Dates in R

DATA 2010–Tools and Techniques in Data Science

Lecture Objectives

- Describe how dates and times can be a challenging data type
- Use the `lubridate` package to manipulate dates

Motivation

- Dates and times are notoriously difficult to program with.
    - How many days in a year?
    - How many hours in a day? How do you account for Daylight Savings?
    - How many seconds in a minute? What about leap seconds?
- Time zones can also be confusing.
    - Not every time zone follows DST
    - Some time zones are a half-hour away from another (e.g. America/St. John's)
- The `lubridate` package can make all this much easier.

Creating date and date-time objects i

- We will often find dates in datasets
    - E.g. enrollment date, birthday, etc.
- However, these are often written in different formats.
    - E.g. 2021/09/17, September 17th 2021
- `lubridate` has three functions that can parse dates from strings if you specify the order of day (`d`), month (`m`), year (`y`).

```
library(lubridate)
ymd(”2017-01-31”)
```

```
## [1] ”2017-01-31”
```

Creating date and date-time objects ii

```
mdy(”January 31st, 2017”)
```

```
## [1] ”2017-01-31”
```

```
dmy(”31-Jan-2017”)
```

```
## [1] ”2017-01-31”
```

- Similarly, if you also have time information, `lubridate` can parse the string if you specify hour (`h`), minute (`m`), second (`s`).

Creating date and date-time objects iii

```
ymd_hms(”2017-01-31 20:11:59”)
```

```
## [1] ”2017-01-31 20:11:59 UTC”
```

```
mdy_hm(”01/31/2017 08:01”)
```

```
## [1] ”2017-01-31 08:01:00 UTC”
```

- Some datasets will also contain each separate component.

Creating date and date-time objects iv

```
library(tidyverse)
library(nycflights13)

flights %>%
  select(year, month, day, hour, minute)
```

```
## # A tibble: 336,776 x 5
##      year month   day  hour minute
##     <int> <int> <int> <dbl>  <dbl>
##  1  2013     1     1     5     15
##  2  2013     1     1     5     29
```

Creating date and date-time objects v

```
##  3  2013     1     1     5    40
##  4  2013     1     1     5    45
##  5  2013     1     1     6     0
##  6  2013     1     1     5    58
##  7  2013     1     1     6     0
##  8  2013     1     1     6     0
##  9  2013     1     1     6     0
## 10  2013     1     1     6     0
## # ... with 336,766 more rows
```

Creating date and date-time objects vi

```
flights %>%
  select(year, month, day, hour, minute) %>%
  mutate(departure = make_datetime(year, month, day,
                                   hour, minute))
```

```
## # A tibble: 336,776 x 6
## year month day hour minute departure
## <int> <int> <int> <dbl> <dbl> <dttm>
## 1 2013 1 1 5 15 2013-01-01 05:15:00
## 2 2013 1 1 5 29 2013-01-01 05:29:00
## 3 2013 1 1 5 40 2013-01-01 05:40:00
```

Creating date and date-time objects vii

```
## 4 2013 1 1 5 45 2013-01-01 05:45:00
## 5 2013 1 1 6 0 2013-01-01 06:00:00
## 6 2013 1 1 5 58 2013-01-01 05:58:00
## 7 2013 1 1 6 0 2013-01-01 06:00:00
## 8 2013 1 1 6 0 2013-01-01 06:00:00
## 9 2013 1 1 6 0 2013-01-01 06:00:00
## 10 2013 1 1 6 0 2013-01-01 06:00:00
## # ... with 336,766 more rows
```

- Finally, you can also specify a date or date-time as an offset in days or seconds with respect to the “Unix Epoch”, which is January 1st 1970.

Creating date and date-time objects viii

```
# Offset in seconds
as_datetime(60 * 60 * 10)
```

```
## [1] ”1970-01-01 10:00:00 UTC”
```

```
# Offset in days
as_date(365 * 10 + 2)
```

```
## [1] ”1980-01-01”
```

Exercises

1. What happens if you run the following code?

```
ymd(c(”2010-10-10”, ”bananas”))
```

2. Look at the help page for the function `today`. What does `tzone` argument do?
3. Convert each of the following strings to date objects:

```
d1 <- ”January 1, 2010”
d2 <- ”2015-Mar-07”
d3 <- ”06-Jun-2017”
d4 <- c(”August 19 (2015)”, ”July 1 (2015)”)
d5 <- ”12/30/14” # Dec 30, 2014
```

Solutions i

Exercise 1

```
ymd(c(”2010-10-10”, ”bananas”))
```

```
## Warning: 1 failed to parse.
```

```
## [1] ”2010-10-10” NA
```

We can the value NA and R prints a warning.

