| Please choose a lesson, or type 0 to return to course menu.

1: Manipulating Data with dplyr

2: Grouping and Chaining with dplyr

3: Tidying Data with tidyr

4: Dates and Times with lubridate

Selection: 4

| Attemping to load lesson dependencies...

| This lesson requires the ‘lubridate’ package. Would you like me to install it for you now?

1: Yes

2: No

Selection: 1

| Trying to install package ‘lubridate’ now...

package ‘lubridate’ successfully unpacked and MD5 sums checked

| Package ‘lubridate’ loaded correctly!

| | 0%

| In this lesson, we'll explore the lubridate R package, by Garrett Grolemund and Hadley Wickham.

| According to the package authors, "lubridate has a consistent, memorable syntax, that makes working

| with dates fun instead of frustrating." If you've ever worked with dates in R, that statement

| probably has your attention.

...

|== | 2%

| 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] "Spanish\_Spain.1252"

| You got it right!

|=== | 3%

| If the output above is not "en\_US.UTF-8", this lesson is not guaranteed to work correctly. Of

| course, you are welcome to try it anyway. We apologize for this inconvenience.

...

|===== | 5%

| lubridate was automatically installed (if necessary) and loaded upon starting this lesson. To build

| the habit, we'll go ahead and (re)load the package now. Type library(lubridate) to do so.

> library(lubridate)

| That's the answer I was looking for.

|====== | 7%

| lubridate contains many useful functions. We'll only be covering the basics here. Type help(package

| = lubridate) to bring up an overview of the package, including the package DESCRIPTION, a list of

| available functions, and a link to the official package vignette.

> help(package = lubridate)

| That's the answer I was looking for.

|======== | 9%

| Let's get going!

...

|========== | 10%

| The today() function returns today's date. Give it a try, storing the result in a new variable

| called this\_day.

> this\_day <- today()

| You nailed it! Good job!

|=========== | 12%

| Print the contents of this\_day to the console.

> this\_day

[1] "2016-02-07"

| That's the answer I was looking for.

|============= | 14%

| 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.

> day(this\_day)

[1] 7

| You're the best!

|============== | 16%

| 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] 1

| You are doing so well!

|================ | 17%

| 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] Sun

Levels: Sun < Mon < Tues < Wed < Thurs < Fri < Sat

| You are quite good my friend!

|================= | 19%

| 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()

| You got it right!

|=================== | 21%

| View the contents of this\_moment.

> this\_moment

[1] "2016-02-07 09:59:35 CET"

| You nailed it! Good job!

|===================== | 22%

| 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.

> second(this\_moment)

[1] 35.09592

| You are really on a roll!

|====================== | 24%

| today() and now() provide neatly formatted date-time information. When working with dates and times

| 'in the wild', this won't always (and perhaps rarely will) be the case.

...

|======================== | 26%

| 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.

...

|========================= | 28%

| 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")

| That's a job well done!

|=========================== | 29%

| Now take a look at my\_date.

> my\_date

[1] "1989-05-17 UTC"

| You are amazing!

|============================= | 31%

| It looks almost the same, except for the addition of a time zone, which we'll discuss later in the

| lesson. Below the surface, there's another important change that takes place when lubridate parses

| a date. Type class(my\_date) to see what that is.

> class(my\_date)

[1] "POSIXct" "POSIXt"

| All that practice is paying off!

|============================== | 33%

| So ymd() took a character string as input and returned an object of class POSIXct. It's not

| necessary that you understand what POSIXct is, but just know that it is one way that R stores

| date-time information internally.

...

|================================ | 34%

| "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 UTC"

| You are doing so well!

|================================= | 36%

| 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] "1975-12-19 UTC"

| You are quite good my friend!

|=================================== | 38%

| 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 UTC"

| Your dedication is inspiring!

|==================================== | 40%

| 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")

[1] NA

Warning message:

All formats failed to parse. No formats found.

| All that practice is paying off!

|====================================== | 41%

| 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 UTC"

| You nailed it! Good job!

|======================================== | 43%

| In addition to dates, we can parse date-times. I've created a date-time object called dt1. Take a

| look at it now.

> dt1

[1] "2014-08-23 17:23:02"

| You are quite good my friend!

|========================================= | 45%

| Now parse dt1 with ymd\_hms().

> ymd\_hms(dt1)

[1] "2014-08-23 17:23:02 UTC"

| That's the answer I was looking for.

|=========================================== | 47%

| 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"

| That's correct!

|============================================ | 48%

| 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. View its contents

| now.

