref_start_window = 42,
    set_values_to = exprs(
      PARAMCD = "CCB", PARAM = "Confirmed Clinical Benefit by Investigator", PARCAT1 = "Tumor Response",
      PARCAT2 = "Investigator", PARCAT3 = "RECIST 1.1", AVAL = yn_to_numeric(AVALC), ANL01FL = "Y"
    )
  ) %>%
  derive_param_confirmed_bor( dataset_adsl = adsl, filter_source = PARAMCD == "OVR" & ANL01FL == "Y", source_pd = pd,
    source_datasets = list(adrs = adrs), reference_date = RANDDT, ref_start_window = 42, ref_confirm = 28,
    set_values_to = exprs(
      PARAMCD = "CBOR", PARAM = "Best Confirmed Overall Response by Investigator", PARCAT1 = "Tumor Response",
      PARCAT2 = "Investigator", PARCAT3 = "RECIST 1.1", AVAL = aval_resp(AVALC), ANL01FL = "Y"
    )
  ) %>%
  derive_extreme_records( dataset_ref = adsl, dataset_add = adrs, by_vars = exprs(STUDYID, USUBJID),
    filter_add = PARAMCD == "CBOR" & AVALC %in% c("CR", "PR"), order = exprs(ADT, RSSEQ), mode = "first",
    exist_flag = AVALC, false_value = "N",
    set_values_to = exprs(
      PARAMCD = "CBCP", PARAM = "Best Confirmed Overall Response of CR/PR by Investigator",
      PARCAT1 = "Tumor Response", PARCAT2 = "Investigator", PARCAT3 = "RECIST 1.1", AVAL = yn_to_numeric(AVALC),
      ANL01FL = "Y"
    )
  )
)
```

- Derive Parameters using Independent Review Facility (IRF)/Blinded Independent Central Review (BICR) responses

The entire sequence of steps outlined above can be replicated for different sets of records, such as utilizing assessments from the Independent Review Facility (IRF) or Blinded Independent Central Review (BICR) instead of the investigator. To accomplish this, you simply need to substitute the initial steps with the selection of the necessary records and then provide these as input to the downstream parameter functions.

It's important to bear in mind that you'll need a new progressive disease and response source object to pass to *source_pd* and *source_resp*, respectively, when working with assessments from the IRF/BICR.

```
adrs_bicr <- rs %>%
  filter( RSEVAL == "INDEPENDENT ASSESSOR" & RSACPTFL == "Y" & RSTESTCD == "OVRRESP" ) %>%
  mutate( PARAMCD = "OVRB", PARAM = "Overall Response by BICR", PARCAT1 = "Tumor Response",
           PARCAT2 = "Blinded Independent Central Review",
           PARCAT3 = "RECIST 1.1" )
```

Then in all the calls to the parameter derivation functions you would replace the `PARAMCD == "OVR"` source with `PARAMCD == "OVRR1"`.

➤ Derive Death Parameter

You can employ the `admiral::derive_extreme_records()` function to generate a new death parameter by utilizing the death date from ADSL. It's essential to limit the columns from ADSL since we'll later merge all the necessary variables across all our ADRS records.

```
adrs <- adrs %>%
  derive_extreme_records( dataset_ref = adsl, dataset_add = adsl, by_vars = exprs(STUDYID, USUBJID),
    filter_add = !is.na(DTHDT), exist_flag = AVALC, false_value = "N",
    set_values_to = exprs( PARAMCD = "DEATH", PARAM = "Death", PARCAT1 = "Reference Event", AVAL = yn_to_numeric(AVALC),
    ANL01FL = "Y", ADT = DTHDT )
  ) %>%
  select(-DTHDT)
```

➤ Derive Last Disease Assessment Parameters

The function `admiral::derive_extreme_records()` can be used to create a parameter for last disease assessment.

