Dates (Week 1)

Janelle Morano

January 28, 2024

Stat Computing - Exercises 01 - Dates and Date-times

Include your answers as code chunks within this document, and render (knit) it as a pdf.

To help the graders differentiate between the question statements and your own comments, please change the formatting of the question statements to italics.

Due to the many formatting conventions for dates and times, and due to time zone considerations, dates can be tricky to work with. These exercises will give you some practice working with dates and times in R.

1. Read the documentation for `as.Date` *and answer the following questions:*

a. Why does `as.Date("2024-01-01")` *work without supplying a format?*

```
# "%Y-%m-%d" is the default format (ISO international standard). Any other format
# must be specified to be recognized and transformed.
```

b. Why doesn't `as.Date(365)` *work and how can you make it work to produce the date 2024-01-01?*

```
# The origin time point of as.Date is "1970-01-01", and the value given will
# count days from the origin, thus it produces "1971-01-01".

# To make it work, set a new origin
as.Date(365, origin = "2023-01-01")
```

```
## [1] "2024-01-01"
```

```
# An alternative way to get the date using the number of days and strptime()
strptime(1, "%j")
```

```
## [1] "2024-01-01 EST"
```

2. Using a combination of `seq` *and* `as.Date`*, display the dates of our Tuesday lectures throughout the semester (you can ignore the fact that we don't meet during spring break).*

```
seq(as.Date("2024-01-23"), as.Date("2024-05-07"), 7)
```

```
##  [1] "2024-01-23" "2024-01-30" "2024-02-06" "2024-02-13" "2024-02-20"
##  [6] "2024-02-27" "2024-03-05" "2024-03-12" "2024-03-19" "2024-03-26"
## [11] "2024-04-02" "2024-04-09" "2024-04-16" "2024-04-23" "2024-04-30"
## [16] "2024-05-07"
```

3. Read the documentation for `strptime` *and use* `format` *convert 2024-01-23 to these formats.*

a. 01/23/2024

```
day <- as.Date("2024-01-23", format = "%Y-%m-%d")
day
```

```
## [1] "2024-01-23"
```

```
format(day, format = "%m/%d/%Y")
```

```
## [1] "01/23/2024"
```

b. 23/01/2024

```
format(day, format = "%d/%m/%Y")
```

```
## [1] "23/01/2024"
```

c. 01/23/24

```
format(day, format = "%m/%d/%y")
```

```
## [1] "01/23/24"
```

d. January 23, 2024

```
format(day, format = "%B %d, %Y")
```

```
## [1] "January 23, 2024"
```

e. Jan 23, 2024

```
format(day, format = "%b %d, %Y")
```

```
## [1] "Jan 23, 2024"
```

4. Use `format` to figure out what day of the week you were born on.

```
format(as.Date("2015-03-26"), format = "%A")
```

```
## [1] "Thursday"
```

5. How many days are in the years 2000, 2022, 2024, and 2100? Why? Hint: R gives the right answer for all of these.

```
a <- seq(as.Date("2000-01-01"), as.Date("2000-12-31"), 1)
length(a)
```

```
## [1] 366
```

```
b <- seq(as.Date("2022-01-01"), as.Date("2022-12-31"), 1)
length(b)
```

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

```
c <- seq(as.Date("2024-01-01"), as.Date("2024-12-31"), 1)
length(c)
```

```
## [1] 366
```

```
d <- seq(as.Date("2100-01-01"), as.Date("2100-12-31"), 1)
length(d)
```

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

Years 2000 and 2024 with 366 days are leap years.

