2025 CCADD Winter Internship - Book Reading Session

Chapter 17: Dates and Times

https://r4ds.hadley.nz/datetimes

0. Introduction

Objectives

1. **Create** date and datetime objects
2. Work with **datetime components**
3. Perform **arithmetics** on time
4. Recognize ways to deal with **timezones**

In [1]:
```
# Import package
library(tidyverse)
library(nycflights13)
```

```
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.
0 ──
✔ dplyr     1.1.4     ✔ readr     2.1.5
✔ forcats   1.0.0     ✔ stringr   1.5.1
✔ ggplot2   3.5.1     ✔ tibble    3.2.1
✔ lubridate 1.9.4     ✔ tidyr     1.3.1
✔ purrr     1.0.2
── Conflicts ────────────────────────────────────────── tidyverse_conflicts
() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all c
onflicts to become errors
```

In [2]:
```
# information about the dataset
head(flights,3)
```

A ti

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time
<int>	<int>	<int>	<int>	<int>	<dbl>	<int>	<int>
2013	1	1	517	515	2	830	819
2013	1	1	533	529	4	850	830
2013	1	1	542	540	2	923	850

Vocabularies

- Types of date/time objects:
  1. date = `<date>` : Stores the date (year, month, day). No time information.
  2. time = `<hms>` : Stores the time of day (hours, minutes, seconds). No date information.
  3. datetime = `<dttm>` = POSIXct: Includes both date and time information. Time zones can be applied.
- Important to be aware of timezones

In [3]:
```
today()
class(today())
```

2025-01-31

'Date'

In [4]:
```
now() # time with timestamp; local time zone
class(now())
```

```
 [1] "2025-01-31 13:15:17 KST"
'POSIXct' · 'POSIXt'
```

- `dttm` : date time class that some R packages(e.g., `lubridate`) utilize.
- `POSIXt` : the default date-time class of R, more broadly utilized.

1. Creating Date and Time Objects

`lubridate` functions

`ymd_hms()`

- By default it puts everything in UTC

In [5]:
```
ymd_hms("2025-02-28 15:39:29")
```

```
[1] "2025-02-28 15:39:29 UTC"
```

`make_date()` & `make_datetime()`:

- if you have in your data separate columns for year month day hour minute, etc.
- these will turn into a datetime or date

In [6]:
```
head(flights,3)
```

A ti

year <int>	month <int>	day <int>	dep_time <int>	sched_dep_time <int>	dep_delay <dbl>	arr_time <int>	sched_arr_time <int>
2013	1	1	517	515	2	830	819
2013	1	1	533	529	4	850	830
2013	1	1	542	540	2	923	850

?flights

In `flights` dataset,

- `year` , `month` , `day` : Date of departure
- `hour` , `minute` : Time of scheduled departure broken into hour and minutes.

In [7]:
```
flights |>
    select(year, month, day, hour, minute) |>
    mutate(
        departure = make_datetime(year, month, day, hour, minute),
        dep_date = make_date(year, month, day)
    ) |>
    head(3)

#departure = <dttm> & dep_date = <date>
```

A tibble: 3 × 7

year	month	day	hour	minute	departure	dep_date
<int>	<int>	<int>	<dbl>	<dbl>	<dttm>	<date>
2013	1	1	5	15	2013-01-01 05:15:00	2013-01-01
2013	1	1	5	29	2013-01-01 05:29:00	2013-01-01
2013	1	1	5	40	2013-01-01 05:40:00	2013-01-01

`as_date()` & `as_datetime()`

- standardized way to store times, when things will be given as big number which is days or seconds, which is called unix epoch
- Unix Epoch: 1970-01-01 00:00:00 UTC
- `as_date()` : days
- `as_datetime()` : seconds

In [8]:
```
as_datetime(365)
```

```
[1] "1970-01-01 00:06:05 UTC"
```

Getting component

- `year()` , `month()` , `hour()` , `minute()` , `second()`
- `day()` :
  - `mday()` : day of month; same with `day`
  - `yday()` : day of year
  - `wday()` : day of week