> dt2

[1] "2014-05-14" "2014-09-22" "2014-07-11"

| Great job!

|============================================== | 50%

| Now parse dt2 using the appropriate lubridate function.

> ymd(dt2)

[1] "2014-05-14 UTC" "2014-09-22 UTC" "2014-07-11 UTC"

| You are quite good my friend!

|================================================ | 52%

| 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] "2016-02-07 08:34:55 CET"

| You are quite good my friend!

|================================================= | 53%

| 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] "2016-02-07 09:59:35 CET"

| You nailed it! Good job!

|=================================================== | 55%

| 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 = 10, minutes = 50)

| You got it!

|==================================================== | 57%

| Take one more look at this\_moment to see that it's been updated.

> this\_moment

[1] "2016-02-07 10:50:35 CET"

| Perseverance, that's the answer.

|====================================================== | 59%

| 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.

...

|======================================================== | 60%

| 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.

...

|========================================================= | 62%

| 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")

| You got it!

|=========================================================== | 64%

| For a complete list of valid time zones for use with lubridate, check out the following Wikipedia

| page:

|

| http://en.wikipedia.org/wiki/List\_of\_tz\_database\_time\_zones

...

|============================================================ | 66%

| View the contents of nyc, which now contains the current date and time in New York.

> nyc

[1] "2016-02-07 04:59:29 EST"

| All that practice is paying off!

|============================================================== | 67%

| 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)

| You nailed it! Good job!

|=============================================================== | 69%

| Take a look at depart.

> depart

[1] "2016-02-09 04:59:29 EST"

| You're the best!

|================================================================= | 71%

| 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)

| Perseverance, that's the answer.

|=================================================================== | 72%

| Take one more look at depart.

> depart

[1] "2016-02-09 17:34:29 EST"

| You are amazing!

|==================================================================== | 74%

| 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.

...

|====================================================================== | 76%

| 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)

| Nice work!

|======================================================================= | 78%

| 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.

...

|========================================================================= | 79%

| The with\_tz() function returns a date-time as it would appear in another time zone. Use ?with\_tz to

| check out the documentation.

> ?with\_tz

| Perseverance, that's the answer.

|=========================================================================== | 81%

| 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, tzone = "Asia/Hong\_Kong")

| You nailed it! Good job!

|============================================================================ | 83%

| Print the value of arrive to the console, so that you can tell your friend what time to pick you up

| from the airport.

> arrive

[1] "2016-02-10 22:24:29 HKT"

| You got it!

|============================================================================== | 84%

| 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.

...

|=============================================================================== | 86%

| 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")

Warning message:

All formats failed to parse. No formats found.

| Keep up the great work!

|================================================================================= | 88%

| View the contents of last\_time.

> last\_time

[1] NA

| You nailed it! Good job!

|================================================================================== | 90%

| Pull up the documentation for new\_interval(), which we'll use to explore how much time has passed

| between arrive and last\_time.

> ?new\_interval

| You are doing so well!

|==================================================================================== | 91%

| Create a new\_interval() that spans from last\_time to arrive. Store it in a new variable called

| how\_long.

> how\_long <- new\_interval(last\_time, arrive)

'new\_interval' is deprecated; use 'interval' instead. Deprecated in version '1.5.0'.

| You are doing so well!

|====================================================================================== | 93%

| Now use as.period(how\_long) to see how long it's been.

> as.period(how\_long)

[1] NA

| You got it!

|======================================================================================= | 95%

| This is where things get a little tricky. Because of things like leap years, leap seconds, and

| daylight savings time, the length of any given minute, day, month, week, or year is relative to

| when it occurs. In contrast, the length of a second is always the same, regardless of when it

| occurs.

...

|========================================================================================= | 97%

| To address these complexities, the authors of lubridate introduce four classes of time related

| objects: instants, intervals, durations, and periods. These topics are beyond the scope of this

| lesson, but you can find a complete discussion in the 2011 Journal of Statistical Software paper

| titled 'Dates and Times Made Easy with lubridate'.

...

|========================================================================================== | 98%

| This concludes our introduction to working with dates and times in lubridate. I created a little

| timer that started running in the background when you began this lesson. Type stopwatch() to see

| how long you've been working!

> stopwatch()

'new\_interval' is deprecated; use 'interval' instead. Deprecated in version '1.5.0'.

[1] "1H 12M 59.4953370094299S"

| Keep up the great work!

|============================================================================================| 100%