

Dates and Times with lubridate

Raphael Carvalho

15/07/2019

Unfortunately, due to different date and time representations, this lesson is only guaranteed to work with an “en_US.UTF-8” locale. To view your locale, type `Sys.getlocale(“LC_TIME”)`.

```
Sys.getlocale("LC_TIME")
```

```
## [1] "pt_BR.UTF-8"
```

The `today()` function returns today’s date. Give it a try, storing the result in a new variable called `this_day`.

```
this_day <- today()
this_day
```

```
## [1] "2019-07-15"
```

There are three components to this date. In order, they are year, month, and day. We can extract any of these components using the `year()`, `month()`, or `day()` function, respectively. Try any of those on `this_day` now.

```
month(this_day)
```

```
## [1] 7
```

We can also get the day of the week from `this_day` using the `wday()` function. It will be represented as a number, such that 1 = Sunday, 2 = Monday, 3 = Tuesday, etc. Give it a shot.

```
wday(this_day)
```

```
## [1] 2
```

Now try the same thing again, except this time add a second argument, `label = TRUE`, to display the *name* of the weekday (represented as an ordered factor).

```
wday(this_day, label = TRUE)
```

```
## [1] Seg
```

```
## Levels: Dom < Seg < Ter < Qua < Qui < Sex < Sáb
```

In addition to handling dates, lubridate is great for working with date and time combinations, referred to as date-times. The `now()` function returns the date-time representing this exact moment in time. Give it a try and store the result in a variable called `this_moment`.

```
this_moment <- now()
this_moment
```

```
## [1] "2019-07-15 18:56:40 -03"
```

Just like with dates, we can extract the year, month, day, or day of week. However, we can also use `hour()`, `minute()`, and `second()` to extract specific time information. Try any of these three new functions now to extract one piece of time information from `this_moment`.

```
hour(this_moment)
```

```
## [1] 18
```

Fortunately, lubridate offers a variety of functions for parsing date-times. These functions take the form of ymd(), dmy(), hms(), ymd_hms(), etc., where each letter in the name of the function stands for the location of years (y), months (m), days (d), hours (h), minutes (m), and/or seconds (s) in the date-time being read in.

To see how these functions work, try ymd("1989-05-17"). You must surround the date with quotes. Store the result in a variable called my_date.

```
my_date <- ymd("1989-05-17")
my_date
```

```
## [1] "1989-05-17"
```

"1989-05-17" is a fairly standard format, but lubridate is 'smart' enough to figure out many different date-time formats. Use ymd() to parse "1989 May 17". Don't forget to put quotes around the date!

```
ymd("1989 May 17")
```

```
## [1] "1989-05-17"
```

Despite being formatted differently, the last two dates had the year first, then the month, then the day. Hence, we used ymd() to parse them. What do you think the appropriate function is for parsing "March 12, 1975"? Give it a try.

```
mdy("March 12, 1975")
```

```
## [1] "1975-03-12"
```

We can even throw something funky at it and lubridate will often know the right thing to do. Parse 25081985, which is supposed to represent the 25th day of August 1985. Note that we are actually parsing a numeric value here – not a character string – so leave off the quotes.

```
dmy(25081985)
```

```
## [1] "1985-08-25"
```

But be careful, it's not magic. Try ymd("192012") to see what happens when we give it something more ambiguous. Surround the number with quotes again, just to be consistent with the way most dates are represented (as character strings).

```
ymd("192012")
```

```
## Warning: All formats failed to parse. No formats found.
```

```
## [1] NA
```

You got a warning message because it was unclear what date you wanted. When in doubt, it's best to be more explicit. Repeat the same command, but add two dashes OR two forward slashes to "192012" so that it's clear we want January 2, 1920.

```
ymd("1920/1/2")
```

```
## [1] "1920-01-02"
```

In addition to dates, we can parse date-times. I've created a date-time object called dt1. Parse dt1 with ymd_hms().

```
ymd_hms(dt1)
```

```
## Error in lapply(list(...), .num_to_date): objeto 'dt1' não encontrado
```

What if we have a time, but no date? Use the appropriate lubridate function to parse "03:22:14" (hh:mm:ss).

```
hms("03:22:14")
```

```
## [1] "3H 22M 14S"
```

lubridate is also capable of handling vectors of dates, which is particularly helpful when you need to parse an entire column of data. I've created a vector of dates called `dt2`.

```
ymd(dt2)
```

```
## Error in lapply(list(...), .num_to_date): objeto 'dt2' não encontrado
```

