QTM 151

Week 6 – lubridate

Umberto Mignozzetti

Recap

We learned:

- orogramming methods
- `dplyr *_join` methods: joining data
- `tidyr` methods: reshape datasets
- `forcats` methods: working with categorical variables

Great job!!

Do you have any questions about any of these contents?

Today we are going to talk about **lubridate** (package for dates and time)

Our GitHub page is: https://github.com/umbertomig/qtm151

Getting Started

Getting Started: loading packages

```r
# Loading tidyverse
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyv

## ✓ ggplot2 3.3.5     ✓ purrr   0.3.4
## ✓ tibble  3.1.6     ✓ dplyr   1.0.8
## ✓ tidyr   1.2.0     ✓ stringr 1.4.0
## ✓ readr   2.1.2     ✓ forcats 0.5.1

## Warning: package 'tidyr' was built under R version 4.1.2

## Warning: package 'readr' was built under R version 4.1.2

## Warning: package 'dplyr' was built under R version 4.1.2

## ── Conflicts ────────────────────────────────────────── tidyverse_c
## x dplyr::filter() masks stats::filter()
```

lubridate

lubridate

Provide tools to work with dates. The methods we will use in here are:

- `ymd()`, `ydm()`, `mdy()`, `myd()`, `dmy()`, `dym()`: read text into data, provided in this order.
- `parse_date_time()`: parse other date and time objects.
- `make_date()` and `make_datetime()`: parse dates and times.
- `round_dates()` and others: round dates and times.
- `difftime()`: time/dates differences.
- `now()` and `today()`: current dates and times.
- `seconds()`, `dseconds()` and others: create time spans and durations.
- `%--%` and date intervals: create and check date intervals.

Dates and Strings

Dates and Strings

Without declare an object as a date, it is just a string.

Check that in R (for ISO 8601):

```
dt ← '2021-02-22'
class(dt)
str(dt)
```

```
dt ← as.Date(dt)
class(dt)
str(dt)
```

ymd() and related functions

ymd() and related functions

These functions are great to parse dates:

```
x ← "2010 September 20th"
ymd(x)

y ← "02.01.2010"
dmy(y)

z ← "Sep, 12th 2010 14:00"
mdy_hm(z)

a ← "2016-07-08 12:34:56"
ymd_hms(a)
```

ymd() and related functions

Your turn: Turn into date:

1. "2010-01-22"
2. "12-02-1234 12:30:25"
3. "May 18, 1933"
4. "1st June 2012"

parse_date_time

parse_date_time

`parse_date_time()` takes two arguments: First, a vector with dates. Second, a vector with orders to parse.

E. g.: `parse_date_time(date = "some_date", order = "some_order")`

Some patterns:

- `dmy`: day, month, year (4-digit)
- `mdY`: month, day, year (2-digit)
- `ymdHMS`: year (4-digit), month, day, hours, minutes, seconds

For more: `help(parse_date_time)`

parse_date_time

Examples:

```
parse_date_time(c("11-01-2016", "2016 Jan 11th"),
  order = c("dmy", "ymd"))
parse_date_time("11-01-2016", order = "dmy")
```

Your turn: Parse:

1. `x ← c("April 13, 2003", "17 April 2005")`
2. `y ← "January 10, 2020 at 23:30:35"`

make_date() and make_datetime()

make_date() and make_datetime()

Both functions work to build date and time from pieces. The syntax for both functions is straightforward:

```
make_date(year = 2012, month = 3, day = 27)
```

```
make_datetime(year = 1234, month = 5, day = 12,
  hour = 1, min = 23, sec = 45)
```

Your turn: make dates from:

- yrs <- c(1234, 1222, 2020)
- mos <- c(4, 2, 12)
- dss <- c(12, 12, 20)

Extract parts functions

Extract parts functions

- year(): Year with century
- month(): Month
- day(): Day of month
- hour(): Hour
- min(): Minute
- second(): Second
- wday(): Weekday
- yday(): Day of year
- tz(): Timezone
- quarter(): Quarter
- semester(): Semester

Extract parts functions

Your turn: Parse the dates and try these functions with:

1. "2010-01-22"
2. "12-02-1234 12:30:25"

round_dates() and others

round_dates() and others

To round dates we can use:

- `round_dates()`: round to the nearest precision.
- `ceiling_date()`: round up the nearest precision.
- `floor_dates()`: round down the nearest precision.

Precision units:

- "second" , "minute" , "hour" , "day" , "week" , "month", "bimonth" , "quarter" , "halfyear" , "year", or multiples (e.g. "10 minutes")

round_dates() and others

Examples:

```
d ← ymd_hms("1234-04-03 07:13:28 UTC")
floor_date(d, unit = 'day')

# Round to nearest 5 minutes
round_date(d, unit = '5 minutes')

# Round up to week
ceiling_date(d, unit = 'week')

# Fun
ceiling_date(d, unit = 'hour') - floor_date(d, unit = 'hour')
```

Your turn: Round `d` in the three different ways: `d ← ymd_hms("1981-08-18 18:08:28 UTC")`

difftime

difftime

We can find difference between two dates using difftime.

```
difftime(today(), mdy("May 20, 2020"), units = 'days')
difftime(now(), mdy_hms("May 20, 2020 22:12:22"), units = 'secs')
```

Your turn: Find how many seconds since your birthday.

seconds(), dseconds() and other duration functions

seconds(), dseconds() and other

We can use duration to change dates:

```
today() - years(1)
today() + months(1)
days(2)
ddays(2)
now() + seconds(20)
2*days()
```

Your turn: Find how many seconds since your birthday.

Intervals

Intervals

We can also compute intervals:

```
x ← dmy("2 January 1998") %--% dmy("30 March 2018")
y ← dmy("2 January 2017") %--% dmy("30 March 2020")
int_start(x)
int_end(x)
int_length(x)
as.period(x)
dmy("5 January 1998") %within% x
dmy("5 January 1998") %within% y
int_overlaps(x, y)
```

Your turn: Compute one interval between your birthday and today. Check it. Does it contains the date `January 20 2000`?

Questions?

Have a great weekend!