Paper SS05 - PHUSE US 2023

Up-to-date with Dates – Imputation Rules in R Compared to SAS

Monika Ludwińska, Intego Group, Warsaw, Poland

ABSTRACT

R programming language is a commonly known data analysis tool and over the past few years has been successfully adopted in the pharmaceutical industry. Representing date variables is crucial as well as a problematic issue. They are a critical part in clinical trials but partial dates are inevitable in analytic data. However, there is a need to investigate methods of handling different imputation techniques.

This paper focuses on handling dates in R and highlights functions useful for transformations while creating dates in a proper format. Moreover, it introduces fundamentals of working with date, time and datetime values. It explores imputation rules with R functions and techniques. What is more, it compares them to SAS ® which is a well-known tool in calculations in clinical trials. This work will show how important it is to impute date variables. Moreover, advantages of adopting R language in this process will be presented on clinical data cases.

INTRODUCTION

Timing variables - date, time and datetime variables – are an essential component and describe timing of the observation (such as start date and end date) and might be copied from SDTM (Study Data Tabulation Model) or derived within ADaM (Analysis Data Model). Derived timing variables in ADaM datasets are stored as numeric and should be formatted to be human-readable with no loss of precision. Variables whose names end in DT are numeric dates, DTM are numeric datetimes and TM are numeric times.

Unfortunately, in SDTM domains partial timing variables are inevitable so Analysis Data Model Implementation Guide provides rules for the imputation flag if date or time is imputed.

This paper will gather information about handling timing variables with different R functions in the most frequently used packages in clinical programming along with describing imputation rules provided by ADaM Implementation Guide. Furthermore, this paper will show comparison between SAS and R and how validation of timing variables in R (while development side is prepared in SAS) is being performed. Because SDTM timing variables come in character format, this paper will only focus on computing numeric date variables from character.

FUNDAMENTALS OF WORKING WITH TIMING VARIABLES

SDTM Implementation Guide requires using the international standard ISO 8601 format to store timing variables, which provides text-based representation of dates and/or times, intervals of time and durations of time. ISO 8601 allows dates to be represented in multiple ways. Here is the example of how a datetime variable may appear in any domain:

2023-01-22T20:51:20

Derived timing variables in ADaM datasets must reflect those copied from the domains. If a datetime variable is derived along with time and date variables, both have to be associated with the *DTM variable. In general, all the three variables are not required, only those needed for the analysis should be provided. However, if the *DTM variable exists, it would be reasonable to also include corresponding *DT variable.

As mentioned before, commonly there are countless amounts of data where timing variables are partial and deriving them in ADaM datasets is problematic. Due to the fact that day, month, year or whole date might be missing, they have to be properly imputed along with proper value in the imputation flag. The ADaM Implementation Guide provides the instruction of setting the imputation flag while calculating partial dates. Imputation rules while deriving missing any part of date can be specific for any study and described in Statistical Analysis Plan. An algorithm for assigning missing element of date might differ between studies but should be always precisely defined in Statistical Analysis Plan. Accurately described dates are integral part of data needed for the analysis.

As R programming language has been increasingly adopted in the pharmaceutical industry, clinical programmers should be familiar with imputation of partial dates in both, SAS and R. One of the advantages of using R in statistical programming is a vast number of packages which could be used for producing specialized outputs. For calculating dates, major role plays base R along with several specific packages. In this paper, the focus will be on the *base*, *lubridate* and *hms* packages.

R BASE PACKAGE

Base is the package which contains basic functions which allow R functions as a language. It grants conversion of a character variables into a date objects with using *as.POSIXlt()/as.POSIXct()* functions for datetime and the *as.Date()* function for date variables. While using *base*, the most essential is proper usage of the *format* argument. The *format* argument should include format from a character variable, so from an input. It is not used to change display of a timing variable. Default formats for datetime and date variables are YYYY-MM-DD HH:MM:SS and YYYY-MM-DD and can be omitted.

