Date and Time Processing in R

Professor Bear

September 1, 2018

In this lesson we'll learn date and time processing in R. R has a range of functions that allow you to work with dates and times. Packages such as lubridate make it easier to work with dates and times.

Additional packages needed

- If necessary install the followings packages.

```
install.packages("lubridate");
```

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following object is masked from 'package:base':
##
##     date
```

Date and Time Processing in R

Getting current date & time

To get current date and time.

```
Sys.timezone()
```

```
## [1] "America/New_York"
```

```
Sys.Date()
```

```
## [1] "2018-09-09"
```

```
Sys.time()
```

```
## [1] "2018-09-09 20:39:28 EDT"
```

The `lubridate` package has similar functions

```
now() ## Requires library(lubridate)
```

```
## [1] "2018-09-09 20:39:28 EDT"
```

```
today()

## [1] "2018-09-09"
```

Converting strings to dates

When date and time data are imported into R they will often default to a character string. This requires one to convert strings to dates to use the data properly.

Convert Strings to Dates

A string that is already in a date format (YYYY-MM-DD) is easily converted is the `as.Date()` function:

```
d <- c("2016-07-08", "2016-09-01", "2016-10-03")
as.Date(d)

## [1] "2016-07-08" "2016-09-01" "2016-10-03"
```

If the string is in a different format you must incorporate the `format` argument. Type `?strftime` to see these formats.

Date-time Conversion Functions

Functions to convert between character representations and objects of classes “POSIXlt” and “POSIXct” representing calendar dates and times. Unix time (also known as POSIX time or Epoch time) is a system for describing instants in time, defined as the number of seconds that have elapsed since 00:00:00

Usage

```
## S3 method for class 'POSIXct'
format(x, format = "", tz = "", usetz = FALSE, ...)
## S3 method for class 'POSIXlt'
format(x, format = "", usetz = FALSE, ...)

## S3 method for class 'POSIXt'
as.character(x, ...)

strftime(x, format = "", tz = "", usetz = FALSE, ...)
strptime(x, format, tz = "")
```

Here is an example of using a format string.

```
d2 <- c("07/01/2016", "08/08/2016", "10/03/2016")
d2 <-as.Date(d2, format = "%m/%d/%Y")
```

Format string modifiers:

%a Abbreviated weekday name in the current locale on this platform. (Also matches full name on input: in some locales there are no abbreviations of names.)

%A Full weekday name in the current locale. (Also matches abbreviated name on input.)

%b Abbreviated month name in the current locale on this platform. (Also matches full name on input: in some locales there are no abbreviations of names.)

%B Full month name in the current locale. (Also matches abbreviated name on input.)

%c Date and time. Locale-specific on output, "%a %b %e %H:%M:%S %Y" on input.

%C Century (00–99): the integer part of the year divided by 100.

%d Day of the month as decimal number (01–31).

%D Date format such as %m/%d/%y: the C99 standard says it should be that exact format (but not all OSes comply).

%e Day of the month as decimal number (1–31), with a leading space for a single-digit number.

%F Equivalent to %Y-%m-%d (the ISO 8601 date format).

%g The last two digits of the week-based year (see %V). (Accepted but ignored on input.)

%G The week-based year (see %V) as a decimal number. (Accepted but ignored on input.)

%h Equivalent to %b.

%H Hours as decimal number (00–23). As a special exception strings such as 24:00:00 are accepted for input, since ISO 8601 allows these.

%I Hours as decimal number (01–12).

%j Day of year as decimal number (001–366).

%m Month as decimal number (01–12).

%M Minute as decimal number (00–59).

%n Newline on output, arbitrary whitespace on input.

%p AM/PM indicator in the locale. Used in conjunction with %I and not with %H. An empty string in some locales (and the behaviour is undefined if used for input in such a locale).

Some platforms accept %P for output, which uses a lower-case version: others will output P.

%r The 12-hour clock time (using the locale's AM or PM). Only defined in some locales.

%R Equivalent to %H:%M.

%S Second as integer (00–61), allowing for up to two leap-seconds (but POSIX-compliant implementations will ignore leap seconds).

%t Tab on output, arbitrary whitespace on input.

%T Equivalent to %H:%M:%S.

%u Weekday as a decimal number (1–7, Monday is 1).

%U Week of the year as decimal number (00–53) using Sunday as the first day 1 of the week (and typically with the first Sunday of the year as day 1 of week 1). The US convention.

%V Week of the year as decimal number (01–53) as defined in ISO 8601. If the week (starting on Monday) containing 1 January has four or more days in the new year, then it is considered week 1. Otherwise, it is the last week of the previous year, and the next week is week 1. (Accepted but ignored on input.)

%w Weekday as decimal number (0–6, Sunday is 0).

%W Week of the year as decimal number (00–53) using Monday as the first day of week (and typically with the first Monday of the year as day 1 of week 1). The UK convention.

%x Date. Locale-specific on output, "%y/%m/%d" on input.

%X Time. Locale-specific on output, "%H:%M:%S" on input.

%y Year without century (00–99). On input, values 00 to 68 are prefixed by 20 and 69 to 99 by 19 – that is the behaviour specified by the 2004 and 2008 POSIX standards, but they do also say 'it is expected that in a future version the default century inferred from a 2-digit year will change'.

%Y Year with century. Note that whereas there was no zero in the original Gregorian calendar, ISO 8601:2004 defines it to be valid (interpreted as 1BC): see https://en.wikipedia.org/wiki/0_(year). Note that the standards also say that years before 1582 in its calendar should only be used with agreement of the parties involved.

For input, only years 0:9999 are accepted.

%z Signed offset in hours and minutes from UTC, so -0800 is 8 hours behind UTC. Values up to +1400 are accepted as from R 3.1.1: previous versions only accepted up to +1200. (Standard only for output.)

%Z (Output only.) Time zone abbreviation as a character string (empty if not available). This may not be reliable when a time zone has changed abbreviations over the years.

Formating with lubridate

If using the `lubridate` package:

