Working with dates and time in R using the lubridate package

Sometimes we have data with dates and/or times that we want to manipulate or summarize. A common example in the health sciences is time-in-study. A subject may enter a study on Feb 12, 2008 and exit on November 4, 2009. How many days was the person in the study? (Don't forget 2008 was a leap year; February had 29 days.) What was the median time-in-study for all subjects?

Another example are experiments that time participants performing an activity, applies a treatment to certain members, and then re-times the activity. What was the difference in times between subjects that received the treatment and those that did not? If our data is stored and read in as something like "01:23:03", then we'll need to convert to seconds.

The lubridate package for the R statistical computing environment was designed to help us deal with these kinds of data. The out-of-the-box base R installation also provides functions for working with dates and times, but the functions in the lubridate package are a little easier to use and remember.

Formatting dates

When we import data into R, dates and times are usually stored as character or factor by default due to symbols such as "-", ":" and "/". (Though see the readr package for functions that attempt to parse date and times automatically.) Using the `str` or `class` functions will tell you how they're stored. If dates or times are stored as character or factor that means we can't calculate or summarize elapsed times.

To format dates, lubridate provides a series of functions that are a permutation of the letters "m", "d" and "y" to represent the ordering of month, day and year. For example, if our data has a column of dates such as May 11, 1996, our dates are ordered month-day-year. Therefore we would use the `mdy` function to transform the column to a date object. If our dates were in the order of, say, year-month-day, we would use the `ymd` function. lubridate provides functions for every permutation of "m", "d", "y".

Let's demonstrate. Below we generate two character vectors of dates, inspect their class, reformat them using the `mdy` function and then inspect their class again.

```
library(lubridate)
begin <- c("May 11, 1996", "September 12, 2001", "July 1, 1988")
end <- c("7/8/97","10/23/02","1/4/91")
class(begin)

## [1] "character"

class(end)

## [1] "character"
```

```
(begin <- mdy(begin))
```

```
## [1] "1996-05-11" "2001-09-12" "1988-07-01"
```

```
(end <- mdy(end))
```

```
## [1] "1997-07-08" "2002-10-23" "1991-01-04"
```

```
class(begin)
```

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

```
class(end)
```

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

The dates now have class “Date” and are printed in year-month-day format. They may appear to still be character data when printed, but they are in fact numbers. The “Date” class means dates are stored as the number of days since January 1, 1970, with negative values for earlier dates. We can use the `as.numeric` function to view the raw values.

```
as.numeric(begin)
```

```
## [1]  9627 11577  6756
```

```
as.numeric(end)
```

```
## [1] 10050 11983  7673
```

With dates stored in this fashion we can do things like subtraction to calculate number of days between two dates.

We can also format dates that contain time information by appending `_h` , `_hm` , or `_hms` to any of the aforementioned functions. “h”, “m”, and “s” stand for hour, minute, and second, respectively. Below we add some time data to our dates and demonstrate how to use `mdy_hms` .

```
begin <- c("May 11, 1996 12:05", "September 12, 2001 1:00", "July 1, 1988 3:32")
end <- c("7/8/97 8:00","10/23/02: 12:00","1/4/91 2:05")
(begin <- mdy_hm(begin))
```

```
## [1] "1996-05-11 12:05:00 UTC" "2001-09-12 01:00:00 UTC"
## [3] "1988-07-01 03:32:00 UTC"
```

```
(end <- mdy_hm(end))
```

```
## [1] "1997-07-08 08:00:00 UTC" "2002-10-23 12:00:00 UTC"
## [3] "1991-01-04 02:05:00 UTC"
```

```
class(begin)
```

```
## [1] "POSIXct" "POSIXt"
```

```
class(end)
```

```
## [1] "POSIXct" "POSIXt"
```

Notice the class is now “POSIXct”. “POSIXct” represents the number of seconds since the beginning of 1970. If a date is before 1970, the number of seconds is negative. Notice also the the letters “UTC” have been appended to the date-times. UTC is short for Universal Coordinated Time. You can read more about UTC here, but it’s basically the time standard by which the world regulates clocks. If we prefer we can specify a time zone when formatting dates by using the tz argument. Here’s how we can specify the Eastern Time Zone in the United States when formatting our dates.

```
begin <- c("May 11, 1996 12:05", "September 12, 2001 1:00", "July 1, 1988 3:32")
end <- c("7/8/97 8:00","10/23/02: 12:00","1/4/91 2:05")
(begin <- mdy_hm(begin, tz = "US/Eastern"))
```

```
## [1] "1996-05-11 12:05:00 EDT" "2001-09-12 01:00:00 EDT"
## [3] "1988-07-01 03:32:00 EDT"
```

```
(end <- mdy_hm(end, tz = "US/Eastern"))
```

```
## [1] "1997-07-08 08:00:00 EDT" "2002-10-23 12:00:00 EDT"
## [3] "1991-01-04 02:05:00 EST"
```

