PharmaSUG 2021 - Paper QT-139

Common Dating in R: With an example of partial date imputation

Teckla Akinyi, GlaxoSmithKline

ABSTRACT

R programming language and RStudio are increasingly being adopted in the pharmaceutical industry as part of submission packages which include ADaM data sets. It is imperative that clinical programmers upskill to handling dates within RStudio. As an open source software there is a sprawl of information on the internet on this topic covering several ways to manipulate dates in RStudio.

INTRODUCTION

This paper will consolidate information and focus on date handling using base R and the *lubridate* package. The focus will be to highlight functions useful to common transformations during the creation of ADaM data sets by a clinical programmer. This will include accessor functions used to extract parts of datetime strings, arithmetic functions used in calculating intervals and culminating in example code for date imputation within the ADaM data set ADAE. A basic understanding of packages in RStudio and the use of *tidyverse*, particularly the *magrittr* pipe and *mutate* function and an understanding of ADaM-ADAE are necessary to follow along.

DATES USING BASE R

Date variables can be represented in a data set as a numeric integer counting the days from a system start date with earlier days as negative numbers and later days as positive number; or as a character string with letters and/or numbers representing the calendar date eg 20201015 or 2020OCT15. For both, the *as.Date()* built-in function in base R can be used to appropriately identify these as date objects, however it does behave differently depending on the input. Following is the exploration on how to convert input strings when they are either numeric or character to date objects.

NUMERIC VECTORS TO DATE VARIABLES

If the variable input is numeric, the reference start date is paramount in accurately storing the dates, as different systems have different start dates for example; Microsoft's Excel reference start date is 1900-Jan-01, SAS products use 1960-Jan-01 while in R programming language it is 1970-Jan-01. The *as.Date()* function requires the reference start date to be included in the call using the origin option. Notice in the example below that the same numeric vector is translated to different dates based on the origin of the system generating the input. R will save date values (without times) as objects with the class "Date":

```
> ############Dates as numeric vector
> num_dates <- c(0, 159, 464, 15674)
> class(num_dates)
[1] "numeric"
> sas <- as.Date(num_dates, origin="1960-01-01")
> sas
[1] "1960-01-01" "1960-06-08" "1961-04-09" "2002-11-30"
> class(sas)
[1] "Date"
> R <- as.Date(num_dates, origin="1970-01-01")
> R
[1] "1970-01-01" "1970-06-09" "1971-04-10" "2012-11-30"
> class(R)
[1] "Date"
> ms_xl <- as.Date(num_dates, origin="1900-01-01")
> ms_xl
[1] "1900-01-01" "1900-06-09" "1901-04-10" "1942-12-01"
> class(ms_xl)
```

```
[1] "Date"
```

CHARACTER VECTORS TO DATE VARIABLES

If the vector input is character, the *as.Date()* function will expect a format to be specified. The following are symbols that can be used with the format option to print dates. The suggestion is to inspect the incoming vector for the order of day, month and year and use the appropriate combination of symbols to create a format. If the format is unknown a string of formats to try can be included using the *tryFormats* option. The table below offer a brief summary of symbols used in creating formats. The full list of allowed formats can be found using the command *"?strptime()":*

Table 1

Symbol	Meaning	Example
%d	Day as number (00-31)	05
%a	Abbreviated weekday	Mon
%A	Unabbreviated weekday	Monday
%m	Month (00-12)	12
%b	Abbreviated month	Jan
%B	Unabbreviated month	January
%y	2-digit year	21
%Y	4-digit year	2021

Table 1. Summary of Symbols used in Creating Date Formats

Using some of the symbols in the table above, the following snippet;

- Reads in 4 different character vectors of dates represented in different ways.
- By assigning the format of the input vector using the symbols in the table above in the order they appear, the dates are correctly parsed and output the date in the ISO 8601 international standards of YYYY-MM-DD.
- Notice, if there is a "dash" or "slash" delimiter in the input date, the format must include this.
- *as.Date()* will parse character strings upto the last one necessary for the format. Any trailing characters are ignored:

```
> ############Dates as character vector
> str_Dates1 <- c("01/05/1965", "08/16/1975","05/29/2021","03/15/2007")
> str_Dates1a <- as.Date(str_Dates1,"%m/%d/%Y")
> str_Dates1a
[1] "1965-01-05" "1975-08-16" "2021-05-29" "2007-03-15"
> class(str_Dates1a)
[1] "Date"
>
> str_Dates2 <- c("05-01-1965", "16-08-1975","29-05-2021","15-03-2007")
> str_Dates2a <- as.Date(str_Dates2,"%d-%m-%Y")
> str_Dates2a
[1] "1965-01-05" "1975-08-16" "2021-05-29" "2007-03-15"
> class(str_Dates2a)
[1] "Date"
>
> str_Dates3 <- c("01MAY1965", "16AUG1975","29MAY2021","15MAR2007")
> str_Dates3a <- as.Date(str_Dates3,"%d%b%Y")
> str_Dates3a
```

```
[1] "1965-05-01" "1975-08-16" "2021-05-29" "2007-03-15"
> class(str_Dates3a)
[1] "Date"
>
> str_Dates4 <- c("01MAY1965MORNING", "16AUGUST1975","29MAY2021","15MARCH2007ETC")
> str_Dates4a <- as.Date(str_Dates4,tryFormats = c("%d%b%Y","%d%b%Y","%Y-%b-%d"))
> str_Dates4a
[1] "1965-05-01" "1975-08-16" "2021-05-29" "2007-03-15"
> class(str_Dates4a)
[1] "Date"
```

DATE TIME FORMATS

When dates include times the classes within R are either POSIXct (ct stands for calendar time) or POSIXlt (lt stands for local time). The POSIXct class stores time with an accuracy of 1 second as the number of seconds since the origin date of January 01, 1970 midnight. POSIXlt results in a nine-element list of year, month, day, hour, minute and second. Hence it is handy in situations with the need to extract month and hour. As with the *as.Date()* function, *as.POSIXct()* and *as.POSIXlt()* expect a format option that aligns with the arrangement of the incoming vector.