DATETIME VARIABLES

Datetime variables can be transformed with *as.POSIXlt()* (local time) and *as.POSIXct()* (calendar time) functions. Common datetime representation in clinical data is YYYY-MM-DD**T**HH:MM. "T" is not a standard character for R functions so it has to be taken into consideration when specifying the *format* argument to code correctly. It should be noted that the time zone of the computer system is assigned as default. The time zone can be changed with the *tz* argument.

```
as.POSIXct(input, format)
```

The example below provides usage of the *as.POSIXct()* function with the default format which is YYYY-MM-DD HH:MM:SS. As mentioned before, the *format* argument can be omitted and the time zone is added to an output. If a character variable does not contain seconds, they are added automatically to an output.

```
> as.POSIXct("2022-09-13 12:40")
[1] "2022-09-13 12:40:00 EDT"
```

Regularly, in clinical data datetime variables are stored with "T" character so it has to be specified in the *format* argument.

```
> as.POSIXct("2022-09-13T12:40", format="%Y-%m-%dT%H:%M")
[1] "2022-09-13 12:40:00 EDT"
```

In the example below, if the *format* argument is omitted when the character variable contains "T", only the date is outputted.

```
> as.POSIXct("2022-09-13T12:40:00")
[1] "2022-09-13 EDT"
```

A *POSIXct* object stores the number of seconds since 1970-01-01. Unlike the *Date* class, it is intended to also store hours, minutes and seconds. As an output, two classes are assigned for a datetime variable – *POSIXct* is inherited from the *POSIXt* class for representing both date and time.

```
> class(as.POSIXct("2022-09-13T12:40", format ="%Y-%m-%dT%H:%M"))
[1] "POSIXct" "POSIXt"
```

DATE VARIABLES

The *base* package includes the *as.Date()* function which allows converting between character representations and objects of the *Date* class representing calendar dates. The default format follows the rules of the ISO 8601 international standard which represents a date as YYYY-MM-DD. The *format* which should be passed to this function must describe what the string contains, rather than what is expected as an output.

```
as.Date(input, format)
```

If a date is stored as YYYY-MM-DD, using *as.Date()* returns a date object in the same format as the default format - in that case the *format* argument can be omitted. Character dates are stored in this way regularly in SDTM domains.

```
> as.Date("2021-01-27")
[1] "2021-01-27"
```

As already mentioned, the *format* argument is not used to change the display of date variable but to properly describe format of an input to perform valid conversion. The example below provides output, when the inaccurate format is specified (inadequate delimiter).

```
> as.Date("2021/01/27", format="%Y-%m-%d")
[1] NA
```

If the proper format is implemented, the *Date* class object is set properly. Eventually, a date object is displayed always as YYYY-MM-DD.

```
> as.Date("2021/01/27", format="%Y/%m/%d")
[1] "2021-01-27"
```

It is worth mentioning that the *as.Date()* function works on a datetime character variable.

```
> as.Date("2022-09-13T12:40")
[1] "2022-09-13"
```

Dates are represented by the *Date* class.

```
> class(as.Date("2021-01-27"))
[1] "Date"
```

R LUBRIDATE PACKAGE

Lubridate is the package, which makes it easier to work with timing variables. It is specifically designed to handle timing variables. This package also provides converting character variables into date objects. *Lubridate* is more convenient to use – it is more intuitive and offers more specific functions for handling dates or intervals.
It should be emphasized that in the *base* package usage of proper format is vital. In *lubridate* on the other hand, a user has to be meticulous with order of parts in timing variable, for instance if it is presented as YYYY-MM-DD or DD-MM-YYYY.