```
ymd(d)

## [1] "2016-07-08" "2016-09-01" "2016-10-03"

mdy(d2)

## Warning: All formats failed to parse. No formats found.
```

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

```
ymd("20110604")
```

```
## [1] "2011-06-04"
```

```
mdy("06-04-2011")
```

```
## [1] "2011-06-04"
```

```
second(d[1])
```

```
## [1] 0
```

Merging Date Strings

Sometimes your date data are collected in separate elements. To merge isolated data in to date objects use the `ISOdate()` function.

```
yr <- c("2012", "2013", "2014", "2015", "2016", "2017")
mo <- c("1", "3", "7", "10")
day <- c("02", "3", "11", "19", "28")
ISOdate(year = yr, month = mo, day = day)
```

```
## [1] "2012-01-02 12:00:00 GMT" "2013-03-03 12:00:00 GMT"
## [3] "2014-07-11 12:00:00 GMT" "2015-10-19 12:00:00 GMT"
## [5] "2016-01-28 12:00:00 GMT" "2017-03-02 12:00:00 GMT"
```

```
as.Date(ISOdate(year = yr, month = mo, day = day))
```

```
## [1] "2012-01-02" "2013-03-03" "2014-07-11" "2015-10-19" "2016-01-28"
## [6] "2017-03-02"
```

Extract & manipulate parts of dates

Here is where the `lubridate` package makes things much easier.

```
d
```

```
## [1] "2016-07-08" "2016-09-01" "2016-10-03"
```

```
year(d)
```

```
## [1] 2016 2016 2016
```

```
year(d[1])
```

```
## [1] 2016
```

```
month(d)
```

```
## [1]  7  9 10
```

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

```
## [1] Jul Sep Oct
## 12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Sep < ... < Dec

month(d, label = TRUE, abbr = FALSE)

## [1] July      September October
## 12 Levels: January < February < March < April < May < June < ... <
December

wday(d, label = TRUE, abbr = FALSE)

## [1] Friday   Thursday Monday
## 7 Levels: Sunday < Monday < Tuesday < Wednesday < Thursday < ... <
Saturday
```

We can also use lubridate to change values.