For common datetime representations in clinical data is YYYY-MM-DDTHH:MM. When specifying the format, the T in between the time and date component is not a standard character for date and time, however incorporating it in the format at the position it appears allows R to interpret the code correctly, for example:

```
 > ############Date-Time
> AESTDTC <- c("2015-10-19T10:15","2015-10-19T10:15","2015-10-19T10:15","2015-10-19T10:15")
> time_date <- as.POSIXlt(AESTDTC, format="%Y-%m-%dT%H:%M")
> time_date
  [1] "2015-10-19 10:15:00 EDT" "2015-10-19 10:15:00 EDT" "2015-10-19 10:15:00 EDT" "2015-1
0-19 10:15:00 EDT"
> class(time_date)
[1] "POSIXlt" "POSIXt"
```

The resultant datetime has a time zone assigned to it. This defaults to the local time zone defined in the system computer. If the time zone of the source data is known, then this can be incorporated using the time zone option *tz*, for example:

```
> #Datetime
> AESTDTC <- c("2015-10-19T10:15","2015-10-19T10:15","2015-10-19T10:15","2015-10-19T10:15")
> time_date <- as.POSIXct(AESTDTC,tz = "America/New_York", format="%Y-%m-%dT%H:%M")
> time_date
[1] "2015-10-19 10:15:00 EDT" "2015-10-19 10:15:00 EDT" "2015-10-19 10:15:00 EDT" "2015-10-
19 10:15:00 EDT"
> class(time_date)
[1] "POSIXct" "POSIXt"
```

POSIXct and POSIXlt are very useful in converting character to strings and subsequently extracting components of the datetime string into individual date and time elements using accessor functions. An example of this need is when the character string 2015-10-19T10:15 is to be separated into individual date and time components, such as in the case of AESTDTC separated into ADT and ADTM. The code below takes in a character datetime string, converts it to POSIXlt and has the individual date and time components extracted using the formats:

```
> #######EXTRACT INDIVIDUAL ELEMENTS
> time_date <- as.POSIXlt(AESTDTC, format="%Y-%m-%dT%H:%M")
> ADTM <- format(time_date, "%H:%M")
> ADTM
[1] "10:15" "10:15" "10:15" "10:15"
>
> ADT <- format(time_date, "%Y-%m-%d")
> ADT <- as.Date(ADT)
> ADT
[1] "2015-10-19" "2015-10-19" "2015-10-19" "2015-10-19"
>
> year <- format(time_date,"%Y")
> year
[1] "2015" "2015" "2015" "2015"
>
> month <- format(time_date,"%B")
> month
[1] "October" "October" "October" "October"
>
> day <- format(time_date,"%d")
> day
[1] "19" "19" "19" "19"
```

The most common manipulation with dates in ADaM creation is in the calculation of analysis days (ADY) or analysis duration (ADUR). In this example AESTDTC and AEENDTC are used to calculate AEDUR using the function *difftime()* with units set to days:

```
> AESTDTC <- c("2015-10-19T10:15","2015-10-19T10:15","2015-10-19T10:15","2015-10-19T10:15")
> AEENDTC <- c("2018-12-19T10:15","2018-10-02T10:15","2017-05-29T10:15","2020-10-19T10:15")
>
> startdates <- as.POSIXlt(AESTDTC,usetz=F, format="%Y-%m-%dT%H:%M")
> enddates <- as.POSIXlt(AEENDTC,usetz=F, format="%Y-%m-%dT%H:%M")
>
> AEDUR <- round(as.numeric(difftime(enddates,startdates,units = "days")),digits=0)
> AEDUR
[1] 1157 1079  588 1827
```

DATES USING *LUBRIDATE* PACKAGE

All the above have been implemented in Base R which contains basic functions preloaded with the installation of R. *Lubridate* is an addon package specifically designed to handle date-time objects in R. It is a part of *tidyverse* but not part of "core" *tidyverse* and is not loaded when *tidyverse* is and thus requires explicit loading by calling "library(*lubridate*)". Otherwise it can be installed and called independently as:

install.packages("lubridate")

library(lubridate)

The *Lubridate* package offers tremendous advantage over base R in handling dates, by providing simpler arithmetic constructs in handling formats, and intervals. Providing more intuitive ways to convert characters to dates and times. For dates stored as numeric vectors, *as.Date()* function with the origin option is still necessary to convert them to the date class with the result date in the ISO 8601 international standards of YYYY-MM-DD.

