Dates and times with lubridate: CHEAT SHEET

Date-times

2017-11-28 12:00:00
A date-time is a point on the timeline, stored as the number of seconds since 1970-01-01 00:00:00 UTC

dt <- as_datetime(1511870400)  
## "2017-11-28 12:00:00 UTC"

12:00:00
A date is a day stored as the number of days since 1970-01-01

d <- as_date(17498)  
## "2017-11-28"

t <- hms(as_hms(85))  
## 00:01:25

GET AND SET COMPONENTS

Use an accessor function to get a component. Assign into an accessor function to change a component in place.

date(x) Date component. date(dt)
year(x) Year, year(dt)
isoyear(x) The ISO 8601 year.
epiyear(x) Epidemiological year.
month(x, label, abbr) Month.
month(dt)
day(x) Day of month. day(dt)
wday(x, label, abbr) Day of week.
qday(x) Day of quarter.
hour(x) Hour, hour(dt)
minute(x) Minutes. minute(dt)
second(x) Seconds. second(dt)
tz(x) Time zone. tz(dt)

GET COMPONENTS FROM ORDERED ELEMENTS

Use the function below whose name replicates the order. Each accepts a tz argument to set the time zone, e.g. ymd(x, tz = "UTC"").

ymd_hms(), ymd_hm(), ymd_h(),
ymd_hms(dt)

ymd_hms(), ymd_hm(), ymd_h(),
ymd_hms(2017-2814:02:00)

mdy_hms(), mdy_hm(), mdy_h(),
mdy_hms(11/28/2017 12:00:00)

dmy_hms(), dmy_hm(), dmy_h(),
dmy_hms(1 Jan 2017 23:59:59)

ymd(), ymdm(),
ymdm(20170131)

mdy(), myd(),
mdy("July 4th, 2000")

dmy(), dmy(),
dmy("4th of July '99")

yq() Q for quarter. yq("2001: Q3")

my(), ym(),
my("07-02-20")

hms: hms() Also lubridate: hms(), h() and ms(), which return periods. hms:hms(0, min=1, hours = 2, roll = FALSE)

Stamp Date-times

stamp() Derive a template from an example string and return a new function that will apply the template to date-times. Also stamp_date() and stamp_time().

1. Derive a template, create a function
s = stamp("Created Tuesday, Jan 17, 1999 3:34")
2. Apply the template to dates
s(ymd("2010-04-05"))

Round Date-times

floor_date(x, unit = "second") Round down to nearest unit.

floor_date(dt, unit = "month")

round_date(x, unit = "second") Round to nearest unit.

round_date(dt, unit = "month")

ceil_date(x, unit = "second", change_on_boundary = NULL) Round up to nearest unit.

ceil_date(dt, unit = "month")

Tip use a date with day > 12

Valid units are second, minute, hour, day, week, month, bimonth, quarter, season, half-year and year.

rollback(date, roll_to_first = FALSE, preserve_hms = TRUE) Roll back to last day of previous month. Also rollforward() and rollback(dt)

Time Zones

R recognizes ~600 time zones. Each encodes the time zone, Daylight Savings Time, and historical calendar variations for an area. R assigns one time zone per vector.

Use the UTC time zone to avoid Daylight Savings.

OlsonNames() Returns a list of valid time zone names. OlsonNames()

Sys.timezone() Gets current time zone.

with_tz(time, tzzone = "") Get the same date-time in a new time zone (a new clock time). Also local_time(dt, tz, units).

force_tz(time, tzzone = "") Get the same clock time in a new time zone (a new date-time).

force_tz(dt, "US/Pacific")
Math with Date-times — Lubridate provides three classes of timespans to facilitate math with dates and date-times.

PERIODS
Add or subtract periods to model events that happen at specific clock times, like the NYSE opening bell.

Make a period with the name of a time unit **pluralized**, e.g.

- years(x = 1) x years
- months(x = 1) x months
- days(x = 1) x days
- hours(x = 1) x hours
- minutes(x = 1) x minutes
- seconds(x = 1) x seconds
- milliseconds(x = 1) x milliseconds
- microseconds(x = 1) x microseconds
- nanoseconds(x = 1) x nanoseconds
- picoseconds(x = 1) x picoseconds

Example:

```r
nor <- ymd_hms("2018-01-01 01:30:00", tz="US/Eastern")
```

**DURATIONS**
Add or subtract durations to model physical processes, like battery life. Durations are stored as seconds, the only time unit with a consistent length. **Diftimes** are a class of durations found in base R.

Make a duration with the name of a period prefixed with a `d`, e.g.

```r
dd <- ddays(14)
dd "1209600s (~2 weeks)"
```

- dyears(x = 1) x 31536000x seconds
- dmonths(x = 1) x 2629800x seconds
- dweeks(x = 1) x 604800x seconds
- ddays(x = 1) x 86400x seconds
- dhours(x = 1) x 3600x seconds
- dminutes(x = 1) x 60x seconds
- dseconds(x = 1) x 1x seconds
- dmilliseconds(x = 1) x 10^3 seconds
- microseconds(x = 1) x 10^6 seconds
- nanoseconds(x = 1) x 10^9 seconds
- picoseconds(x = 1) x 10^12 seconds

Example:

```r
dy <- dyears(1)
dy 31536000x seconds
```

**INTERVALS**
Divide an interval by a duration to determine its physical length, divide an interval by a period to determine its implied length in clock time.

Make an interval with `interval()` or `%--%`, e.g.

```r
i <- interval(ymd("2017-01-01"), d)
```

- `int_length(i)` Length in seconds.
- `int_start(i)` Start
- `int_end(i)` End
- `int_overlaps(i, j)` Does interval or date-time `i` overlap interval or date-time `j`?
- `int_aligns(i, j)` Do intervals `i` and `j` align?
- `int_diff_times(i, j)` Make the intervals that occur between the date-times in a vector.
- `int_flip(i)` Reverse the direction of an interval.
- `int_standardize(i)` Make `i` a standardized interval.
- `int_shift(i, by)` Shifts an interval up or down the timeline by a timespan.
- `as.interval(x, start, end)` Coerce a timespan to an interval with the start and end date-time.