The `update()` function allows us to update one or more components of a date-time. For example, let's say the current time is 08:34:55 (hh:mm:ss). Update `this_moment` to the new time using the following command: `update(this_moment, hours = 8, minutes = 34, seconds = 55)`.

```
update(this_moment, hours = 8, minutes = 34, seconds = 55)
```

```
## [1] "2019-07-15 08:34:55 -03"
```

It's important to recognize that the previous command does not alter `this_moment` unless we reassign the result to `this_moment`. To see this, print the contents of `this_moment`.

```
this_moment
```

```
## [1] "2019-07-15 18:56:40 -03"
```

Unless you're a superhero, some time has passed since you first created `this_moment`. Use `update()` to make it match the current time, specifying at least hours and minutes. Assign the result to `this_moment`, so that `this_moment` will contain the new time.

```
this_moment <- update(this_moment, hours = 18, minutes = 46)
this_moment
```

```
## [1] "2019-07-15 18:46:40 -03"
```

Now, pretend you are in New York City and you are planning to visit a friend in Hong Kong. You seem to have misplaced your itinerary, but you know that your flight departs New York at 17:34 (5:34pm) the day after tomorrow. You also know that your flight is scheduled to arrive in Hong Kong exactly 15 hours and 50 minutes after departure.

Let's reconstruct your itinerary from what you can remember, starting with the full date and time of your departure. We will approach this by finding the current date in New York, adding 2 full days, then setting the time to 17:34.

To find the current date in New York, we'll use the `now()` function again. This time, however, we'll specify the time zone that we want: "America/New_York". Store the result in a variable called `nyc`. Check out `?now` if you need help.

```
nyc <- now(tzone = "America/New_York")
nyc
```

```
## [1] "2019-07-15 17:56:40 EDT"
```

Your flight is the day after tomorrow (in New York time), so we want to add two days to `nyc`. One nice aspect of lubridate is that it allows you to use arithmetic operators on dates and times. In this case, we'd like to add two days to `nyc`, so we can use the following expression: `nyc + days(2)`. Give it a try, storing the result in a variable called `depart`.

```
depart <- nyc + days(2)
depart
```

```
## [1] "2019-07-17 17:56:40 EDT"
```

So now `depart` contains the date of the day after tomorrow. Use `update()` to add the correct hours (17) and minutes (34) to `depart`. Reassign the result to `depart`.

```
depart <- update(depart, hours = 17, minutes = 34)
depart
```

```
## [1] "2019-07-17 17:34:40 EDT"
```

Your friend wants to know what time she should pick you up from the airport in Hong Kong. Now that we have the exact date and time of your departure from New York, we can figure out the exact time of your arrival in Hong Kong.

The first step is to add 15 hours and 50 minutes to your departure time. Recall that `nyc + days(2)` added two days to the current time in New York. Use the same approach to add 15 hours and 50 minutes to the date-time stored in `depart`. Store the result in a new variable called `arrive`.

```
arrive <- depart + hours(15) + minutes(50)
arrive
```

```
## [1] "2019-07-18 09:24:40 EDT"
```

The `arrive` variable contains the time that it will be in New York when you arrive in Hong Kong. What we really want to know is what time it will be in Hong Kong when you arrive, so that your friend knows when to meet you.

The `with_tz()` function returns a date-time as it would appear in another time zone. Use `with_tz()` to convert `arrive` to the “Asia/Hong_Kong” time zone. Reassign the result to `arrive`, so that it will get the new value.

```
arrive <- with_tz(arrive, "Asia/Hong_Kong")
arrive
```

```
## [1] "2019-07-18 21:24:40 HKT"
```

Fast forward to your arrival in Hong Kong. You and your friend have just met at the airport and you realize that the last time you were together was in Singapore on June 17, 2008. Naturally, you’d like to know exactly how long it has been.

Use the appropriate lubridate function to parse “June 17, 2008”, just like you did near the beginning of this lesson. This time, however, you should specify an extra argument, `tz = “Singapore”`. Store the result in a variable called `last_time`.

```
last_time <- mdy("June 17, 2008", tz = "Singapore")
last_time
```

```
## [1] "2008-06-17 +08"
```

Create an `interval()` that spans from `last_time` to `arrive`. Store it in a new variable called `how_long`.

```
how_long <- interval(last_time, arrive)
how_long
```

```
## [1] 2008-06-17 +08--2019-07-18 21:24:40 +08
```

Now use `as.period(how_long)` to see how long it’s been.

```
as.period(how_long)
```

```
## [1] "11y 1m 1d 21H 24M 40S"
```