However for character vectors, *Lubridate* parses strings through functions that model the order in which the year, month and day elements appear in the string. The vectors are parsed into these functions and the result is an ISO 8601 standard format date. When working with character strings the following intuitive functions in the table below support the conversion to a date class:

Table 2

Parse Strings with year, month and day components		
Function	Example call	Output
dmy()	dmy("29/05/2021")	2021-05-29

myd()	mdy("05-29-2021")	2021-05-29
ymd()	ymd("20210529")	2021-05-29
dym()	dym("292021May")	2021-05-29
yq()*	yq("2021-03")	2021-07-01
Parse Strings with hour, minute and second components		
Function	Example call	Output
hm()	hm("6,5")	6H 5M 0S
ms()	ms("6/5")	6M 5S
hms()	hms("6:5:45")	6H 5M 45S
Parse Strings with hour, minute and second components		
Function	Example call	Output
ymd_hms()	ymd_hms("2019/01/19T17:15:30")	2019-01-19 17:15:30 UTC
ymd_hm()	ymd_hm("2019/01/19T17:15")	2019-01-19 17:15:00 UTC
ymd_h()	ymd_h("2019-01-19T17")	2019-01-19 17:00:00 UTC
dmy_hms()	dmy_hms("19-01-2019T17:15:30")	2019-01-19 17:15:30 UTC
dmy_hm()	dmy_hm("19-01-2019T17:15")	2019-01-19 17:15:00 UTC
dmy_h()	dmy_h("19-01-2019T17")	2019-01-19 17:00:00 UTC
mdy_hms()	mdy_hms("01192019T17:15:30")	2019-01-19 17:15:30 UTC
mdy_hm()	mdy_hm("01192019T17:15")	2019-01-19 17:15:00 UTC
mdy_h()	mdy_h("01192019T17")	2019-01-19 17:00:00 UTC
ydm_hms()	ydm_hms("20191901T17:15:30")	2019-01-19 17:15:30 UTC
ydm_hm()	ydm_hm("20191901T17:15")	2019-01-19 17:15:00 UTC
ydm_h()	ydm_h("20191901T17")	2019-01-19 17:00:00 UTC

Table 2. Sample Functions from the *Lubridate* package useful in Converting Character Strings to Date

Note that when using *lubridate* to parse character strings, the delimiter or lack thereof, does not influence the output date. The most important thing is correct identification of the order of day, month, year and use the appropriate call. This greatly simplifies the conversion of character strings to class POSIXct.

ACCESSOR FUNCTIONS

Once the strings are converted to datetime class POSIXct, accessor functions can be used to extract individual components. Back to the example of extracting ADTM and ADT from AESTDTC. In the below snippet, the character vector is created and parsed through the accessor function *date()* to extract the date component. At the moment, *lubridate* does not have a function to extract the time component so the solution from base R is applied to extract ADTM:

```
> AESTDTC <- c("2015-10-19T10:15","2015-10-19T10:15","2015-10-19T10:15","2015-10-19T10:15")
> ADT <- date(AESTDTC)
> ADT
[1] "2015-10-19" "2015-10-19" "2015-10-19" "2015-10-19"
>
> ADTM <- format(as.POSIXlt(AESTDTC, format="%Y-%m-%dT%H:%M"), "%H:%M")
> ADTM
[1] "10:15" "10:15" "10:15" "10:15"
```

Other accessor functions that might be of interest are tabulated below. Of note are that accessor functions are named after the singular form of a time element; confusingly some period functions discussed later, will use the plural version of the units for example *second()* is an accessor that extract the second component of the date-time object, whereas *seconds()* changes an integer argument to a time component of second.

The accessor functions below are used with instants; that is to extract specific moments in time for example 04MAR2021 and timespans which are a length of time that may or may not be connected to a particular instant for example 4 days, 3 months or two and half hours. Within timespans, *lubridate* has the capability to work with time zones; however, this is beyond the scope of this paper as time zones are rarely considered within clinical programming.

This tables uses the example of AESTDTC created by the call:

AESTDTC <- ydm_hms("20191901T17:15:30")

Table 3

Function	Meaning	Example call	Output
second()	Gets second component of date-time	second(AESTDTC)	30
minute()	Gets minute component of date-time	minute(AESTDTC)	15
hour()	Gets hour component of date-time	hour(AESTDTC)	17
days()	Gets day component of date-time from system origin date-time	days(AESTDTC)	1547918130d 0H 0M 0S
yday()	Gets day component of date-time from start of that year	yday(AESTDTC)	19
mday()	Gets day component of date-time from start of that month	mday(AESTDTC)	19
wday()	Gets day component of date-time from start of that week	wday(AESTDTC)	7
month()	Gets month component of date-time	month(AESTDTC)	1
year()	Gets year component of date-time	year(AESTDTC)	2019
date()	Gets date component of date-time	date(AESTDTC)	2019-01-19
tz()	Gets timezone component of date-time	tz(AESTDTC)	UTC

Table 3. Sample Accessor Functions useful in Extracting Specific Moments in Time

TIME SPANS

Lubridate creates three timespan classes to deal with the nuances of time providing specific and consistent behavior within each class. These are intervals, periods, and duration.

1. Durations: These measure the exact amount of time that occurs between two instants. Duration objects are useful as they can be added to and subtracted from date-time objects.
2. Periods: These measure the change in clock time that occurs between two instants. These are in human-readable units such as years, months and days. These functions quickly create periods of

convenient lengths that can be added to or subtracted from dates. Do pay attention to the aforementioned note on period function names using the plural version of the unit name, whereas the earlier discussed accessor uses the singular form.

3. Intervals: These are timespans that begin at a specific instant and end at a specific instant. They retain complete information about a timespan and provide the only reliable way to convert between periods and durations.