```
adrs <- adrs %>%
  derive_extreme_records( dataset_ref = adsl, dataset_add = adrs, by_vars = exprs(STUDYID, USUBJID),
    filter_add = PARAMCD == "OVR" & ANL01FL == "Y", order = exprs(ADT, RSSEQ), mode = "last",
    set_values_to = exprs(
      PARAMCD = "LSTA", PARAM = "Last Disease Assessment by Investigator", PARCAT1 = "Tumor Response",
      PARCAT2 = "Investigator", PARCAT3 = "RECIST 1.1", ANL01FL = "Y"
    )
  )
```

➤ Derive Measurable Disease at Baseline Parameter

You can utilize the `admiral::derive_param_exist_flag()` function to verify if a patient exhibits measurable disease at baseline, based on a company-specific criterion. In this instance, we examine TU for target lesions during the baseline visit. It's crucial to limit the columns from ADSL since we'll subsequently merge all the necessary variables across all our ADRS records.

```

adrs <- adrs %>%
  derive_param_exist_flag( dataset_ref = adslmdis, dataset_add = tu,
    condition = TUEVAL == "INVESTIGATOR" & TUSTRESC == "TARGET" & VISIT == "SCREENING",
    false_value = "N", missing_value = "N",
    set_values_to = exprs(
      PARAMCD = "MDIS", PARAM = "Measurable Disease at Baseline by Investigator", PARCAT2 = "Investigator",
      PARCAT3 = "RECIST 1.1", AVAL = yn_to_numeric(AVALC), ANL01FL = "Y"
    )
  )

```

> Assign ASEQ

The function `admiral::derive_var_obs_number()` can be used to derive `ASEQ`. An example call is:

```

adrs <- adrs %>%
  derive_var_obs_number( by_vars = exprs(STUDYID, USUBJID), order = exprs(PARAMCD, ADT, VISITNUM, RSSEQ), check_type = "error"
  )

```

> Add ADSL variables

If needed, the other ADSL variables can now be added. List of ADSL variables already merged held in vector `adsl_vars`.

> Saving final dataset

```

# —— Save output ——
# Change to whichever directory you want to save the dataset in
dir <- tools::R_user_dir("admiralonco_templates_data", which = "cache")
if (!file.exists(dir)) {
  # Create the folder
  dir.create(dir, recursive = TRUE, showWarnings = FALSE)
}
save(adrs, file = file.path(dir, "adrs_basic.rda"), compress = "bzip2")

```

Creating ADTTE

This section outlines the creation of an ADaM (Analysis Dataset Model) for time-to-event (ADTTE) data, focusing on common oncology endpoint parameters.

Central to programming a time-to-event dataset is defining the events and censoring times. Admiral/Admiralonco facilitates single events like death (Overall Survival) or composite events like disease progression or death (Progression Free Survival). Additionally, multiple source datasets can contribute to defining the event and censoring times.

While most functions utilized here are available in `admiral`, the `tte_sources` helper object is introduced as an example from Admiralonco. Typically, each company would tailor their version of this object to suit their specific requirements, as filtering conditions and metadata descriptions are likely to vary.

Required Packages:

```
library(admiral)
library(admiralonco)
library(pharmaverseadam)
library(dplyr)
library(lubridate)
```

Steps to create ADTTE:

- ❖ Read in Data
- ❖ Derive Parameters (CNSR, ADT, STARTDT)
- ❖ Derive Analysis Value (AVAL)
- ❖ Derive Analysis Sequence Number (ASEQ)
- ❖ Add ADSL Variables

➤ Read in Data

Initially, all datasets required for constructing the time-to-event dataset should be imported into the environment, following a process customized to the company's specifications.

For illustrative purposes, the ADaM datasets contained within Pharmaverseadam are utilized. Alternatively, one might opt to utilize ADEVENT as input.

