Lubrid Package

Marwa ZAHOUANI

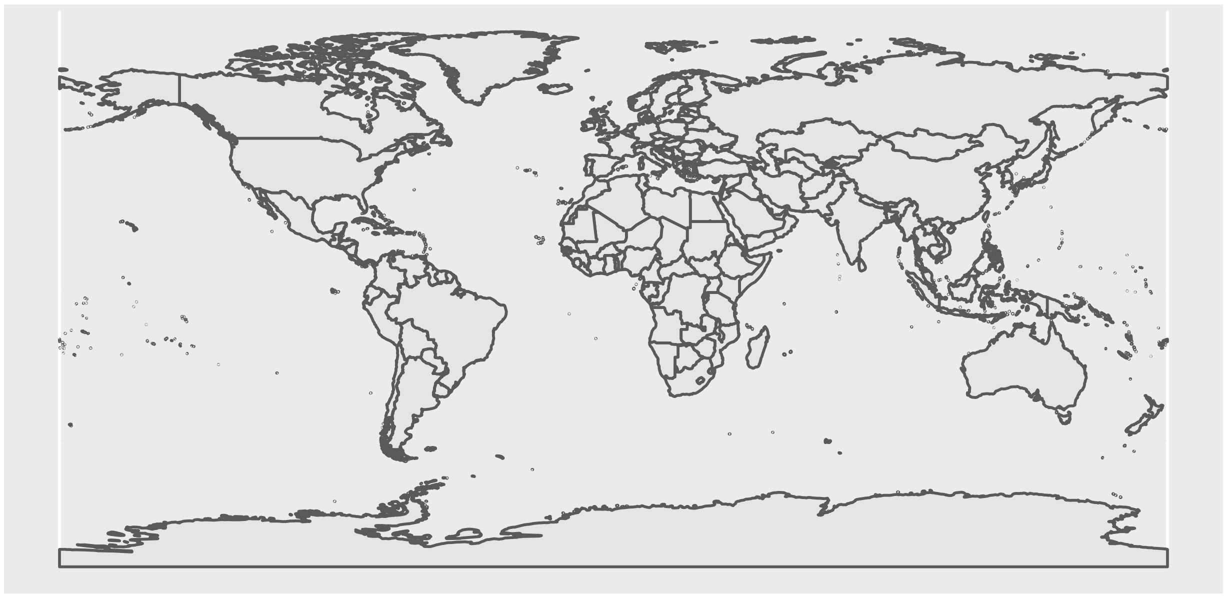
15/02/2022

```r
library(rnaturalearth)
```

```
## Warning: le package 'rnaturalearth' a été compilé avec la version R 4.1.2
```

```r
library(ggplot2)

world_map_data <- ne_countries(scale= "medium", returnclass ="sf")
ggplot(data= world_map_data)+geom_sf()
```



```r
bannerCommenter::banner("Lubridate Package")
```

```
##
## ###################################################################
## ##                     Lubridate Package                         ##
## ###################################################################
```

```
library(lubridate)
```

```
##
## Attachement du package : 'lubridate'

## Les objets suivants sont masqués depuis 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v tibble  3.1.4     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.1     v forcats 0.5.1
## v purrr   0.3.4

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x lubridate::as.difftime() masks base::as.difftime()
## x lubridate::date()        masks base::date()
## x dplyr::filter()          masks stats::filter()
## x lubridate::intersect()   masks base::intersect()
## x dplyr::lag()             masks stats::lag()
## x lubridate::setdiff()     masks base::setdiff()
## x lubridate::union()       masks base::union()
```

```
library(nycflights13)
```

```
## Warning: le package 'nycflights13' a été compilé avec la version R 4.1.2
```

What is the Lubridate Package

Lubridate is an R package that makes it easier to work with dates and times.These tools are grouped below by common purpose. Below is a concise tour of some of the things lubridate can do for you. Lubridate was created by ### Garrett Grolemund and ### Hadley Wickham, and is now maintained by ### Vitalie Spinu.

Dates and times

Introduction

This sheet will show you how to work with dates and times in R. At first glance, dates and times seem simple. You use them all the time in your regular life, and they don't seem to cause much confusion. However, the more you learn about dates and times, the more complicated they seem to get. To warm up, try these three seemingly simple questions: - Does every year have 365 days? - Does every day have 24 hours? - Does every minute have 60 seconds?