The table below highlights functions that might be useful in determining timespans using *lubridate:*

Table 4

Durations			
Function	Meaning	Example call	Output
is.duration()	Checks if the object is a duration	is.duration(as.Date("2009-08-03"))	# FALSE
as.duration()	Changes an object to a duration	as.duration(2464510)	2464510s (~4.07 weeks)"
duration()	Creates a duration object within specified values	duration(90, "weeks")	"54432000s (~1.72 years)"
dseconds()	Creates a duration object in seconds	dseconds(x = 52)	"52s"
dminutes()	Creates a duration object in minutes	dminutes(x = 52)	"3120s (~52 minutes)"
dhours()	Creates a duration object in hows	dhours(x = 52)	"187200s (~2.17 days)"
ddays()	Creates a duration object in days	ddays(x = 52)	4492800s (~7.43 weeks)
dweeks()	Creates a duration object in weeks	dweeks(x = 52)	31449600s (~52 weeks)
dyears()	Creates a duration object in years	dyears(x = 52)	1640995200s (~52 years)
Periods			
Function	Meaning	Example call	Output
is.period()	Checks if an object is a period	is.period(5)	FALSE
as.period()	Changes an object to a period	as.period(5)	5s
period()	Changes an object to a period of specified units	Period(5,units=”weeks”)	"35d 0H 0M 0S"
seconds()	Changes an object to a period in seconds	seconds(35)	35S"
minutes()	Changes an object to a period in minutes	minutes(35)	35M 0S"
hours()	Changes an object to a period in hours	hours(35)	"35H 0M 0S"
days()	Changes an object to a period in days	days(35)	"35d 0H 0M 0S"
weeks()	Changes an object to a period in weeks	weeks(35)	"245d 0H 0M 0S"

months()	Changes an object to a period in months	months(35)	"35m 0d 0H 0M 0S"
years()	Changes an object to a period in years	years(35)	"35y 0m 0d 0H 0M 0S"
Intervals			
Function	Meaning	Example call	Output
interval()	creates an Interval object with the specified start and end dates. If the start date occurs before the end date, the interval will be positive. Otherwise, it will be negative	int <- interval(start = ydm("2017-15-03"), end = ydm("2021-29-05"))	2017-03-15 UTC--2021-05-29 UTC
is.interval()	Checks if object is an interval	is.interval(int)	TRUE
as.interval()	Converts and object to interval	as.interval(duration(days=31), ymd("2009-01-01"))	2009-01-01 UTC--2009-02-01 UTC
int_start()	Accessor for start of interval	int_start(int)	2017-03-15 UTC"
int_end()	Accessor for end of interval	int_end(int)	2021-05-29 UTC"
int_shift()	shifts the start and end dates of an interval up or down the timeline by a specified amount	int_shift(int, duration(years = -1))	2016-03-14 18:00:00 UTC--2020-05-28 18:00:00 UTC
int_flip(),	Reverses the order of start and enddate	int_flip(int)	2021-05-29 UTC--2017-03-15 UTC
int_aligns()	tests if two intervals share an endpoint	int_aligns(int1, int3)	FALSE
int_overlaps(),	tests if two intervals overlap.	int_overlaps(int1, int2)	TRUE
%within%	Checks if a date or interval 'a' falls within an interval 'b'	ymd("2001-05-03") %within% int	FALSE

Table 4. Sample Functions from *Lubridate* useful in the Determination of Time Spans

GOOD TO KNOW FUNCTIONS FROM *LUBRIDATE*

1. Rounding functions are useful in finding the closest unit. Handy functions for a clinical programmer include:
   - *floor_date()* – rounds down to the nearest unit
   - *ceiling_date()* – rounds up to the nearest unit
   - *round_date()* – rounds to the nearest unit
   - *rollback()* – rolls back to the last day of the previous month

   Examples of the rounding functions implementation are shown below:

```
> x <- ymd_hms("2009-08-03 12:01:59.23")
> floor_date(x, "month")
[1] "2009-08-01 UTC"
```

```
> ceiling_date(x, "month")
[1] "2009-09-01 UTC"
> round_date(x, "month")
[1] "2009-08-01 UTC"
> rollback(x)
[1] "2009-07-31 12:01:59 UTC"
```

2. *format_ISO8601()* is a very useful function that will take in a POSIXct datetime object and format it according to the international standard, with the precision set with substring. In the example below the precision is set to display the result through minutes:

```
> format_ISO8601(as.POSIXct("2018-02-01 03:04:05", tz="EST"), precision="ymdhm")
[1] "2018-02-01T03:04"
```

IMPUTATION EXAMPLE USING AE DATES AND DURATION CALCULATION

Partial dates are common in ADaM data sets, particularly those containing historical information such as start/end date of an adverse event in ADAE or a medication in ADCM. While preferable to be left as is in SDTM or when presented in listings, imputations to a complete date are necessary when dates are used in calculating durations. These are defined in the statistical analysis plan from which programmers implement their scripts.