DATETIME VARIABLES

For datetime variables, the *lubridate* package offers many functions, all revolving around the order of parts of timing variable.
In clinical trials data, mainly datetime variables appear in the following order: YYYY-MM-DD**T**HH:MM. In this case, the function *ymd_hm()* can be used. If an inadequate function is adopted for the character variable, R throws a warning.

```
> ymd_hms("2022-09-13T12:40")
[1] NA
Warning message:
All formats failed to parse. No formats found.
```

The example above presents the *ymd_hms()* function used for the character variable displayed as YYYY-MM-DD**T**HH:MM, without seconds. In such circumstances, the function *ymd_hm()* should be used:

```
> ymd_hm("2022-09-13T12:40")
[1] "2022-09-13 12:40:00 UTC"
```

While using the *lubridate* package, delimiters are not taken into consideration in character variables. If a datetime variable contains “T” or space between date and time it converts a character variable into a datetime object without specifying any additional arguments.

```
> ymd_hm("2022/09/13 12/40")
[1] "2022-09-13 12:40:00 UTC"
```

DATE VARIABLES

Character date variables can be converted into date objects by using several functions. How to use them depends on in which order the year, month and date appear in a character string – *dmy(), myd(), ymd(), ydm(), dym(), mdy().*

The provided example below presents usage of the *ymd()* function. It means that the order of date is as follows: year-month-date.

```
> x <- c("2021/01/27", "2021-02-24")
> ymd(x)
[1] "2021-01-27" "2021-02-24"
```

One of the advantages of the *lubridate* package is that it contains functions which can extract specific parts from a datetime variable what is similar to SAS. A date variable can be also obtained with the *date()* function performed on a datetime character variable. It should be noted that, for instance the *ymd()* function throws a warning while performing on the example below:

```
> date("2022-09-13T12:40")
[1] "2022-09-13"

> ymd("2022-09-13T12:40")
[1] NA
Warning message:
All formats failed to parse. No formats found.
```

TIME VARIABLES

Time variables can be obtained in the same way as date variables so with proper function in proper order (*hm(), ms(), hms()*). *Lubridate* does not have a function which can extract time from datetime as the *date()* function.

The example below investigates the *hms()* function. Time as a character is converted, but the class of the object is *Period* which needs to be taken into consideration if a user works with R to validate outputs prepared in SAS. Calculating time variables is easier to achieve with the *hms* package.

```
lubridate::hms("10:12:01")
[1] "10H 12M 1S"

> class(lubridate::hms("10:12:01"))
[1] "Period"
attr(,"package")
[1] "lubridate"
```

R HMS PACKAGE

Base and *lubridate* packages are widely used for calculations of datetime or date variables but have a number of limitations for calculating time variables, taking into consideration that a user is using R to validate or produce ADaM datasets. While validating variables, not only values should match, but also classes of objects. While importing sas7bdat files into R, time variables are stored mainly in *Difftime* and *Hms* classes.
There is no simple function in *base* which can produce variable with the above mentioned classes – if needed, a user can first build a *Difftime* object and then coerce its class to *Hms*. In the *lubridate* otherwise, the *hms()* function creates *Period* class, which eventually will cause difference in validating variables between SAS and R.

Efficient imputing of time variables can be accomplished with the *hms* package. It implements a class for storing and formatting time-of-day values, based on the *Difftime* class. The example below shows the function *as_hms()* from the above mentioned package.

```
> as_hms("10:12:00")
10:12:00
> class(as_hms("10:12:00"))
[1] "hms"      "difftime"
```

It should be noted that in the example above, date in character is displayed as HH:MM:SS and the *as_hms()* function throws an error on time character variable stored as HH:MM. But the *as_hms()* function can play a role of function to extract time from datetime variable already derived in ADaM.

```
> as_hms(as.POSIXct("2022-09-13 12:40:00"))
12:40:00
```

It is worth mentioning that when *lubridate* and *hms* packages are simultaneously loaded, naming conflict may occur in R session. The example above shows the usage of the *hms()* function from the *lubridate* package (*lubridate::hms*). If explicit mention of *lubridate* does not occur, it uses the *hms()* function from the *hms* package, which needs numeric values as an input and throws an error.