Prerequisites

This chapter will focus on the lubridate package, which makes it easier to work with dates and times in R. Lubridate is not part of core tidyverse because you only need it when you're working with dates/times. We will also need nycflights13 for practice data.

Creating date/times

There are three types of date/time data that refer to an instant in time:

A date. Tibbles print this as .

A time within a day. Tibbles print this as .

A date-time is a date plus a time: it uniquely identifies an instant in time (typically to the nearest second). Tibbles print this as . Elsewhere in R these are called POSIXct, but I don't think that's a very useful name.

To get the current date or date-time you can use today() or now():

```
today()
```

```
## [1] "2022-02-16"
```

```
now()
```

```
## [1] "2022-02-16 13:28:34 CET"
```

Otherwise, there are three ways you're likely to create a date/time:

- From a string.
- From individual date-time components.
- From an existing date/time object.

They work as follows.

From strings

Date/time data often comes as strings. You've seen one approach to parsing strings into date-times in date-times. Another approach is to use the helpers provided by lubridate. They automatically work out the format once you specify the order of the component. To use them, identify the order in which year, month, and day appear in your dates, then arrange "y", "m", and "d" in the same order. That gives you the name of the lubridate function that will parse your date. For example:

```
ymd("2018-03-19")
```

```
## [1] "2018-03-19"
```

```
mdy("March 19th, 2018")
```

```
## [1] "2018-03-19"
```

```
dmy("19-Mar-2018")
```

```
## [1] "2018-03-19"
```

These functions also take unquoted numbers. This is the most concise way to create a single date/time object, as you might need when filtering date/time data. ymd() is short and unambiguous:

```
ymd(20180319)
```

```
## [1] "2018-03-19"
```

ymd() and friends create dates. To create a date-time, add an underscore and one or more of "h", "m", and "s" to the name of the parsing function:

```
ymd_hms("2018-03-19 19:11:59")
```

```
## [1] "2018-03-19 19:11:59 UTC"
```

```
mdy_hm("03/19/2018 08:01")
```

```
## [1] "2018-03-19 08:01:00 UTC"
```

You can also force the creation of a date-time from a date by supplying a timezone:

```
ymd(20170131, tz = "GMT")
```

```
## [1] "2017-01-31 GMT"
```

From individual components

Instead of a single string, sometimes you'll have the individual components of the date-time spread across multiple columns. This is what we have in the flights data:

```
data(flights)
flights %>%
  select(year, month, day, hour, minute)
```

```
## # A tibble: 336,776 x 5
##     year month   day  hour minute
##    <int> <int> <int> <dbl>  <dbl>
##  1  2013     1     1     5     15
##  2  2013     1     1     5     29
##  3  2013     1     1     5     40
##  4  2013     1     1     5     45
##  5  2013     1     1     6      0
##  6  2013     1     1     5     58
##  7  2013     1     1     6      0
##  8  2013     1     1     6      0
##  9  2013     1     1     6      0
## 10  2013     1     1     6      0
## # ... with 336,766 more rows
```

From other types

You may want to switch between a date-time and a date. That's the job of as_datetime() and as_date():

```
as_datetime(today())
```

```
## [1] "2022-02-16 UTC"
```

```
as_date(now())
```

```
## [1] "2022-02-16"
```

Sometimes you'll get date/times as numeric offsets from the "Unix Epoch", 1970-01-01. If the offset is in seconds, use as_datetime(); if it's in days, use as_date().

```
as_datetime(60 * 60 * 10)
```

```
## [1] "1970-01-01 10:00:00 UTC"
```

```
as_date(365 * 10 + 2)
```

```
## [1] "1980-01-01"
```

Date-time components

Now that you know how to get date-time data into R's date-time data structures, let's explore what you can do with them. This section will focus on the accessor functions that let you get and set individual components. The next section will look at how arithmetic works with date-times.