```
d2

## [1] "2016-07-01" "2016-08-08" "2016-10-03"

class(d2)

## [1] "Date"

mday(d2)

## [1] 1 8 3

mday(d2) <- c(3, 10, 22)
d2

## [1] "2016-07-03" "2016-08-10" "2016-10-22"

update(d2, year = c(2013, 2014, 2015), month = 9)

## [1] "2013-09-03" "2014-09-10" "2015-09-22"

d2 + years(1) - days(c(2, 9, 21))

## [1] "2017-07-01" "2017-08-01" "2017-10-01"
```

Creating date sequences

To create a sequence of dates, that is a time series, we can use the seq operator on date objects.

```
seq(as.Date("2000-1-1"), Sys.Date(), by = "years")

##  [1] "2000-01-01" "2001-01-01" "2002-01-01" "2003-01-01" "2004-01-01"
##  [6] "2005-01-01" "2006-01-01" "2007-01-01" "2008-01-01" "2009-01-01"
## [11] "2010-01-01" "2011-01-01" "2012-01-01" "2013-01-01" "2014-01-01"
## [16] "2015-01-01" "2016-01-01" "2017-01-01" "2018-01-01"
```

```r
seq(as.Date("2011-1-1"), Sys.Date(), by = "quarter")
```

```
##  [1] "2011-01-01" "2011-04-01" "2011-07-01" "2011-10-01" "2012-01-01"
##  [6] "2012-04-01" "2012-07-01" "2012-10-01" "2013-01-01" "2013-04-01"
## [11] "2013-07-01" "2013-10-01" "2014-01-01" "2014-04-01" "2014-07-01"
## [16] "2014-10-01" "2015-01-01" "2015-04-01" "2015-07-01" "2015-10-01"
## [21] "2016-01-01" "2016-04-01" "2016-07-01" "2016-10-01" "2017-01-01"
## [26] "2017-04-01" "2017-07-01" "2017-10-01" "2018-01-01" "2018-04-01"
## [31] "2018-07-01"
```

```r
seq(as.Date("2016-1-1"), Sys.Date(), by = "33 days")
```

```
##  [1] "2016-01-01" "2016-02-03" "2016-03-07" "2016-04-09" "2016-05-12"
##  [6] "2016-06-14" "2016-07-17" "2016-08-19" "2016-09-21" "2016-10-24"
## [11] "2016-11-26" "2016-12-29" "2017-01-31" "2017-03-05" "2017-04-07"
## [16] "2017-05-10" "2017-06-12" "2017-07-15" "2017-08-17" "2017-09-19"
## [21] "2017-10-22" "2017-11-24" "2017-12-27" "2018-01-29" "2018-03-03"
## [26] "2018-04-05" "2018-05-08" "2018-06-10" "2018-07-13" "2018-08-15"
```

Calculations with dates

Dates are objects that support operators such as logical comparisons, addition, subtraction, and many others.

```r
n <- Sys.Date()
n
```

```
## [1] "2018-09-09"
```

```r
n2 <- as.Date("2015-10-03")
n > n2
```

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

```r
n - n2
```

```
## Time difference of 1072 days
```

Note that not all operators that work on numbers work on dates.

```r
n + n2
```

However lubridate may support some operators.

```r
n + days(4)
```

```
## [1] "2018-09-13"
```

```r
n - hours(4)
```

```
## [1] "2018-09-08 20:00:00 UTC"
```

Dealing with time zones & daylight savings

To change the time zone for a date/time we can update the tz parameter with lubridates *with_tz()* function. "For a complete list of valid time zones for use with lubridate, see tz database time zones.

```
boston <- now("America/New_York")
boston

## [1] "2018-09-09 20:39:28 EDT"

meeting <- ymd_hms("2016-10-03 09:00:00", tz = "Pacific/Auckland")
with_tz(meeting, "America/Chicago")

## [1] "2016-10-02 15:00:00 CDT"
```

Additional resources

- Dates and times made easy with lubridate - original lubridate paper
- Date and time classes in R