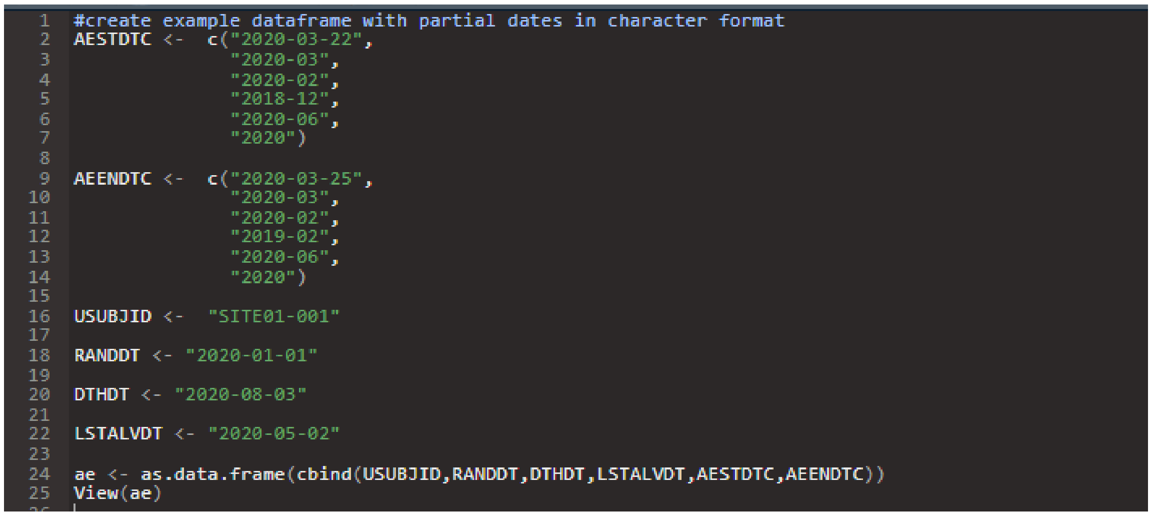
Do note that imputation rules may vary within studies and the analysis plan must be checked. Here, the following common imputation rules will be used to illustrate implementation with the *lubridate* and *tidyverse* packages.

Rules for imputation used in this example: Note, imputation rules are study dependent, do check your analysis plan for appropriate rules

- For start date: if day is missing then impute to the first day of the month. If both day and month are missing the impute to 01-Jan.
- For end date: if day is missing then impute with last day of that month and if day and month both are missing then impute with 31-Dec. If the imputed end date is after the date of death or last known date alive then set end date to death date if not missing, else set to last known date alive.
- If the year is missing for both start and end dates, then keep as is, no imputation is required if complete date is missing.

To begin a dummy data frame is created with the character variables AESTDTC, AEENDTC, USUBJIS, RANDDT, DTHDT and LASTALVDT for adverse event start date, end date, unique subject identifier, randomization, death and last known alive dates, respectively.

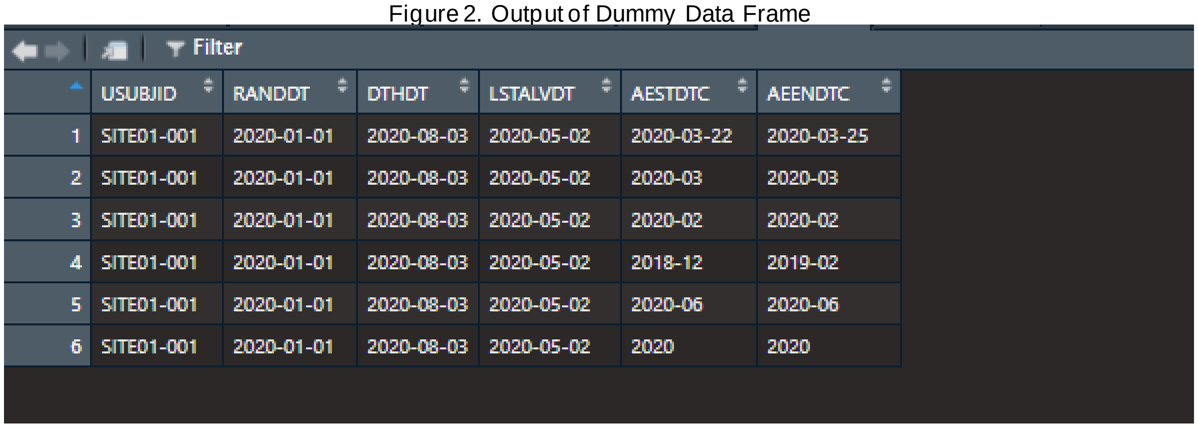
Figure 1. Creating a dummy data frame



```
#create example dataframe with partial dates in character format
AESTDTC <-  c("2020-03-22",
              "2020-03",
              "2020-02",
              "2018-12",
              "2020-06",
              "2020")

AEENDTC <-  c("2020-03-25",
              "2020-03",
              "2020-02",
              "2019-02",
              "2020-06",
              "2020")

USUBJID <-  "SITE01-001"

RANDDT <- "2020-01-01"

DTHDT <- "2020-08-03"

LSTALVDT <- "2020-05-02"

ae <- as.data.frame(cbind(USUBJID,RANDDT,DTHDT,LSTALVDT,AESTDTC,AEENDTC))
View(ae)
```

Figure 1. Creating a dummy data frame

Figure 2. Output of Dummy Data Frame



	USUBJID	RANDDT	DTHDT	LSTALVDT	AESTDTC	AEENDTC
1	SITE01-001	2020-01-01	2020-08-03	2020-05-02	2020-03-22	2020-03-25
2	SITE01-001	2020-01-01	2020-08-03	2020-05-02	2020-03	2020-03
3	SITE01-001	2020-01-01	2020-08-03	2020-05-02	2020-02	2020-02
4	SITE01-001	2020-01-01	2020-08-03	2020-05-02	2018-12	2019-02
5	SITE01-001	2020-01-01	2020-08-03	2020-05-02	2020-06	2020-06
6	SITE01-001	2020-01-01	2020-08-03	2020-05-02	2020	2020