Solutions ii

Exercise 2

The argument `tzone` changes the time zone in which the date is printed. This could potentially give us tomorrow's date if we take a different time zone (e.g. `tzone = ”Pacific/Auckland”`).

Solutions iii

Exercise 3

```
mdy(”January 1, 2010”)
```

```
## [1] ”2010-01-01”
```

```
ymd(”2015-Mar-07”)
```

```
## [1] ”2015-03-07”
```

Solutions iv

Exercise 3 (cont’d)

```
dmy(”06-Jun-2017”)
```

```
## [1] ”2017-06-06”
```

```
mdy(c(”August 19 (2015)”, ”July 1 (2015)”))
```

```
## [1] ”2015-08-19” ”2015-07-01”
```

```
mdy(”12/30/14”)
```

```
## [1] ”2014-12-30”
```

Date-time components

- `lubridate` provides functions to extract components from dates
    - `year()`, `day()`, `second()`, etc.
- Two interesting components:
    - `month()`: Can output either the number or name of the month.
    - `wday()`: Can output either the number of name of the day of the week.

Examples

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

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

```
wday(now(), label = TRUE, abbr = FALSE)
```

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

Date-time arithmetic i

```
# Simple arithmetic
today() + years(1)
```

```
## [1] ”2022-09-24”
```

```
today() + weeks(4)
```

```
## [1] ”2021-10-22”
```

Date-time arithmetic ii

```
# But be careful!
ymd(”2021/01/31”) + months(1)
```

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

- Notice the difference between `year()`, which extracts the year component, and `years()`, which adds (or subtracts) years from a given date.
- We can also round a date up or down to the nearest year, month, etc.

Date-time arithmetic iii

```
ceiling_date(today(), unit = ”month”)
```

```
## [1] ”2021-10-01”
```

```
floor_date(today(), unit = ”year”)
```

```
## [1] ”2021-01-01”
```

Time spans i

- The package `lubridate` measures time spans in three different ways:
    - **durations**, which represent an exact number of seconds.
    - **periods**, which represent human units like weeks and months.
    - **intervals**, which represent a starting and ending point.
- Functions like `months()` and `years()` refer to *periods*.
- *Durations* can be constructed using the following functions
    - `dseconds()`, `ddays()`, `dweeks()`, etc.

Time spans ii

```
# Careful: durations are an exact number of seconds!
one_pm <- ymd_hms(”2016-03-12 13:00:00”,
                  tz = ”America/New_York”)


one_pm
```

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

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

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

Time spans iii

- Notice the change in time zones.
- Finally, *intervals* have a starting and an ending time point.
    - They can be constructed from two date-time objects using the operator `%--%`

```r
next_year <- today() + years(1)
today() %--% next_year
```

```
## [1] 2021-09-24 UTC--2022-09-24 UTC
```

- You can also determine how many *durations* fall inside an *interval* using division.

Time spans iv

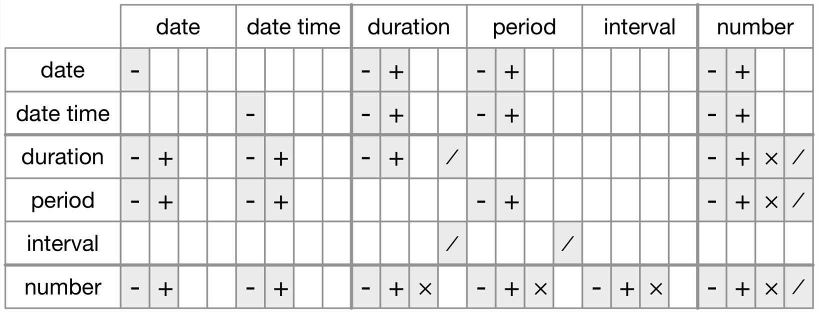
```
(today() %--% next_year)/ddays(1)
```

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

```
(today() %--% next_year)/dweeks(1)
```

```
## [1] 52.14286
```

Time spans v



	date				date time				duration				period				interval				number			
date	-								-	+			-	+							-	+		
date time					-				-	+			-	+							-	+		
duration	-	+			-	+			-	+		/									-	+	×	/
period	-	+			-	+							-	+							-	+	×	/
interval												/				/								
number	-	+			-	+			-	+	×		-	+	×		-	+	×		-	+	×	/

Permitted arithmetic operations between the different data types (*R for Data Science*).