CLINICAL DATA CASES FOR IMPUTING TIMING VARIABLES

This part of paper focuses on two different examples of SDTM datasets and based on them ADaM datasets. The aim of this paragraph is to prepare specific ADaM variables in R with mentioned packages and functions and then perform validation of the same variables calculated in SAS. Preparation of analysis variables is separated into two approaches – the first one only uses *base* and the second uses other packages - to broadly show advantages of mixing functions from different packages for efficient R programming.

Mentioned SDTM data is divided into two datasets - sdtm1 and sdtm2 and ADaM datasets associated with them – adam1 and adam2 prepared in SAS

Before performing calculations, additional notes were made:

- Library *haven* is used to load sas7bdat files
- Library *diffdf* is used for comparing datasets.

SDTM1 – OVERVIEW OF DATASET

Dataset sdtm1 is displayed in *table 1.* It contains USUBJID variable with identifier, treatment start date TRTSDT (to avoid step with merging ADSL and needed for ASTDY calculations) and CMSTDTC/CMENDTC (start and stop date of an event). This dataset contains partial dates. It is used to perform imputation rules on clinical trial data. It was mentioned before that for actions on partial dates, rules should be written in Statistical Analysis Plan. The rules are applied as follow:

- If start day is missing – assign first day of month
- If start month and day is missing – assign 1st January
- If end day is missing – assign last day of month (28/29/30/31), depends on month and year
- If end month and day is missing – assign 31st December

Imputation flag for partial dates is evaluated as “D” if only day is missing, “M” if day and month are missing and empty if date is full or completely missing.

Calculation of ASTDY is based on ASTDT and TRTSDT conditions:

- If ASTDT is on or after TRTSDT, then relative days is difference between ASTDT and TRTSDT +1
- If ASTDT is before TRTSDT, then relative day is difference between ASTDT and TRTSDT.

Variables which needs to be calculated from this dataset are: ASTDT (Analysis Start Date), ASTDY (Analysis Start Relative Day), ASTDTF (Analysis Start Date Imputation Flag) and AENDT (Analysis End Date).

Table 1. SDTM1 dataset.

USUBJID	TRTSDT	CMSTDTC	CMENDTC
0001	2021-09-21	2020-01	
0002	2021-10-06	2018	2021-08
0003	2021-11-16	2021-03-16	2021-03-16
0004	2021-12-14	2021-09	
0005	2022-01-04	2021-10-21	2021-10-21
0006	2022-01-14	2015	
0007	2021-11-04	2016-03	

SDTM2 – OVERVIEW OF DATASET

Dataset sdtm2 is displayed in *table 2*. It contains USUBJID variable with identifier and LBDTC (datetime of specimen collection). It is used to calculate numeric datetime, time and time variables.

Variables which need to be calculated from this dataset are: ADTM (Analysis Datetime), ADT (Analysis Date), and ATM (Analysis Time).

Table 2. SDTM2 dataset.

USUBJID	LBDTC
0001	2021-09-13T16:40
0001	2021-09-21T09:20
0001	2021-09-22T10:00
0001	2021-09-24T08:38
0001	2021-10-05T08:35
0001	2021-10-26T09:07
0001	2021-11-08T09:02

ADAM1 AND ADAM2 – OVERVIEW OF DATASETS

Adam1 and adam2 are datasets prepared in SAS and are used as a base for comparison between SAS and R. They are loaded into R with the *haven* package. Adam1 dataset contains the same variables as sdtm1 dataset and analysis variables: ASTDT, ASTDTF, ASTDY, AENDT.

Analysis start and end dates, after loading into R are stored with the *Date* class. ASTDY is stored as numeric.

```
> class(adam1$ASTDT)
[1] "Date"
```

Table 3. ADAM1 dataset.