```
data("adsl")
data("adrs_onco")
adrs <- adrs_onco
```

➤ Derive Parameters (CNSR, ADT, STARTDT)

To compute dependent variables such as CNSR, ADT, STARTDT, EVNTDESC, SRCDOM, PARAMCD, etc., you can employ the `admiral::derive_param_tte()` function. This function appends a parameter to the input dataset, with one observation per subject. Typically, it's invoked multiple times.

For each subject, the function assesses whether an event occurred. If so, the analysis date (ADT) is established as the earliest event date. In the absence of an event, the analysis date is determined as the latest censoring date.

The events and censorings are defined by the `admiral::event_source()` and the `admiral::censor_source()` class respectively. It defines

- which observations (filter parameter) of a source dataset (dataset_name parameter) are potential events or censorings,
- the value of the CNSR variable (censor parameter), and
- which variable provides the date (date parameter).

The date can be provided in three formats: date (--DT variable), datetime (--DTM variable), or character ISO-8601 date (--DTC variable).

CDISC strongly advocates for CNSR = 0 for events and positive integers for censorings. Admiral/Admiralonco adhere to this recommendation. Consequently, the censor parameter is exclusive to `admiral::censor_source()` and defaults to 1.

The `dataset_name` parameter requires a character value serving as an identifier. The actual data used for deriving the parameter is furnished via the `source_datasets` parameter of `admiral::derive_param_tte()`. It anticipates a named list of datasets, where the names correspond to the identifiers specified for the `dataset_name` parameter. This enables defining events and censoring independently of the data.

➤ Pre-Defined Time-to-Event Source Objects

The table below shows all pre-defined `tte_source` objects which should cover the most common oncology use cases.

object	dataset_name	filter	date	censor	set_values_to
lastalive_censor	adsl	NULL	LSTALVDT	1	EVNTDESC: "Alive" CNSDTDSC: "Alive During Study" SRCDOM: "ADSL" SRCVAR: "LSTALVDT"
trts_censor	adsl	NULL	TRTSDT	1	EVNTDESC: "Treatment Start" CNSDTDSC: "Treatment Start" SRCDOM: "ADSL" SRCVAR: "TRTSDT"
pd_event	adrs	PARAMCD == "PD" & AVALC == "Y" & ANL01FL == "Y"	ADT	0	EVNTDESC: "Disease Progression" SRCDOM: "ADRS" SRCVAR: "ADT" SRCSEQ: ASEQ
death_event	adrs	PARAMCD == "DEATH" & AVALC == "Y" & ANL01FL == "Y"	ADT	0	EVNTDESC: "Death" SRCDOM: "ADRS" SRCVAR: "ADT" SRCSEQ: ASEQ
lasta_censor	adrs	PARAMCD == "LSTA" & ANL01FL == "Y"	ADT	1	EVNTDESC: "Last Tumor Assessment" CNSDTDSC: "Last Tumor Assessment" SRCDOM: "ADRS" SRCVAR: "ADT" SRCSEQ: ASEQ

rand_censor	adsl	NULL	RANDDT	1	EVNTDESC: "Randomization" CNSDTDSC: "Randomization" SRCDOM: "ADSL" SRCVAR: "RANDDT"
-------------	------	------	--------	---	--

As highlighted earlier, each company would tailor their version of this process to align with their specific filtering conditions and metadata requirements. An alternative approach could involve sourcing death data from ADSL instead of ADRS, and utilizing a different EVNTDESC.

```
adsl_death_event <- event_source( dataset_name = "adsl", date = DTHDT,
  set_values_to = exprs( EVNTDESC = "STUDY DEATH", SRCDOM = "ADSL", SRCVAR = "DTHDT" ))
```

At this stage, an optional step might be necessary to facilitate the derivation of response duration. If utilizing ADRS/ADEVENT parameters as input for any response dates (as opposed to a variable in ADSL), you would utilize `admiral::derive_vars_merged()` to introduce the response date as a temporary variable (e.g., TEMP_RESPDT). This temporary variable would then serve as the start date for `admiral::derive_param_tte()`. Additionally, you would use this process to filter the source ADSL dataset to exclusively derive records for responders. This step could also be replicated as needed for IRF/BICR and confirmed responses.

Here is an example of the code needed.