Notice the last date is EST instead of EDT. EST means “Eastern Standard Time”. EDT means “Eastern Daylight Time”. Any day and time that falls during Daylight Savings is EDT. Otherwise it’s EST. How do we know the appropriate time zone phrase to use in the tz argument? We can use the `OlsonNames` function to see a character vector of all time zone names. Just enter `OlsonNames()` in the R console and hit Enter.

We can also read in times without dates using the functions `ms`, `hm`, or `hms`, where again “h”, “m”, and “s” stand for “hours”, “minutes”, and “seconds”. Here are a few examples.

```
time1 <- c("1:13", "0:58", "1:01")
time2 <- c("12:23:11", "09:45:31", "12:05:22")
time3 <- c("2:14", "2:16", "3:35")
```

```
(time1 <- ms(time1))
```

```
## [1] "1M 13S" "58S"    "1M 1S"
```

```
(time2 <- hms(time2))
```

```
## [1] "12H 23M 11S" "9H 45M 31S"  "12H 5M 22S"
```

```
(time3 <- hm(time3))
```

```
## [1] "2H 14M 0S" "2H 16M 0S" "3H 35M 0S"
```

Once again, don’t be fooled by the print out. These times are actually stored as seconds. Use `as.numeric` to verify.

```
as.numeric(time1)
```

```
## [1] 73 58 61
```

```
as.numeric(time2)
```

```
## [1] 44591 35131 43522
```

```
as.numeric(time3)
```

```
## [1]  8040  8160 12900
```

The class of these new time objects is neither “Date” nor “POSIX” but rather “Period”.

```
class(time1)
```

```
## [1] "Period"
## attr(,"package")
## [1] "lubridate"
```

Period is one of three classes lubridate provides for time spans. Let’s learn more about these classes.

Durations, Intervals and Periods

lubridate provides three classes, or three different ways, to distinguish between different types of time spans.

1. Duration
2. Interval
3. Period

Understanding these classes will help you get the most out of lubridate.

The most simple is Duration. This is simply a span of time measured in seconds. There is no start date.

An Interval is also measured in seconds but has an associated start date. An Interval measures elapsed seconds between two specific points in time.

A Period records a time span in units larger than seconds, such as years or months. Unlike seconds, years and months differ in time. June has 30 days while July has 31 days. February has 28 days except for leap years when it has 29 days. With the Period class, we can add 1 month to February 1 and get March 1. It allows us to perform calculations in calendar or clock time as opposed to absolute number of seconds.

Let’s see these three classes in action. Below we define two dates in the US Eastern time zone. The start day is March 11, 2017 at 5:21 AM. The end day is March 12, 2017 at the same time. Note that Daylight Savings begins (or began, depending on when you’re reading this) on March 12 at 2:00 AM.

```
start <- mdy_hm("3-11-2017 5:21", tz = "US/Eastern")
end <- mdy_hm("3-12-2017 5:21", tz = "US/Eastern")
```

Since we’re dealing with elapsed time between two dates, let’s start with Intervals. We can define an Interval using the `%--%` operator.

```
time.interval <- start %--% end
time.interval
```

```
## [1] 2017-03-11 05:21:00 EST--2017-03-12 05:21:00 EDT
```

Notice how Intervals print. They show the beginng date and end date. And also notice how the time zone changes from EST to EDT indicating that Daylight Savings has started. If we look at the structure of an Interval object we see it contains elapsed time in seconds, 82800, and the start date.

```
str(time.interval)
```

```
## Formal class 'Interval' [package "lubridate"] with 3 slots
##   ..@ .Data: num 82800
##   ..@ start: POSIXct[1:1], format: "2017-03-11 05:21:00"
##   ..@ tzone: chr "US/Eastern"
```

To create a Duration between these two dates, we can use the `as.duration` function.

```
time.duration <- as.duration(time.interval)
time.duration
```

```
## [1] "82800s (~23 hours)"
```

Notice a Duration object prints the elapsed time in seconds as well as something a little friendlier to read, in this case hours. Because Daylight Savings went into effect at 2:00 AM during the interval, an hour was skipped. Thus the duration between these two time points is only 23 hours.

