SeminaR: Dates and times

ZMT

Jan 15, 2020

Contents

In this workshop, we will discuss how to work with dates and times in R.

```
library(tidyverse)
library(lubridate)
```

System date and time

Let's start by checking the date and time of your computer.

```
t <- today()
t
```

```
## [1] "2020-01-30"
```

```
class(t)
```

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

```
n <- now()
n
```

```
## [1] "2020-01-30 13:00:53 +08"
```

```
class(n)
```

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

Create date/time from other objects

Three ways:

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

From a string

```
ymd("2020-01-28")
```

```
## [1] "2020-01-28"
```

```r
dmy("28jan2020")
```

```
## [1] "2020-01-28"
```

```r
mdy("January 28th, 2020")
```

```
## [1] "2020-01-28"
```

You can also convert dates from unquoted numbers.

```r
ymd(20200128)
```

```
## [1] "2020-01-28"
```

Converting date and time. What is the default timezone?

```r
ymd_hms("2010-01-28 23:59:59")
```

```
## [1] "2010-01-28 23:59:59 UTC"
```

```r
dmy_hms("28jan2020 23:59:59")
```

```
## [1] "2020-01-28 23:59:59 UTC"
```

```r
dmy_hms("28jan2020 23-59-59")
```

```
## [1] "2020-01-28 23:59:59 UTC"
```

```r
dmy_hm("28jan2020 23:20")
```

```
## [1] "2020-01-28 23:20:00 UTC"
```

```r
dmy_h("28jan2020 23")
```

```
## [1] "2020-01-28 23:00:00 UTC"
```

From individual components

If we have the components of date/time in various variables as numbers, we can put them together using `make_datetime()`.

```r
make_datetime(
  year = 2020,
  month = 1,
  day = 28,
  hour = 14,
  min = 40,
  sec = 30
)
```

```
## [1] "2020-01-28 14:40:30 UTC"
```

From an existing date/time object

We can create a date/time object from an existing date object and vice versa.

```
x <- ymd_hms("2010-01-28 23:59:59")

# Coarce to a date
d <- as_date(x)
d
```

```
## [1] "2010-01-28"
```

```
# Coarce to a date/time object
t <- as_datetime(d)
t
```

```
## [1] "2010-01-28 UTC"
```

Exercise 1

A dataframe `dat` is provided, including components of date/time. Use these components to create a new date/time variable and assign the dataframe a new name `dat1`.

```
set.seed(1)
dat <- tibble(
  year = 2020,
  month = round(runif(30, min = 1, max = 12)),
  day = round(runif(30, min = 1, max = 28)),
  hour = round(runif(30, min = 1, max = 24)),
  minute = round(runif(30, min = 1, max = 59)),
  second = round(runif(30, min = 1, max = 59))
)
```

```
# Ex:
```

Extracting components

You can extract each component of a date/time object.

```
a <- "2020-05-04 02:55:54"
a_dt <- ymd_hms(a)

year(a_dt)
```

```
## [1] 2020
```

```
day(a_dt)
```

```
## [1] 4
```

```r
hour(a_dt)
```

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

You can also extract the day of the week, month, or year for a given date/time.

```r
mday(a_dt) # day of the month
```

```
## [1] 4
```

```r
yday(a_dt) # day of the year
```

```
## [1] 125
```

```r
wday(a_dt) # day of the week
```

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

If you want to know which quarter or semester (half a year) is in, you can do that too.

```r
quarter(a_dt)
```

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

```r
quarter(a_dt, with_year = TRUE)
```

```
## [1] 2020.2
```

```r
semester(a_dt)
```

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

```r
semester(a_dt, with_year = TRUE)
```

```
## [1] 2020.1
```

Exercise 2

Break down the variable `dt` from `dat1` and create a variable for each component. In addition, find out the day of the month, day of the year, and day of the week. Display month and day of the week in words instead of numbers.

```r
dat2 <- dat1 %>%
  select(dt)
```

```
# Ex:
```