```
adsl <- adsl %>%
  derive_vars_merged( dataset_add = adrs, filter_add = PARAMCD == "RSP" & AVALC == "Y" & ANL01FL == "Y",
    by_vars = exprs(STUDYID, USUBJID), new_vars = exprs(TEMP_RESPDT = ADT) )
```

You can directly pass the predefined objects to `admiral::derive_param_tte()` to generate a new time-to-event parameter. Below are example calls for Overall Survival (OS), Progression Free Survival (PFS), and duration of response (which, as mentioned, is solely derived for responder patients, thus necessitating filtering the source ADSL dataset). Including a randomization date censor helps capture patients who never undergo tumor assessment.

```
adtte <- derive_param_tte(
  dataset_adsl = adsl, start_date = RANDDT, event_conditions = list(death_event), censor_conditions = list(lastalive_censor, rand_censor),
  source_datasets = list(adsl = adsl, adrs = adrs), set_values_to = exprs(PARAMCD = "OS", PARAM = "Overall Survival")) %>%
  derive_param_tte(
    dataset_adsl = adsl, start_date = RANDDT, event_conditions = list(pd_event, death_event), censor_conditions = list(lasta_censor, rand_censor),
    source_datasets = list(adsl = adsl, adrs = adrs), set_values_to = exprs(PARAMCD = "PFS", PARAM = "Progression Free Survival") ) %>%
  derive_param_tte(
    dataset_adsl = filter(adsl, !is.na(TEMP_RESPDT)), start_date = TEMP_RESPDT, event_conditions = list(pd_event, death_event),
    censor_conditions = list(lasta_censor), source_datasets = list(adsl = adsl, adrs = adrs), set_values_to = exprs(PARAMCD = "RSD", PARAM = "Duration of Response")
  )
```

➤ Creating Your Own Time-to-Event Source Objects

We recommend consulting the `admiral` Creating a BDS Time-to-Event ADaM vignette for comprehensive guidance on various options and more examples.

In another common oncology scenario, concerning PFS when censoring at new anti-cancer therapy, there are two approaches:

- Utilize ANLzzFL to ensure that records after new anti-cancer therapy never contribute to the PD and DEATH parameters, as explained in the ADRS vignette.

- Control this on the ADTTE side by filtering which records are utilized in `admiral::event_source()` and `admiral::censor_source()`. For instance, for PD or death event date, you can use the `filter` argument to exclude events occurring after new anti-cancer therapy.

Regarding the censor, it can be set to any date required for your analysis, such as the date of the last tumor assessment prior to new anti-cancer therapy or the last radiological assessment. If multiple censor dates are provided, the function will select the latest occurring date among them. Therefore, exercise caution when feeding in multiple censor dates, ensuring only one is set per patient.

This scenario is illustrated in the example below (where `NACTDT` is pre-derived as the first date of new anti-cancer therapy, and `LASTANDT` is the single tumor assessment censor date as described above). Please refer to [Admiral onco Creating and Using New Anti-Cancer Start Date for deriving NACTDT](#).

```
pd_nact_event <- event_source( dataset_name = "adsl", filter = PDDT < NACTDT | is.na(NACTDT), date = PDDT,
  set_values_to = exprs( EVNTDESC = "Disease Progression prior to NACT", SRCDOM = "ADSL", SRCVAR = "PDDT" ))

death_nact_event <- event_source( dataset_name = "adsl", filter = DTHDT < NACTDT | is.na(NACTDT), date = DTHDT,
  set_values_to = exprs( EVNTDESC = "Death prior to NACT", SRCDOM = "ADSL", SRCVAR = "DTHDT" ))

lasta_nact_censor <- censor_source( dataset_name = "adsl", date = LASTANDT,
  set_values_to = exprs( EVNTDESC = "Last Tumor Assessment prior to NACT", CNSDDESC = "Last Tumor Assessment prior to NACT", SRCDOM = "ADSL",
    SRCVAR = "LASTANDT" ))

adtte <- derive_param_tte( dataset_adsl = adsl, start_date = RANDDT,
  event_conditions = list(pd_nact_event, death_nact_event),
  censor_conditions = list(lasta_nact_censor, rand_censor), source_datasets = list(adsl = adsl),
  set_values_to = exprs(PARAMCD = "PFSNACT", PARAM = "Progression Free Survival prior to NACT"))
```

➤ Derive Analysis Value (AVAL)

The analysis value (AVAL) can be derived by calling `admiral::derive_vars_duration()`. This example derives the time to event in days.