Getting components

You can pull out individual parts of the date with the accessor functions year(), month(), mday() (day of the month), yday() (day of the year), wday() (day of the week), hour(), minute(), and second().

```
datetime <- ymd_hms("2022-07-07 10:37:56")

year(datetime)
```

```
## [1] 2022
```

```
month(datetime)
```

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

```
mday(datetime)
```

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

```
yday(datetime)
```

```
## [1] 188
```

```
wday(datetime)
```

```
## [1] 5
```

For month() and wday() you can set label = TRUE to return the abbreviated name of the month or day of the week. Set abbr = FALSE to return the full name.

```
month(datetime, label = TRUE)
```

```
## [1] juil
## 12 Levels: janv < févr < mars < avr < mai < juin < juil < août < ... < déc
```

```
wday(datetime, label = TRUE, abbr = FALSE)
```

```
## [1] jeudi
## 7 Levels: dimanche < lundi < mardi < mercredi < jeudi < ... < samedi
```

Rounding

An alternative approach to plotting individual components is to round the date to a nearby unit of time, with floor_date(), round_date(), and ceiling_date(). Each function takes a vector of dates to adjust and then the name of the unit round down (floor), round up (ceiling), or round to. This, for example, allows us to plot the number of flights per week.

Setting components:

You can also use each accessor function to set the components of a date/time:

```
(datetime <- ymd_hms("2016-07-08 12:34:56"))
```

```
## [1] "2016-07-08 12:34:56 UTC"
```

```
year(datetime) <- 2022
datetime
```

```
## [1] "2022-07-08 12:34:56 UTC"
```

```
month(datetime) <- 02
datetime
```

```
## [1] "2022-02-08 12:34:56 UTC"
```

```
hour(datetime) <- hour(datetime) + 1
datetime
```

```
## [1] "2022-02-08 13:34:56 UTC"
```

Alternatively, rather than modifying in place, you can create a new date-time with update(). This also allows you to set multiple values at once.

```
update(datetime, year = 2022, month = 2, mday = 16, hour = 11)
```

```
## [1] "2022-02-16 11:34:56 UTC"
```

Time spans

Next you'll learn about how arithmetic with dates works, including subtraction, addition, and division. Along the way, you'll learn about three important classes that represent time spans:

- durations, which represent an exact number of seconds.
- periods, which represent human units like weeks and months.
- intervals, which represent a starting and ending point.

Durations

In R, when you subtract two dates, you get a difftime object:

```
# How old is M?
m_age <- today() - ymd(19900707)
m_age
```

```
## Time difference of 11547 days
```

A difftime class object records a time span of seconds, minutes, hours, days, or weeks. This ambiguity can make difftimes a little painful to work with, so lubridate provides an alternative which always uses seconds: the duration.

```
as.duration(m_age)
```

```
## [1] "997660800s (~31.61 years)"
```

Durations come with a bunch of convenient constructors:

```
dseconds(15)
```

```
## [1] "15s"
```

```
dminutes(10)
```

```
## [1] "600s (~10 minutes)"
```

```
dhours(c(12, 24))
```

```
## [1] "43200s (~12 hours)" "86400s (~1 days)"
```

```
ddays(0:5)
```

```
## [1] "0s"                "86400s (~1 days)"  "172800s (~2 days)"
## [4] "259200s (~3 days)" "345600s (~4 days)" "432000s (~5 days)"
```

```
dweeks(3)
```

```
## [1] "1814400s (~3 weeks)"
```

```
dyears(1)
```

```
## [1] "31557600s (~1 years)"
```

Durations always record the time span in seconds. Larger units are created by converting minutes, hours, days, weeks, and years to seconds at the standard rate (60 seconds in a minute, 60 minutes in an hour, 24 hours in day, 7 days in a week, 365 days in a year).

You can add and multiply durations:

```
2 * dyears(1)
```

```
## [1] "63115200s (~2 years)"
```

```
dyears(1) + dweeks(12) + dhours(15)
```

```
## [1] "38869200s (~1.23 years)"
```

You can add and subtract durations to and from days:

```
tomorrow <- today() + ddays(1)
tomorrow
```

```
## [1] "2022-02-17"
```

```
last_year <- today() - dyears(1)
last_year
```

```
## [1] "2021-02-15 18:00:00 UTC"
```

However, because durations represent an exact number of seconds, sometimes you might get an unexpected result:

```
one_pm <- ymd_hms("2022-03-16 13:00:00", tz = "America/New_York")

one_pm
```

```
## [1] "2022-03-16 13:00:00 EDT"
```

```
one_pm + ddays(1)
```

```
## [1] "2022-03-17 13:00:00 EDT"
```