In [9]:
```
now()
```

```
[1] "2025-01-31 13:15:17 KST"
```

In [10]:
```
datetime_example <- ymd_hms("2025-03-28 15:39:29")
```

In [11]:
```
year(datetime_example)
mday(datetime_example)
```

```
yday(datetime_example) # 31 + 28  = 59
```

2025

28

87

In [12]:
```
wday(now()) # order of the level: ['Sun','Mon','Tue','Wed','Thu','Fri','Sat'
wday(now(), label = TRUE)
wday(now(), label = TRUE, abbr = FALSE)
```

6

Fri

▶ **Levels**:

Friday

▶ **Levels**:

2. Arithmetic on Time

Rounding datetimes

- types:
  - `round_date()`, `floor_date()`, `ceilling_date()`
- argument:
  - `<>_date(<datetime>, unit = "hour")`

In [13]:
```
datetime_example
print("=========")
round_date(datetime_example, unit = "hour")
round_date(datetime_example, unit = "min")
round_date(datetime_example, unit = "day")
```

```
[1] "2025-03-28 15:39:29 UTC"
[1] "========="
[1] "2025-03-28 16:00:00 UTC"
[1] "2025-03-28 15:39:00 UTC"
[1] "2025-03-29 UTC"
```

Updating datetimes

In [14]:
```
hour(datetime_example)
```

15

In [15]:
```
hour(datetime_example) <- hour(datetime_example) + 1
hour(datetime_example)
datetime_example
```

16

```
[1] "2025-03-28 16:39:29 UTC"
```

In [16]:
```
update(datetime_example, years = 2024, months = 5, mdays = 30)
```

```
[1] "2024-05-30 16:39:29 UTC"
```

Concepts of time spans

- **Durations**: exact number of seconds elapsed
- **Periods**: human units (like days)
- **Intervals**: start and end datetime

Duration

In [17]:
```
ny_age <- today() - ymd("2004-09-30")
ny_age
```

```
Time difference of 7428 days
```

In [18]:
```
as.duration(ny_age) #lubridate always store it in seconds
```

641779200s (~20.34 years)

In [19]:
```
#duration days are calculated in num of seconds
ddays(0:3)
dhours(2)
ddays(0:3) + dhours(2)
```

0s · 86400s (~1 days) · 172800s (~2 days) · 259200s (~3 days)

7200s (~2 hours)

7200s (~2 hours) · 93600s (~1.08 days) · 180000s (~2.08 days) · 266400s (~3.08 days)

Period

In [20]:
```
days(1)
```

1d 0H 0M 0S

In [21]:
```
ymd_hms(now(), tz = "Asia/Seoul")
ymd_hms(now(), tz = "Asia/Seoul") + days(1)
```

```
[1] "2025-01-31 13:15:17 KST"
[1] "2025-02-01 13:15:17 KST"
```

Interval

In [22]:
```
y2023 <- ymd("2023-01-01") %--% ymd("2024-01-01")
y2023
y2023 / days(1) # how many period (days) in 2023
```

2023-01-01 UTC--2024-01-01 UTC

365

%--% = to

In [23]:
```
y2024 <- ymd("2024-01-01") %--% ymd("2025-01-01")
y2024
y2024 / days(1)
```

2024-01-01 UTC--2025-01-01 UTC

366

In [24]:
```
class(y2024)
```

'Interval'

In [25]:
```
y2024 / days(1)
y2024 / ddays(1)
y2024 / dyears(1)
```

366

366

1.00205338809035

Summary

In [26]:
```
start_time <- ymd("2025-01-01")
```

In [27]:
```
# Duration: Fixed in seconds >> Calculates exact difference in "seconds" aft

dmonths(12)
start_time + dyears(1)
```

31557600s (~1 years)

[1] "2026-01-01 06:00:00 UTC"

In [28]:
```
# Period: Calculates the date exactly one year later based on the calendar
years(1)
start_time + years(1)
```

1y 0m 0d 0H 0M 0S

2026-01-01

In [29]:
```
# Interval: Compares the duration between two dates
time_interval <- start_time %--% (start_time + years(1))
as.duration(time_interval) # 실제 초 단위 계산
```

31536000s (~52.14 weeks)

NOTE

- **Duration** (in years)
  - `lubridate` calculates time based on an average year length of 365.25 days (accounting for leap years).
  - 1 year = approximately 365.25 days, 1 day = 86,400 seconds.
  - `dmonths(12)` = 365.25 × 86,400 = 31,557,600 seconds.