```
adtte <- adtte %>% derive_vars_duration( new_var = AVAL, start_date = STARTDT, end_date = ADT )
```

Other time units, such as months commonly used in oncology analyses, can be specified by indicating the `out_unit` parameter. Refer to the example below. It's worth noting that due to the underlying use of the `lubridate::time_length()` function, the performance may slightly deviate from expectations. For instance, both `time_length(ymd("2021-01-01") %--% ymd("2021-02-01"), "month")` and `time_length(ymd("2021-02-01") %--% ymd("2021-03-01"), "month")` result in exactly 1 month. While this is a logical approach, it differs from the convention of assuming every month has an equal number of days and simply using `/30.4375` or a similar convention. This discrepancy is noticeable only for short durations. However, if the user prefers an alternative approach, they could calculate in default days and then incorporate additional processing to convert to months using their company-specific convention.

```
adtte_months <- adtte %>% derive_vars_duration( new_var = AVAL, start_date = STARTDT, end_date = ADT, out_unit = "months" )
```

➤ Derive Analysis Sequence Number (ASEQ)

The `admiral` function `admiral::derive_var_obs_number()` can be used to derive ASEQ:

```
adtte <- adtte %>% derive_var_obs_number( by_vars = exprs(STUDYID, USUBJID), order = exprs(PARAMCD), check_type = "error" )
```

➤ Add ADSL Variables

Variables from ADSL which are required for time-to-event analyses, e.g., treatment variables or covariates can be added using `admiral::derive_vars_merged()`.

```
adtte <- adtte %>% derive_vars_merged( dataset_add = adsl, new_vars = exprs(ARMCD, ARM, ACTARMCD, ACTARM, AGE, SEX),  
                                     by_vars = exprs(STUDYID, USUBJID) )
```

Saving the final dataset:

```
# Save output ———  
  
# Change to whichever directory you want to save the dataset in  
dir <- tools::R_user_dir("admiralco_templates_data", which = "cache")  
if (!file.exists(dir)) {  
  # Create the folder  
  dir.create(dir, recursive = TRUE, showWarnings = FALSE)  
}  
save(adtte, file = file.path(dir, "adtte.rda"), compress = "bzip2")
```

METADATA DOCUMENTATION:

By generating comprehensive metadata documentation using R scripts, you ensure transparency and traceability in the creation of ADaM datasets, facilitating regulatory compliance and reproducibility in oncology clinical trials.

You're a Statistical Programmer working on an oncology clinical trial project. As part of your responsibilities, you need to ensure transparency and traceability in the creation of ADaM (Analysis Data Model) datasets for regulatory submission. Your task is to generate comprehensive metadata documentation using R scripts, detailing the dataset creation process and ensuring transparency throughout.

Example: Generating Comprehensive Metadata Documentation

```

# Load necessary packages
library(dplyr)
library(tidyr)
library(knitr)
library(kableExtra)

# Load cleaned and transformed datasets
demographics <- read.csv("demographics_cleaned.csv")
treatment <- read.csv("treatment_cleaned.csv")
adverse_events <- read.csv("adverse_events_cleaned.csv")
clinical_outcomes <- read.csv("clinical_outcomes_cleaned.csv")

# Merge datasets into a single ADaM dataset
adam_dataset <- demographics %>%
  left_join(treatment, by = "patient_id") %>%
  left_join(adverse_events, by = "patient_id") %>%
  left_join(clinical_outcomes, by = "patient_id")