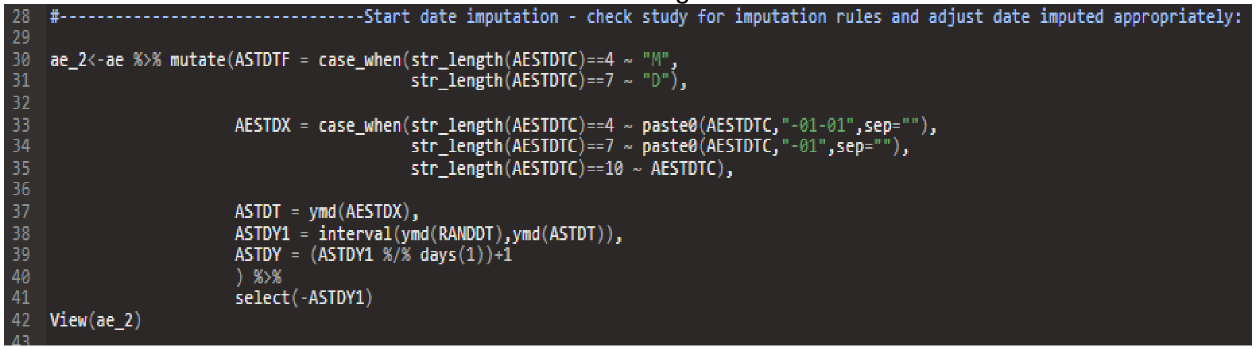
Figure 2. Output of Dummy Data Frame

START DATE IMPUTATION AND START DAY DERIVATION

This piece of code assumes start-date is sourced from SDTM and is a character variable in YYYY-MM-DD format.
The function *case_when()* works like an if statement, using the function *str_length()* to check for incomplete character start dates:

a. if month and day are missing the character length is 4 and the imputation flag is set to 'M'; following, the string "01-01" is added to the intermediate character variable AESTDX to impute to January-01.
b. if day is missing the character length is 7 and the imputation flag is set to 'D'; following, the string "-01" is added to the intermediate character variable AESTDX date to impute to the first of the month.
c. if the string length is 10 no imputation is necessary and the intermediate variable AESTDX is set to AESTDTC.
d. AESTDX should have all character dates as YYYY-MM-DD.
e. ASTDT is then formatted using the *ymd()* to YYYY-MM-DD date format.

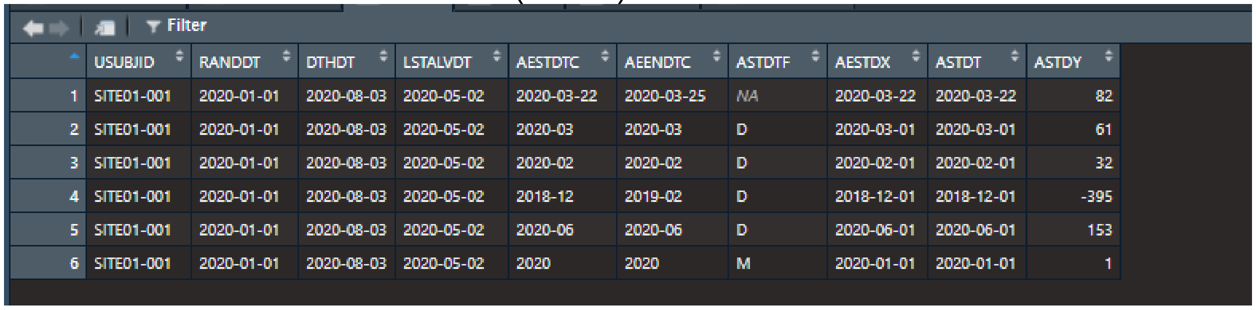
Figure 3



```
#--------------------------------Start date imputation - check study for imputation rules and adjust date imputed appropriately:

ae_2<-ae %>% mutate(ASTDTF = case_when(str_length(AESTDTC)==4 ~ "M",
                                       str_length(AESTDTC)==7 ~ "D"),

                    AESTDX = case_when(str_length(AESTDTC)==4 ~ paste0(AESTDTC,"-01-01",sep=""),
                                       str_length(AESTDTC)==7 ~ paste0(AESTDTC,"-01",sep=""),
                                       str_length(AESTDTC)==10 ~ AESTDTC),

                    ASTDT = ymd(AESTDX),
                    ASTDY1 = interval(ymd(RANDDT),ymd(ASTDT)),
                    ASTDY = (ASTDY1 %/% days(1))+1
                    ) %>%
                    select(-ASTDY1)
View(ae_2)
```

Figure 3. Start Date Imputation and Analysis Start Day Derivation Snippet

Figure 4. Output Data Frame including Imputed Analysis Start Date (ASTDT) and Analysis Start Date (ASTDY) variables



	USUBJID	RANDDT	DTHDT	LSTALVDT	AESTDTC	AEENDTC	ASTDTF	AESTDX	ASTDT	ASTDY
1	SITE01-001	2020-01-01	2020-08-03	2020-05-02	2020-03-22	2020-03-25	NA	2020-03-22	2020-03-22	82
2	SITE01-001	2020-01-01	2020-08-03	2020-05-02	2020-03	2020-03	D	2020-03-01	2020-03-01	61
3	SITE01-001	2020-01-01	2020-08-03	2020-05-02	2020-02	2020-02	D	2020-02-01	2020-02-01	32
4	SITE01-001	2020-01-01	2020-08-03	2020-05-02	2018-12	2019-02	D	2018-12-01	2018-12-01	-395
5	SITE01-001	2020-01-01	2020-08-03	2020-05-02	2020-06	2020-06	D	2020-06-01	2020-06-01	153
6	SITE01-001	2020-01-01	2020-08-03	2020-05-02	2020	2020	M	2020-01-01	2020-01-01	1

Figure 4. Output Data Frame including Imputed Analysis Start Date (ASTDT) and Analysis Start Date (ASTDY) variables

END DATE IMPUTATION AND END DAY DERIVATION

This piece of code assumes end-date is sourced from SDTM and is a character variable in YYYY-MM-DD format.