- **Interval** (in weeks)
  - Calculated based on actual calendar dates → the number of days between the start and end date.
  - `%--%` creates an interval, and calling `as.duration()` returns the exact number of seconds between the start and end dates.
  - Since 2025 is not a leap year, exactly 365 days (31,536,000 seconds) are used.

3. Time Zone

- The package, `clock` for backend (easier to update)
- Definition of time zone is stil changing:
  - Daylight saving

In [30]:
```
internship_start_korea <- ymd_hms("2025-01-02 09:00:00", tz = "Asia/Seoul")
```

In [31]:
```
internship_start_korea
```

```
[1] "2025-01-02 09:00:00 KST"
```

Converting time zone

In [32]:
```
# Keep the definition of the time but only fix the time zone
force_tz(internship_start_korea, "Europe/Paris")
```

```
[1] "2025-01-02 09:00:00 CET"
```

In [33]:
```
with_tz(internship_start_korea, "Europe/Paris")
```

```
[1] "2025-01-02 01:00:00 CET"
```

In [34]:
```
with_tz(internship_start_korea, "America/Chicago")
with_tz(internship_start_korea, "Australia/Sydney")
```

```
[1] "2025-01-01 18:00:00 CST"
[1] "2025-01-02 11:00:00 AEDT"
```

Daylight saving time

In [35]:
```
internship_starts <- internship_start_korea + weeks(9:20)
internship_starts #|> hour()
```

```
 [1] "2025-03-06 09:00:00 KST" "2025-03-13 09:00:00 KST"
 [3] "2025-03-20 09:00:00 KST" "2025-03-27 09:00:00 KST"
 [5] "2025-04-03 09:00:00 KST" "2025-04-10 09:00:00 KST"
 [7] "2025-04-17 09:00:00 KST" "2025-04-24 09:00:00 KST"
 [9] "2025-05-01 09:00:00 KST" "2025-05-08 09:00:00 KST"
[11] "2025-05-15 09:00:00 KST" "2025-05-22 09:00:00 KST"
```

Europe/Rome starts DST on the last Sunday of March, moving one hour ahead

In [36]:
```
internship_starts |> with_tz("Europe/Rome") |> hour()
```

1 · 1 · 1 · 1 · 2 · 2 · 2 · 2 · 2 · 2 · 2 · 2

America/Chicago starts DST on the second Sunday of March

- Usually CST(Central Standard Time, UTC-6)
- While DST, CDT(Central Daylight Time, UTC-5)

In [37]:
```
internship_starts_chicago <- with_tz(internship_start_korea, "America/Chicag
internship_starts_chicago #|> hour()
```

```
 [1] "2025-03-05 18:00:00 CST" "2025-03-12 18:00:00 CDT"
 [3] "2025-03-19 18:00:00 CDT" "2025-03-26 18:00:00 CDT"
 [5] "2025-04-02 18:00:00 CDT" "2025-04-09 18:00:00 CDT"
 [7] "2025-04-16 18:00:00 CDT" "2025-04-23 18:00:00 CDT"
 [9] "2025-04-30 18:00:00 CDT" "2025-05-07 18:00:00 CDT"
[11] "2025-05-14 18:00:00 CDT" "2025-05-21 18:00:00 CDT"
```

In [38]:
```
internship_starts_chicago |> with_tz("Europe/Rome") |> hour()
```

1 · 0 · 0 · 0 · 1 · 1 · 1 · 1 · 1 · 1 · 1 · 1

UTC is not affected by DST

In [39]:
```
internship_starts_chicago |> with_tz("UTC") |> hour()
```

0 · 23 · 23 · 23 · 23 · 23 · 23 · 23 · 23 · 23 · 23 · 23

Australia/Sydney ends DST on the first Sunday of April

In [40]:
```
internship_starts_chicago |> with_tz("Australia/Sydney") |> hour()
```

11 · 10 · 10 · 10 · 10 · 9 · 9 · 9 · 9 · 9 · 9 · 9