# Generate metadata documentation
metadata <- data.frame(
  Variable = c("patient_id", "age", "gender", "treatment_group", "adverse_event", "tumor_response"),
  Description = c("Unique identifier for each patient", "Age of the patient at baseline", "Gender of the patient",
    "Treatment group assigned to the patient", "Adverse event experienced by the patient", "Tumor response as assessed by the investigator"),
  Type = c("Numeric", "Numeric", "Categorical", "Categorical", "Categorical", "Categorical"),
  Format = c("Integer", "Numeric", "Character", "Character", "Character", "Character"),
  Units = c("N/A", "Years", "N/A", "N/A", "N/A", "N/A"),
  Missing_Values = c("N/A", "N/A", "N/A", "N/A", "N/A", "N/A")
)

# Print metadata documentation
kable(metadata, caption = "Metadata Documentation for ADaM Dataset") %>%
  kable_styling(bootstrap_options = c("striped", "hover"), full_width = FALSE) %>%
  scroll_box(width = "100%", height = "400px")

```

In this example, we load cleaned and transformed datasets for demographics, treatment, adverse events, and clinical outcomes. We then merge these datasets into a single ADaM dataset. Next, we generate comprehensive metadata documentation describing each variable in the dataset, including its description, type, format, units, and handling of missing values. Finally, we present the metadata documentation in a structured and readable format using the `kable()` function from the `kableExtra` package.

QUALITY CONTROL AND VALIDATION:

By utilizing R programming for rigorous quality control checks and validation procedures, we can identify and address discrepancies or anomalies in the ADaM dataset, ensuring the integrity and reliability of the data for further analysis in oncology clinical trials.

You are a biostatistician working on an oncology clinical trial project. As part of your role, you are responsible for ensuring the quality and integrity of the ADaM (Analysis Data Model) datasets generated for analysis. Utilizing R programming, you need to perform rigorous quality control checks and validation procedures to identify and address any discrepancies or anomalies in the ADaM dataset.

Example: Quality Control Checks and Validation Procedures

```

# Load necessary packages
library(dplyr)

# Load ADaM dataset
adam_dataset <- read.csv("adam_dataset.csv")

# Perform quality control checks
# Check for missing values
missing_values <- colSums(is.na(adam_dataset))

# Check for outliers in continuous variables
outliers <- sapply(adam_dataset[, sapply(adam_dataset, is.numeric)], function(x) any(abs(scale(x)) > 3))

# Check for consistency in categorical variables
inconsistent_categories <- adam_dataset %>%
  select_if(is.character) %>%
  summarise_all(function(x) length(unique(x)) > 1)

# Print QC findings
cat("Missing Values:\n")
print(missing_values)
cat("\nOutliers:\n")
print(outliers)
cat("\nInconsistent Categories:\n")
print(inconsistent_categories)

# Perform validation procedures
# Validate treatment group assignments
valid_treatment_groups <- c("Placebo", "Drug A", "Drug B")
invalid_treatment_assignments <- subset(adam_dataset, !treatment_group %in% valid_treatment_groups)

# Validate tumor response categories
valid_response_categories <- c("Complete Response", "Partial Response", "Stable Disease", "Progressive Disease")
invalid_response_categories <- subset(adam_dataset, !tumor_response %in% valid_response_categories)

# Print validation findings
cat("\nInvalid Treatment Assignments:\n")
print(invalid_treatment_assignments)
cat("\nInvalid Tumor Response Categories:\n")
print(invalid_response_categories)

```

In this example, we start by loading the ADaM dataset generated for the oncology clinical trial. We then perform several quality control checks, including identifying missing values, outliers in continuous variables, and inconsistencies in categorical variables.

Next, we conduct validation procedures to ensure the accuracy and validity of specific variables. For example, we validate treatment group assignments and tumor response categories against predefined criteria.

REPRODUCIBILITY AND VERSION CONTROL:

By employing R's version control and script capabilities, you ensure the reproducibility, efficiency, traceability, and transparency of the ADaM dataset creation process in oncology clinical trials. This systematic approach facilitates collaboration and enables efficient updates and modifications to the dataset creation process.

You are a data scientist working on an oncology clinical trial project. As part of your responsibilities, you need to ensure the reproducibility of the ADaM (Analysis Data Model) dataset creation process. Leveraging R's version control and script capabilities, you aim to enable efficient updates and modifications to the dataset creation process while maintaining transparency and traceability.

Example: Utilizing Version Control and Scripting for Reproducible Dataset Creation

a) Version Control with Git:

Start by initializing a Git repository for your project to track changes to your R scripts and datasets. To initiate a Git repository from within R, you can use the `git2r` package, which provides an interface to the `libgit2` library. Make sure you have installed the `git2r` package into your R session and call all functions of this package using: `library(git2r)`.