The function *case_when()* works like an if statement, using the function *str_length()* to check for incomplete character END dates:

a. if the string length is 10 no imputation is necessary and the intermediate variable AEENDX is set to AEENDTC.
b. if month and day are missing the character length is 4 and the imputation flag is set to 'M'; following, the string "12-31" is added to the intermediate character variable AEENDX to impute to December-31.
c. Since not every month has the last day as the 31st the following is applied.
   If day is missing the character length is 7 and the imputation flag is set to 'D'; following, the string "-15" is added to the intermediate character variable AEENDX date to impute to the 15th day of the month.

The 15th day is chosen here to allow all dates to parse to the function ymd() otherwise if the 31st is chosen the month of February will trigger an error since FEB 31st is not a real date.

The intermediate character variable AEENDX is parsed to *ceiling_date()* with units set as month. This rounds up all end-dates with the imputation flag "D" to the first of the following month. The function *rollback()* is used to roll these dates back to the last day of the previous month. If the imputation flag is not "D" and the flag is not missing the date is left as is.

This is particularly handy when dealing with February dates; observations 3 and 4 are illustrative of this where the partial end dates 2020-02 and 2019-02 are successfully imputed in the variable AENDT_imp to 2020-02-29 and 2019-02-28, considering that 2020 was a leap year.

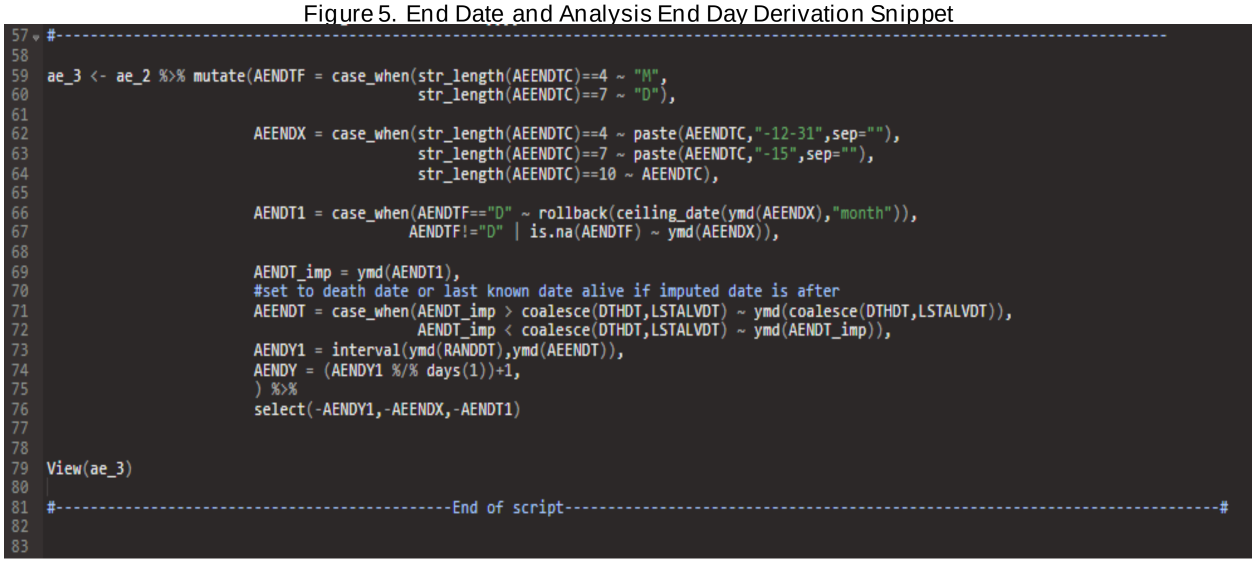
d. Finally the imputed end date in AENDT_imp is compared to death of death and last known date alive to extract the AENDT in accordance to the imputation rule, "If the imputed end date is after the date of death or last known date alive then set end date to death date if not missing, else set to last known date alive." The coalesce function finds the first non-missing date, checking the death date first.

   Observation 6 is a great example here where the imputed date (AENDT_imp) is 2020-12-31 which is greater than 2020-08-03 thus the final AEENDT is equivalent to the death date. AEENDT is then formatted using the *ymd()* function to YYYY-MM-DDdate format

e. The analysis end date is then calculated using the *interval()* function and setting the start as the Randomization date (RANDDT) and the end as the AEENDT in days.

f. All intermediate variables are dropped in the select call.