USUBJID	TRTSDT	CMSTDTC	CMENDTC	ASTDT	AENDT	ASTDTF	ASTDY
0001	2021-09-21	2020-01		2020-01-01	*NA*	D	*NA*
0002	2021-10-06	2018	2021-08	2018-01-01	2021-08-31	M	*NA*
0003	2021-11-16	2021-03-16	2021-03-16	2021-03-16	2021-03-16		-245
0004	2021-12-14	2021-09		2021-09-01	*NA*	D	*NA*
0005	2022-01-04	2021-10-21	2021-10-21	2021-10-21	2021-10-21		-75
0006	2022-01-14	2015		2015-01-01	*NA*	M	*NA*
0007	2021-11-04	2016-03		2016-03-01	*NA*	D	*NA*

Adam2 dataset contains the same variables as sdtm2 dataset and analysis variables: ADTM, ADT and ATM. Analysis Datetime is stored as the *POSIXct* and *POSIXt* classes, Analysis Date as the *Date* class and Analysis Time as the *Hms* and *Difftime* classes.

```
> class(adam2$ADTM)
[1] "POSIXct" "POSIXt"
> class(adam2$ADT)
[1] "Date"
> class(adam2$ATM)
[1] "hms"      "difftime"
```

Table 4. ADAM2 dataset.

USUBJID	LBDTC	ADTM	ADT	ATM
0001	2021-09-13T16:40	2021-09-13 16:40:00	2021-09-13	16:40:00
0001	2021-09-21T09:20	2021-09-21 09:20:00	2021-09-21	09:20:00
0001	2021-09-22T10:00	2021-09-22 10:00:00	2021-09-22	10:00:00
0001	2021-09-24T08:38	2021-09-24 08:38:00	2021-09-24	08:38:00
0001	2021-10-05T08:35	2021-10-05 08:35:00	2021-10-05	08:35:00
0001	2021-10-26T09:07	2021-10-26 09:07:00	2021-10-26	09:07:00
0001	2021-11-08T09:02	2021-11-08 09:02:00	2021-11-08	09:02:00

SDTM1 – R BASE PACKAGE

Firstly, to properly calculate analysis timing variables from sdtm1 dataset, only the *base* package is used.

- ASTDT is calculated as pasting the first day of month or the first day of January and then with usage of the *as.Date()* function
- ASTDTF is calculated based on number of characters (*nchar()*) of date in character format
- ASTDY is calculated with usage of the *difftime()* function which is used for calculating time intervals
- For AENDT, assigning the last day of month is sensitive if only the day is missing, because the last day of the month differs from month to month, and every four years for February. To assign last day of month, firstly, the first day of month is pasted (paste0(CMENDTC, "-01")) and later one month is added with the function *months()* and finally one day is subtracted.
  If CMENDTC is equal to "2021-08" above mentioned steps provide following values: "2021-08" → "2021-08-01" → "2021-09-01" → "2021-08-31".

```
phuse1Base = sdtm1 %>%
  mutate(ASTDT = as.Date(case_when(nchar(CMSTDTC) == 7 ~ paste0(CMSTDTC, "-01"),
                                   nchar(CMSTDTC) == 4 ~ paste0(CMSTDTC, "-01-01"),
                                   TRUE ~ CMSTDTC)),
         ASTDTF = case_when(nchar(CMSTDTC) == 7 ~ "D",
                            nchar(CMSTDTC) == 4 ~ "M",
                            TRUE ~ ""),
         ASTDY = case_when(ASTDTF == "" & ASTDT >= TRTSDT ~ as.numeric(difftime(ASTDT, TRTSDT, units = "days")+1),
                           ASTDTF == "" & ASTDT < TRTSDT ~ as.numeric(difftime(ASTDT, TRTSDT, units = "days")),
                           TRUE ~ as.numeric(NA)),
         AENDT = case_when(nchar(CMENDTC) == 7 ~ as.Date(paste0(CMENDTC, "-01"), format = "%Y-%m-%d") + months(1) -1,
                           nchar(CMENDTC) == 4 ~ as.Date(paste0(CMENDTC, "-12-31"), format = "%Y-%m-%d"),
                           TRUE ~ as.Date(CMENDTC, format = "%Y-%m-%d")))
```

Figure 1. Calculating phuse1Base with base R.