Periods

To solve this problem, lubridate provides periods. Periods are time spans but don't have a fixed length in seconds, instead they work with "human" times, like days and months. That allows them to work in a more intuitive way:

```
one_pm
```

```
## [1] "2022-03-16 13:00:00 EDT"
```

```
one_pm + days(1)
```

```
## [1] "2022-03-17 13:00:00 EDT"
```

Like durations, periods can be created with a number of friendly constructor functions.

```
seconds(15)
```

```
## [1] "15S"
```

```r
minutes(10)
```

```
## [1] "10M 0S"
```

```r
hours(c(12, 24))
```

```
## [1] "12H 0M 0S" "24H 0M 0S"
```

```r
days(7)
```

```
## [1] "7d 0H 0M 0S"
```

```r
months(1:6)
```

```
## [1] "1m 0d 0H 0M 0S" "2m 0d 0H 0M 0S" "3m 0d 0H 0M 0S" "4m 0d 0H 0M 0S"
## [5] "5m 0d 0H 0M 0S" "6m 0d 0H 0M 0S"
```

```r
weeks(3)
```

```
## [1] "21d 0H 0M 0S"
```

```r
years(1)
```

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

You can add and multiply periods:

```r
10 * (months(6) + days(1))
```

```
## [1] "60m 10d 0H 0M 0S"
```

```r
days(50) + hours(25) + minutes(2)
```

```
## [1] "50d 25H 2M 0S"
```

And of course, add them to dates. Compared to durations, periods are more likely to do what you expect:

```r
# A leap year
ymd("2021-01-01") + dyears(1)
```

```
## [1] "2022-01-01 06:00:00 UTC"
```

```r
ymd("2021-01-01") + years(1)
```

```
## [1] "2022-01-01"
```

```r
# Daylight Savings Time
one_pm + ddays(1)
```

```
## [1] "2022-03-17 13:00:00 EDT"
```

```r
one_pm + days(1)
```

```
## [1] "2022-03-17 13:00:00 EDT"
```

Intervals

It's obvious what dyears(1) / ddays(365) should return: one, because durations are always represented by a number of seconds, and a duration of a year is defined as 365 days worth of seconds.

What should years(1) / days(1) return? Well, if the year was 2015 it should return 365, but if it was 2016, it should return 366! There's not quite enough information for lubridate to give a single clear answer. What it does instead is give an estimate, with a warning:

```
years(1) %/% days(1)
```

```
## [1] 365
```

Time zones

Information about time zones in R. Sys.timezone returns the name of the current time zone.

```
Sys.timezone()
```

```
## [1] "Europe/Paris"
```

And see the complete list of all time zone names with OlsonNames():

```
length(OlsonNames())
```

```
## [1] 593
```

```
head(OlsonNames())
```

```
## [1] "Africa/Abidjan"     "Africa/Accra"       "Africa/Addis_Ababa"
## [4] "Africa/Algiers"     "Africa/Asmara"      "Africa/Asmera"
```

In R, the time zone is an attribute of the date-time that only controls printing. For example, these three objects represent the same instant in time:

```
(x1 <- ymd_hms("2022-02-16 12:00:00", tz = "Africa/Algiers"))
```

```
## [1] "2022-02-16 12:00:00 CET"
```

```
(x2 <- ymd_hms("2022-02-16 18:00:00", tz = "Europe/Copenhagen"))
```

```
## [1] "2022-02-16 18:00:00 CET"
```

```
(x3 <- ymd_hms("2022-02-17 04:00:00", tz = "Pacific/Auckland"))
```

```
## [1] "2022-02-17 04:00:00 NZDT"
```

You can verify that they're the same time using subtraction:

```
x1 - x2
```

```
## Time difference of -6 hours
```

```
x1 - x3
```

```
## Time difference of -4 hours
```

```
x2 - x3
```

```
## Time difference of 2 hours
```

References

- https://r4ds.had.co.nz/dates-and-times.html
- https://rdrr.io/cran/lubridate/man/lubridate-package.html
- https://cran.r-project.org/web/packages/lubridate/vignettes/lubridate.html#:~:text=Lubridate%20is%20an%20R%20pa