What is the frequency distribution of the quarters of these dates?

```
dat1 %>%
  mutate(q = quarter(dt, with_year = TRUE)) %>%
  count(q)
```

```
## # A tibble: 4 x 2
##       q     n
##   <dbl> <int>
## 1 2020.     7
## 2 2020.     9
## 3 2020.     7
## 4 2020.     7
```

Time spans

Three important classes that represent time spans:

- Duration: an exact number of seconds
- Period: allows calculation by calendar week, month, and year - this way of calculation is probably more familiar to us.
- Intervals: measured in seconds but with a starting date

Let's take a look at functions that work with these time span classes.

Duration

If John Snow (born on March 15, 1813) was alive today, how old would he be?

```
js_age <- today() - ymd(18130315)
js_age
```

```
## Time difference of 75561 days
```

```
class(js_age)
```

```
## [1] "difftime"
```

Duration is the difference between two dates or date/time objects. In base R, substraction of the two dates or date/time objects creates a `difftime` object. Its unit can be seconds, minutes, hours, days, and weeks (read the help file `?difftime`). It can be confusing to work with.

```
x <- seq(from = 10, to = 50, by = 10)
as.difftime(x, units = "secs")
```

```
## Time differences in secs
## [1] 10 20 30 40 50
```

```r
as.difftime(x, units = "hours")
```

```
## Time differences in hours
## [1] 10 20 30 40 50
```

```r
as.difftime(x, units = "weeks")
```

```
## Time differences in weeks
## [1] 10 20 30 40 50
```

In `{lubridate}`, a duration is always represented in seconds.

```r
as.duration(js_age)
```

```
## [1] "6528470400s (~206.87 years)"
```

We can construct duration based on numbers in second, minute, hour, day, week and year using convenient functions from `lubridate`. The output of these functions is the equivalent number of seconds.

```r
dseconds(15)
```

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

```r
dminutes(10)
```

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

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

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

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

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

```r
dweeks(3)
```

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

```r
dyears(1)
```

```
## [1] "31536000s (~52.14 weeks)"
```

Note that there is no `dmonths()`. Why is that?

You can add and multiply durations:

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

```
## [1] "126144000s (~4 years)"
```

```
dyears(1) + dweeks(4) + ddays(7)
```

```
## [1] "34560000s (~1.1 years)"
```

You can add and subtract durations to and from days:

```
today()
```

```
## [1] "2020-01-30"
```

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

```
## [1] "2020-01-31"
```

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

```
## [1] "2019-01-30"
```

Periods

Calculation using durations may get you an unexpected result. The number of seconds in a year or a month can vary because of different days in a month, leap year, and daylight savings time.

```
# A leap year
ymd("2016-01-01") + dyears(1)
```

```
## [1] "2016-12-31"
```

```
# Daylight Savings Time
one_pm <- ymd_hms("2016-03-12 13:00:00", tz = "America/New_York")
one_pm + ddays(1)
```

```
## [1] "2016-03-13 14:00:00 EDT"
```

This is where period comes in. It allows adding one year to January 1, 2020 and become January 1, 2021, instead of 31 December 2020 because of the leap year. Compared to durations, periods are more likely to do what you expect:

```
# A leap year
ymd("2016-01-01") + dyears(1)
```

```
## [1] "2016-12-31"
```

```r
ymd("2016-01-01") + years(1)
```

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

```r
# Daylight Savings Time
one_pm
```

```
## [1] "2016-03-12 13:00:00 EST"
```

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

```
## [1] "2016-03-13 14:00:00 EDT"
```

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

```
## [1] "2016-03-13 13:00:00 EDT"
```

We can construct periods from numbers:

```r
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"
```

```
years(1)
```

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

You can also add and multiply periods.