Exercises

1. Create a vector of dates giving the first day of every month in 2015.
2. Create a vector of dates giving the first day of every month in the current year (your code should also work after January 1st).
3. Write a function that given your birthday (as a date), returns how old you are in years.

Solutions i

Exercise 1

```
ymd(”20150101”) + months(0:11)
```

```
## [1] ”2015-01-01” ”2015-02-01” ”2015-03-01”
”2015-04-01” ”2015-05-01”
## [6] ”2015-06-01” ”2015-07-01” ”2015-08-01”
”2015-09-01” ”2015-10-01”
## [11] ”2015-11-01” ”2015-12-01”
```

Solutions ii

Exercise 2

The main difference is that we need to start at January 1st of the current year. We can round down today’s date.

```
floor_date(today(), unit = ”years”) + months(0:11)
```

```
## [1] ”2021-01-01” ”2021-02-01” ”2021-03-01”
”2021-04-01” ”2021-05-01”
## [6] ”2021-06-01” ”2021-07-01” ”2021-08-01”
”2021-09-01” ”2021-10-01”
## [11] ”2021-11-01” ”2021-12-01”
```

Solutions iii

Exercise 2 (cont’d)

```
# Also valid
first_day <- ymd(paste0(year(today()), ”0101”))
first_day + months(0:11)
```

```
## [1] ”2021-01-01” ”2021-02-01” ”2021-03-01”
”2021-04-01” ”2021-05-01”
## [6] ”2021-06-01” ”2021-07-01” ”2021-08-01”
”2021-09-01” ”2021-10-01”
## [11] ”2021-11-01” ”2021-12-01”
```

Solutions iv

Exercise 3

```
calculate_age <- function(date_birth) {
  age <- (date_birth %--% today())/dyears(1)
  # Round down
  return(floor(age))
}

calculate_age(ymd(”20010911”))
```

```
## [1] 20
```

Time zones i

- In every day life, time zones are usually described in ambiguous terms.
    - E.g. Both Winnipeg and Saskatoon are on Central Time, but there is a one hour difference in the summer.
    - E.g. BST could either refer to “British Summer Time” or “Bangladesh Standard Time”
- There are two standardized ways of discussing time zones:
    - As offsets of Universal Coordinated Time (UTC)
    - Using IANA time zones.
- For example:
    - Winnipeg in the summer is UTC -5; in the winter is UTC -6.

Time zones ii

    - The IANA code is “America/Winnipeg”.
- The latter standardization is preferred, as it automatically takes into account changes in time (e.g. changing offsets).
- With `lubridate`, there are essentially two operations with time zones:
    - `with_tz()`: Keep the same “instant in time” but convert to a different time zone.
    - `force_tz()`: Change the underlying “instant in time”. Useful when the date-time is labelled with the wrong time zone.

Examples i

```
# Setting the time zone attribute
x1 <- ymd_hms(”2021-09-20 16:00:00”,
              tz = ”America/Winnipeg”)
x2 <- ymd_hms(”2021-09-20 23:00:00”,
              tz = ”Europe/Copenhagen”)
x3 <- ymd_hms(”2021-09-21 9:00:00”,
              tz = ”Pacific/Auckland”)
```

Examples ii

```
# These are all the same instant in time
x1 - x2
```

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

```
x1 - x3
```

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

Examples iii

```
# Convert
x1
```

```
## [1] ”2021-09-20 16:00:00 CDT”
```

```
with_tz(x1, tzone = ”America/St_Johns”)
```

```
## [1] ”2021-09-20 18:30:00 NDT”
```

```
force_tz(x1, tzone = ”America/Edmonton”)
```

```
## [1] ”2021-09-20 16:00:00 MDT”
```