After the calculation, the *diffdf()* function is used to compare the datasets: adam1 (made in SAS) with phuse1Base made in R.

```
> diffdf(adam1, phuse1Base)
No issues were found!
```

SDTM1 – R LUBRIDATE PACKAGE

In the next part, only the *lubridate* package is used to prepare analysis variables.

- ASTDT is calculated as pasting the first day of month or the first day of January and then with usage of the *ymd()* function
- ASTDTF is calculated based on number of characters (*nchar()*) of date in character format
- ASTDY is calculated with usage of the *interval()* function with *days()* to display time difference in days
- To calculate properly AENDT, firstly AENDT_ is calculated. If month and day are missing, the last day of December is pasted. If only the day is missing, the same approach is implemented as with the *base* package. Firstly, the first day of month is pasted. Afterwards, the *ceiling_date()* function is used to round the date by a month and one day is subtracted.

```
phuse1Other = sdtm1 %>%
  mutate(ASTDT = ymd(case_when(nchar(CMSTDTC) == 7 ~ paste0(CMSTDTC, "-01"),
                               nchar(CMSTDTC) == 4 ~ paste0(CMSTDTC, "-01-01"),
                               TRUE ~ CMSTDTC)),
         ASTDTF = case_when(nchar(CMSTDTC) == 7 ~ "D",
                            nchar(CMSTDTC) == 4 ~ "M",
                            TRUE ~ ""),
         ASTDY = case_when(ASTDTF == "" & ASTDT >= TRTSDT ~ interval(TRTSDT, ASTDT) / days(1),
                           ASTDTF == "" & ASTDT < TRTSDT ~ interval(TRTSDT, ASTDT) / days(1),
                           TRUE ~ as.numeric(NA)),
         AENDT_ = ymd(case_when(nchar(CMENDTC) == 7 ~ paste0(CMENDTC, "-01"),
                           nchar(CMENDTC) == 4 ~ paste0(CMENDTC, "-12-31"),
                           TRUE ~ CMENDTC)),
         AENDT = case_when(nchar(CMENDTC) == 7 ~ ceiling_date(AENDT_, unit = "month")-1,
                           TRUE ~ AENDT_)) %>%
  select(-AENDT_)
```

Figure 2. Calculating phuse1Other with lubridate R.

After the calculation, the *diffdf()* function is used to compare the datasets: adam1 (made in SAS) with phuse1Other made in R.

```
> diffdf(adam1, phuse1Other)
No issues were found!
```

SDTM2 – R BASE PACKAGE

Calculating datetime and date variables in *base* can be easily accomplished, but calculation of time variables is problematic, because of the difference in classes between SAS and R. Analysis variables from sdtm2 dataset are calculated as follow.

- ADTM is calculated with the *as.POSIXct()* function, with the *format* argument including “T” character
- ADT is calculated with the *as.Date()* function. It is used directly on the character datetime variable
- ATM is calculated with *difftime()* and *strptime()* functions. It calculates the difference between two dates in seconds. Unfortunately, as a result *Difftime* class is assigned. Manual assignment of class has to be performed (class(phuse2Base$ATM) = c("hms", "difftime"))

```
phuse2Base = sdtm2 %>%
  mutate(ADTM = as.POSIXct(LBDTC, format="%Y-%m-%dT%H:%M", tz="UTC"),
         ADT = as.Date(LBDTC),
         ATM = difftime(strptime(LBDTC, "%Y-%m-%dT%H:%M") ,strptime(LBDTC, "%Y-%m-%d") , units="secs"))

class(phuse2Base$ATM) = c("hms", "difftime")
```

Figure 3. Calculating phuse2Base with base R.

After the calculation, the *diffdf()* function is used to compare the datasets: adam2 (made in SAS) with phuse2Base made in R.

```
> diffdf(adam2, phuse2Base)
No issues were found!
```

SDTM2 – R LUBRIDATE AND HMS PACKAGE

Lubridate package is made for quick calculations of timing variables. With addition of the *hms* package, it creates a versatile tool for imputation for statistical programmers. Below analysis variables are calculated with both packages.