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

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

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

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

Exercise 3

A new column year is created in `dat1` using random numbers. Convert this column into both duration and period and add them to `dt` column. Check whether and to what extent the results are different and why?

```
dat3 <- dat1
set.seed(2)
dat3$year <- sample(-20:20, nrow(dat3), replace = TRUE)

# Ex:
```

Intervals

`dyears(1) / ddays(1)` returns 365 because durations are always represented by exact number of seconds, and a duration of a year is defined as 365 days worth of seconds.

Let's do the same calculation using periods.

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

```
## estimate only: convert to intervals for accuracy
```

```
## [1] 365.25
```

A period represents calendar time and it returns 365.25 with a message stating that it's an imprecise estimate. Why is that? That is because it would return 366 if it was a leap year. Since we do not know the date that period started, we cannot know the period accurately; we should use an interval instead (if we have information on the start dates).

`interval()` function creates interval objects. `%--%` is the shorthand for the function. An interval object prints the start and end dates rather than seconds.

```
my_int1 <- ymd_hms("2016-03-12 13:00:00") %--% ymd_hms("2017-04-12 13:00:00")
my_int1
```

```
## [1] 2016-03-12 13:00:00 UTC--2017-04-12 13:00:00 UTC
```

```
my_int2 <- ymd_hms("2016-08-01 13:00:00") %--% ymd_hms("2017-08-12 13:00:00")
my_int2
```

```
## [1] 2016-08-01 13:00:00 UTC--2017-08-12 13:00:00 UTC
```

You can extract start and end dates.

```
int_start(my_int1)
```

```
## [1] "2016-03-12 13:00:00 UTC"
```

```
int_end(my_int1)
```

```
## [1] "2017-04-12 13:00:00 UTC"
```

You can change the start or end dates in place.

```
int_start(my_int1) <- ymd_hms("2016-01-01 13:00:00")
int_end(my_int1) <- ymd_hms("2017-01-01 13:00:00")
my_int1
```

```
## [1] 2016-01-01 13:00:00 UTC--2017-01-01 13:00:00 UTC
```

You can test whether a date falls within an interval.

```
ymd_hms("2016-05-01 13:00:00") %within% my_int1
```

```
## [1] TRUE
```

```
ymd_hms("1999-05-01 13:00:00") %within% my_int1
```

```
## [1] FALSE
```

You can also test whether two intervals overlaps.

```
int_overlaps(my_int1, my_int2)
```

```
## [1] TRUE
```

Time zones

Time zone is an attribute of a date/time object that controls how the time is displayed. Default time zone in `lubridate` parse functions is UTC (Coordinated Universal Time).

```
t1 <- ymd_hms("2020-01-01 6:30:00", tz = "Asia/Yangon")
t2 <- ymd_hms("2020-01-01 8:00:00", tz = "Singapore")
t3 <- ymd_hms("2020-01-01 0:00:00", tz = "GMT")
t4 <- ymd_hms("2020-01-01 0:00:00")
```

These 3 times are the same.

```
t1 - t2
```

```
## Time difference of 0 secs
```

```
t1 - t3
```

```
## Time difference of 0 secs
```

```
t1 - t4
```

```
## Time difference of 0 secs
```

R uses the international standard IANA time zones. These use a consistent naming scheme "/", typically in the form "/" Examples include "America/New_York", "Europe/Paris", and "Pacific/Auckland". There are a few exceptions because not every country lies on a continent. For example, "Singapore".

You can find out what R thinks your current time zone is.

```
Sys.timezone()
```

```
## [1] "Asia/Kuala_Lumpur"
```

You can check out the complete list of all time zones using `OlsonNames()`

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

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

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

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

Two ways to change the time zone:

1. Change how the time is displayed and keep the time instant unchanged.

```
t1
```

```
## [1] "2020-01-01 06:30:00 +0630"
```

```
t5 <- with_tz(t1, tzone = "Asia/Calcutta")
t1 - t5
```

```
## Time difference of 0 secs
```

2. Change the time instant. Use this when the instant is labelled with an incorrect timezone.

```
t6 <- force_tz(t1, tzone = "Asia/Calcutta")
t1 - t6
```

```
## Time difference of -1 hours
```