```
install.packages("git2r")
library(git2r)
init()
```

b) Scripting the Dataset Creation Process:

Write R scripts to perform each step of the ADaM dataset creation process, including data cleaning, variable mapping, and validation checks. Save these scripts in the project directory.

```
# Script 1: Data Cleaning
# data_cleaning.R

# Load necessary packages
library(dplyr)

# Load raw data
raw_data <- read.csv("raw_data.csv")

# Perform data cleaning tasks
cleaned_data <- raw_data %>%
  filter(!is.na(variable1)) %>%
  mutate(variable2 = ifelse(variable2 < 0, NA, variable2))

# Save cleaned data
write.csv(cleaned_data, "cleaned_data.csv", row.names = FALSE)
```

c) Executing the Scripts:

Execute the R scripts sequentially to perform the dataset creation process.

```
Rscript data_cleaning.R
```

d) Versioning the Scripts:

Commit the R scripts and any changes to the Git repository to track the history of modifications.

```
git add data_cleaning.R
git commit -m "Performed data cleaning tasks"
```

e) Reproducibility and Efficiency:

By scripting the dataset creation process and utilizing version control with Git, you ensure the reproducibility of the process. Any updates or modifications to the scripts can be tracked, facilitating efficient collaboration and transparency.

```
# Update the data cleaning script
# data_cleaning.R

# Perform additional cleaning tasks
cleaned_data <- cleaned_data %>%
  mutate(variable3 = ifelse(is.na(variable3), "Unknown", variable3))

# Save updated data
write.csv(cleaned_data, "cleaned_data.csv", row.names = FALSE)

# Commit the changes
git add data_cleaning.R
git commit -m "Added additional cleaning tasks"
```

f) Traceability and Transparency:

With version control and script capabilities, you can trace the history of modifications and ensure transparency in the dataset creation process. Anyone with access to the repository can review the changes made to the scripts and datasets over time.

CONCLUSION:

This holistic approach ensures not only the meticulous extraction and cleaning of data but also the standardization and mapping of variables, all while adhering to stringent industry standards and regulatory guidelines. By delving into pertinent areas such as meta-data documentation, quality control, and validation, we reinforce the transparency and traceability of our dataset creation process.

Moreover, our emphasis on version control and reproducibility underscores the importance of maintaining the integrity and consistency of our analyses. By employing R's capabilities in a systematic and transparent manner, we empower researchers to consistently generate ADaM datasets with precision, accuracy, and compliance.

Ultimately, this presentation serves as a guiding beacon for researchers in the oncology domain, equipping them with the tools and techniques necessary to navigate the complexities of clinical trial data analysis effectively. Through our collective efforts, we strive towards unlocking novel insights and breakthroughs that hold the promise of transforming cancer treatment paradigms and improving patient outcomes.

REFERENCE:

1. <https://pharmaverse.github.io/admiralonco/index.html>
2. <https://cloud.r-project.org/web/packages/admiralonco/index.html>
3. <https://pharmaverse.org/>
4. <https://www.pharmasug.org/proceedings/2023/SD/PharmaSUG-2023-SD-325.pdf>
5. <https://www.cdisc.org/standards/foundational/sdtmig/sdtmig-v3-4>
6. <https://www.cdisc.org/standards/foundational/adam/adamig-v1-3>
7. https://ctep.cancer.gov/protocoldevelopment/docs/recist_guideline.pdf

CONTACT INFORMATION:

Ishwar Singh Chouhan, Atorus Research

Email: Ishwar.Chouhan@atorusresearch.com, ic57739@gmail.com

Any brand and product names are trademarks of their respective companies.