- ADTM is calculated with the *ymd_hm()* function performed on the character datetime variable
- ADT is calculated with the *date()* function used on already imputed ADTM, which extracts the date from the datetime
- ATM is calculated with the *as_hms()* function from the *hms* package, which extract the time from the derived datetime variable.

```
phuse2Other = sdtm2 %>%
  mutate(ADTM = ymd_hm(LBDTC),
         ADT = date(ADTM),
         ATM = as_hms(ADTM))
```

Figure 4. Calculating phuse2Other with lubridate and hms R.

After the calculation, the *diffdf()* function is used to compare the datasets: adam2 (made in SAS) with phuse2Other made in R.

```
> diffdf(adam2, phuse2Other)
No issues were found!
```

SAS & R – COMPARISON OF FUNCTIONS

SAS is well-equipped to deal with timing variables and provides built-in formats that can be used to render numeric timing variables in human-readable formats. What is more, there are several functions that can be used for extracting dates and times from datetime values, which makes it easier to code.

In SAS, the easiest way to impute analysis timing variables is to calculate a datetime variable and then extract date and time with functions *datepart()* and *timepart()*. It can be also achieved with the *input()* function performed on a *DTC variable with the proper format.

These calculations are also possible with R functions. In R, the *date()* function from the *lubridate* package and the *as_hms()* function from the *hms* package can be used to extract part of derived datetime variables.
Calculating datetime in R can be mainly done with the *as.POSIXct()* function from the *base* package or *ymd_hm()/ymd_hms()* functions from the *lubridate* package– they also assigns proper classes to a variable, which is also compared with the *diffdf()* function.

Date variables can be extracted from already calculated datetime variables, so the approach is the same as in SAS, but it is also possible to calculate directly from character variables with the *date()* or the *as.Date()* function. Above mentioned functions assign the proper class. The most convenient way to calculate a time variable is to extract time from the already calculated datetime variable with the *as_hms()* function.

Table 5. Comparison of SAS and R functions.

Timing variable	SAS function	R function
Datetime	input(*DTC, is8601dt.)	as.POSIXct(*DTC, format="%Y-%m-%dT%H:%M") ymd_hm(*DTC)
Date	datepart(datetime) input(*DTC, is8601da.)	date(datetime) as.Date(*DTC) or date(*DTC)
Time	timepart(datetime) input(scan(*DTC, 2, “T”), is8601tm.)	as_hms(datetime)

CONCLUSION

Working with dates requires paying attention if a date is incomplete. In clinical trials programmers may be asked to impute a reasonable date per requirements. SAS is an acclaimed statistical software that has been used for decades by specialists but in recent years, R has gained a wide group of supporters in the pharmaceutical industry due to its potential and diversity.

This work has meticulously examined R functions from the *base* as well as additional packages: *lubridate* and *hms* which together design an immense engine for imputing timing variables in ADaM datasets. Moreover, the aim of this paper was to introduce R as an accessible validating tool in clinical trials. It has compared functions from both statistical languages for calculating timing variables and displayed examples of imputations.
The purpose of this paper was also to introduce R as a user-friendly tool for programmers who start their adventure while experiencing R in clinical trials. Learning R is an opportunity for statistical programmers to upskill and become multilingual.

REFERENCES

Study Data Tabulation Model Implementation Guide: Human Clinical Trials, Version 3.4, CDISC, 2022
Analysis Data Model Implementation Guide, Version 1.3, CDISC, 2021
https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/as.POSIX*
https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/as.Date
https://lubridate.tidyverse.org/
https://hms.tidyverse.org/

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:
Author Name: Monika Ludwińska
Company: Intego Group LLC
Address: Grzybowska 78
City / Postcode: Warsaw 00-844
Work Phone: +48 222 500 032 ext. 2583
Email: monika.ludwinska@intego-group.com
Web: https://intego-group.com/