If we look at the structure of a Duration object we see it just contains elapsed time in seconds.

```
str(time.duration)
```

```
## Formal class 'Duration' [package "lubridate"] with 1 slot
##   ..@ .Data: num 82800
```

We can create a Period from an Interval using the `as.period` function.

```
time.period <- as.period(time.interval)
time.period
```

```
## [1] "1d 0H 0M 0S"
```

A Period prints elapsed time as integers in the form of years, months, weeks, days and so on. Notice this Period is 1 day long. While only 23 hours have technically elapsed since the start date, according to our clock one day has elapsed.

If we look at the structure we see a Period contains several slots for “clock time” values and, like the Duration object, no associated date.

```
str(time.period)
```

```
## Formal class 'Period' [package "lubridate"] with 6 slots
##   ..@ .Data : num 0
##   ..@ year  : int 0
##   ..@ month : int 0
##   ..@ day   : int 1
##   ..@ hour  : int 0
##   ..@ minute: int 0
```

To recap:

- An Interval is elapsed time in seconds between two specific dates. (If no time is provided, the time for each date is assumed to be 00:00:00, or midnight.)
- A Duration is elapsed time in seconds independent of a start date.
- A Period is elapsed time in “calendar” or “clock” time (4 weeks, 2 months, etc) independent of a start date.

Calculations and conversions

Once we format dates and define our time span we often want to do some calculations and conversions. For example, we may want to calculate the mean elapsed time in weeks for different groups.

Let’s create some data and demonstrate. First we enter arbitrary start and end dates and define an Interval

```
start <- c("2012-08-21", "2012-09-01", "2012-08-15", "2012-09-18")
end <- c("2012-09-16", "2012-09-06", "2012-08-22", "2012-10-11")
elapsed.time <- start %--% end
```

If we view the `elapsed.time` object we’ll just see date ranges. We can use `as.duration` or even `as.numeric` to view the elapsed time in seconds but that’s not very useful in this case. It would be better if we converted seconds to another unit of time such as weeks or days. Fortunately lubridate makes this easy.

The trick is to convert intervals to durations and then divide the duration by a duration object in the units we desire. That’s a mouthful but easy to demonstrate. Below we demonstrate how to convert to weeks. First we convert our interval to a duration, and then we divide by `dweeks(1)` . The function call `dweeks(1)` generates a duration of one week in seconds, which is 604800. Dividing that into our duration returns number of weeks.

```
as.duration(elapsed.time) / dweeks(1)
```

```
## [1] 3.7142857 0.7142857 1.0000000 3.2857143
```

We can do the same with hours, days, minutes and years.

```
as.duration(elapsed.time) / dhours(1)
```

```
## [1] 624 120 168 552
```

```
as.duration(elapsed.time) / ddays(1)
```

```
## [1] 26  5  7 23
```

```
as.duration(elapsed.time) / dminutes(1)
```

```
## [1] 37440  7200 10080 33120
```

```
as.duration(elapsed.time) / dyears(1)
```

```
## [1] 0.07123288 0.01369863 0.01917808 0.06301370
```

Once we have the durations in the units we want, we can then do things like find the mean.

```
mean(as.duration(elapsed.time) / dweeks(1))
```

```
## [1] 2.178571
```

Of course this was just for demonstration. With only 4 values, the mean is not a very useful summary.

As another example, consider the following vector of character data summarizing a duration of time. “12w” means 12 weeks and “4d” means 4 days.

```
StudyTime <- c("12w 4d", "11w", "10w 5d", NA, "12w 6d")
```

What if we wanted to convert that to numeric weeks? First we’ll give the R code and them explain how it works.

```
as.duration(period(StudyTime, units = c("week","day"))) / dweeks(1)
```

```
## [1] 12.57143 11.00000 10.71429       NA 12.85714
```

First we use the `period` function to define a Period using our data. The units argument says the first part of our data represents weeks and the second part represents days. That is then converted to a Duration object that stores time in seconds. Finally we divide by `dweeks(1)` to convert seconds to weeks. Notice how the NA remains NA and that “11w” converts to 11 just fine even though it had no days appended to it.

There is much more to the lubridate package. Read the vignette and check out the examples on each function’s help page. But hopefully the material in this post gets you started with reading in dates, creating time-spans, and making conversions and calculations.

For questions or clarifications regarding this article, contact the UVa Library StatLab: statlab@virginia.edu

Clay Ford
Statistical Research Consultant
University of Virginia Library
January 11, 2017