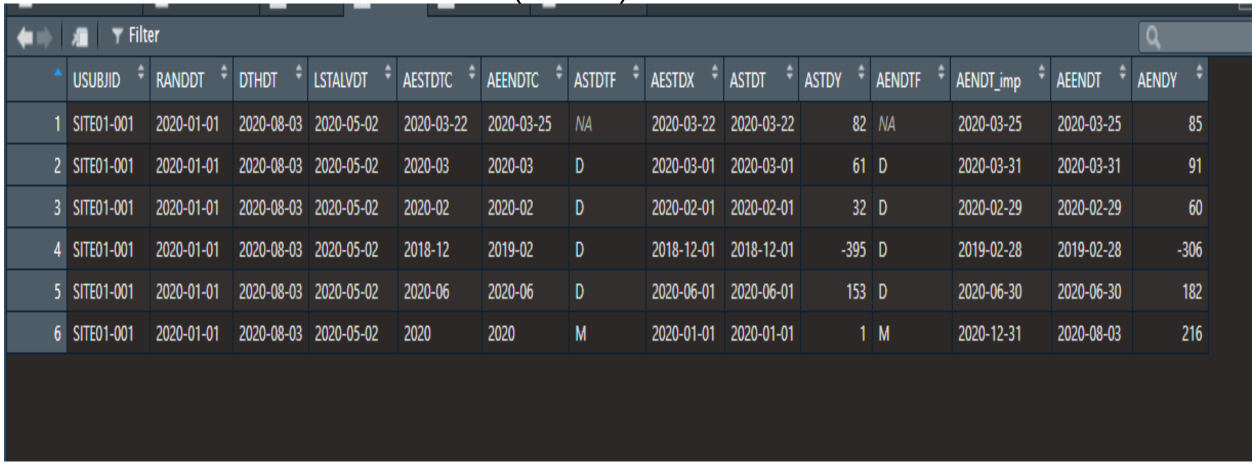
Figure 5. End Date and Analysis End Day Derivation Snippet



```
#--------------------------------------------------------------------------------------------------------------

ae_3 <- ae_2 %>% mutate(AENDTF = case_when(str_length(AEENDTC)==4 ~ "M",
                                           str_length(AEENDTC)==7 ~ "D"),

                        AEENDX = case_when(str_length(AEENDTC)==4 ~ paste(AEENDTC,"-12-31",sep=""),
                                           str_length(AEENDTC)==7 ~ paste(AEENDTC,"-15",sep=""),
                                           str_length(AEENDTC)==10 ~ AEENDTC),

                        AENDT1 = case_when(AENDTF=="D" ~ rollback(ceiling_date(ymd(AEENDX),"month")),
                                          AENDTF!="D" | is.na(AENDTF) ~ ymd(AEENDX)),

                        AENDT_imp = ymd(AENDT1),
                        #set to death date or last known date alive if imputed date is after
                        AEENDT = case_when(AENDT_imp > coalesce(DTHDT,LSTALVDT) ~ ymd(coalesce(DTHDT,LSTALVDT)),
                                           AENDT_imp < coalesce(DTHDT,LSTALVDT) ~ ymd(AENDT_imp)),
                        AENDY1 = interval(ymd(RANDDT),ymd(AEENDT)),
                        AENDY = (AENDY1 %/% days(1))+1,
                        ) %>%
                        select(-AENDY1,-AEENDX,-AENDT1)


View(ae_3)

#---------------------------------------------End of script------------------------------------------------------------#
```

Figure 5. End Date and Analysis End Day Derivation Snippet

Figure 6. Output Data Frame including Imputed Analysis End Date (AEEND) and Analysis End Day (AENDY) variables



	USUBJID	RANDDT	DTHDT	LSTALVDT	AESTDTC	AEENDTC	ASTDTF	AESTDX	ASTDT	ASTDY	AENDTF	AENDT_imp	AEENDT	AENDY
1	SITE01-001	2020-01-01	2020-08-03	2020-05-02	2020-03-22	2020-03-25	NA	2020-03-22	2020-03-22	82	NA	2020-03-25	2020-03-25	85
2	SITE01-001	2020-01-01	2020-08-03	2020-05-02	2020-03	2020-03	D	2020-03-01	2020-03-01	61	D	2020-03-31	2020-03-31	91
3	SITE01-001	2020-01-01	2020-08-03	2020-05-02	2020-02	2020-02	D	2020-02-01	2020-02-01	32	D	2020-02-29	2020-02-29	60
4	SITE01-001	2020-01-01	2020-08-03	2020-05-02	2018-12	2019-02	D	2018-12-01	2018-12-01	-395	D	2019-02-28	2019-02-28	-306
5	SITE01-001	2020-01-01	2020-08-03	2020-05-02	2020-06	2020-06	D	2020-06-01	2020-06-01	153	D	2020-06-30	2020-06-30	182
6	SITE01-001	2020-01-01	2020-08-03	2020-05-02	2020	2020	M	2020-01-01	2020-01-01	1	M	2020-12-31	2020-08-03	216

Figure 6. Output Data Frame including Imputed Analysis End Date (AEEND) and Analysis End Day (AENDY) variables

CONCLUSION

A basic introduction to dates in R has been presented, exploring reading in character and numeric vectors as dates. Base R, albeit accurate, lags in simplicity to the *lubridate* functions presented herein. By no means is this paper exhaustive in its exploration of functions and their options and further exploration is encouraged by use of the *lubridate* cheat sheet on Rstudio's website and Hadley Wickham's paper on the Journal of statistical software website, both provided as recommended readings here. The imputation example can be wrapped up in a function, that operates akin to a function in SAS but that is beyond the beginner's scope intended for this paper.

RECOMMENDED READING

- Grolemund, Garrett, and Hadley Wickham. 2011. "Dates and times made easy with lubridate." *Journal of statistical software,* 40.3:1-25.
- "RStudio Cheatsheets." *Rstudio*. Accessed March 26, 2021. https://www.rstudio.com/resources/cheatsheets/

CONTACT INFORMATION

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

Teckla Akinyi
GlaxoSmithKline
1250 S. Collegeville Road
Collegeville, Pennsylvania, US, 19426-0989
Email: teckla.g.akinyi@gsk.com