6. In a few sentences, describe the main differences between POSIXct and POSIXlt. Hint: `?POSIXlt`

POSIXct:

- calendar time where value represents the number of seconds since 1970-01-01 in UTC time zone

- more convenient for including in data frames

POSIXlt:

- named list of vectors representing date-times in local time
- closer to human-readable forms
- objects are assumed in current time zone unless otherwise specified

7. Convert 2024-01-01 and 2024-07-01 to POSIXct format, and print them out. What do the three-letter abbreviations stand for?

```
print(as.POSIXct("2024-01-01"))
```

```
## [1] "2024-01-01 EST"
```

```
print(as.POSIXct("2024-07-01"))
```

```
## [1] "2024-07-01 EDT"
```

EST and EDT are eastern standard time and eastern daylight time, because POSIXct will automatically convert date-time to local time zones (and account for daylight savings time throughout the year).

8. Use `as.POSIXct` to create two date time objects, one referring to 2024-01-26 at noon in Ithaca's time zone and one referring 2024-01-26 at noon in UTC time. Calculate the difference between these two objects.

```
ith <- (as.POSIXct("2024-01-26 12:00:00", tz = "EST"))
utc <- (as.POSIXct("2024-01-26 12:00:00", tz = "UTC"))

ith-utc
```

```
## Time difference of 5 hours
```

9. Use the `attr` function to set the `"tzone"` attribute for the UTC object to Ithaca time, and confirm that the two printed times differ by the amount in the previous question.

```
# Check the time zone
attr(utc, which = "tzone")
```

```
## [1] "UTC"
```

```
# Change the time zone of utc to "EST"
attr(utc, which = "tzone") <- "EST"
# Check the time zone again
attr(utc, which = "tzone")
```

```
## [1] "EST"
```

```
# See the time displayed for each object, which should be in EST
ith
```

```
## [1] "2024-01-26 12:00:00 EST"
```

```
utc
```

```
## [1] "2024-01-26 07:00:00 EST"
```

```
# Verify that the time difference is 5 hours
ith-utc
```

```
## Time difference of 5 hours
```

10. Create a vector that has the datetimes 2024-01-26 12:00:00 and 2024-07-26 12:00:00 in the America/New_York time zone. Then successively change the time zone to the below time zones, and print the result. Do you notice anything unexpected?

America/Chicago, America/Denver, America/Phoenix, America/Los_Angeles

```
# America/New_York
times <- c(as.POSIXct("2024-01-26 12:00:00", tz = "America/New_York"),
           as.POSIXct("2024-07-26 12:00:00", tz = "America/New_York"))
attr(times, which = "tzone")
```

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

```
print(times)
```

```
## [1] "2024-01-26 12:00:00 EST" "2024-07-26 12:00:00 EDT"
```

```
# America/Chicago
attr(times, which = "tzone") <- "America/Chicago"
times
```

```
## [1] "2024-01-26 11:00:00 CST" "2024-07-26 11:00:00 CDT"
```

```
# America/Denver
attr(times, which = "tzone") <- "America/Denver"
times
```

```
## [1] "2024-01-26 10:00:00 MST" "2024-07-26 10:00:00 MDT"
```

```
# America/Phoenix
attr(times, which = "tzone") <- "America/Phoenix"
times
```

```
## [1] "2024-01-26 10:00:00 MST" "2024-07-26 09:00:00 MST"
```

```
# America/Los_Angeles
attr(times, which = "tzone") <- "America/Los_Angeles"
times
```

```
## [1] "2024-01-26 09:00:00 PST" "2024-07-26 09:00:00 PDT"
```

The time zones change as standard time or daylight time for the given year and local time zone. Arizona doesn't recognize daylight savings time so the time zone is the same throughout the year.

11. Share something interesting that you learned from reading the documentation for these functions, or found online.

Although I had previously worked with date-times in R before, I have always worked with data sets where the default time zone was kept at UTC to avoid the confusion of daylight savings time when comparing acrosss time zones and throughout the year. I hadn't appreciated how the tzone attribute can maintain local time zone changes but relationally calculate accurate times. Additionally, I hadn't thought about years before 1 CE and that they are printed as negative years from 0 rather than the standard of "Year" BCE (before common era).