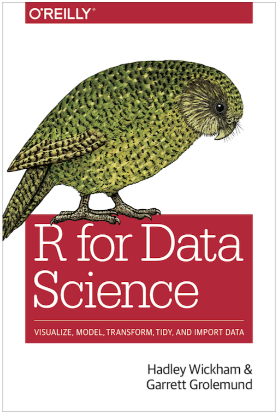
Socio-Informatics 348

Data Transformation
Dates and Times

Dr Lisa Martin

Department of Information Science
Stellenbosch University

Today's Reading


O'REILLY
R for Data Science
VISUALIZE, MODEL, TRANSFORM, TIDY, AND IMPORT DATA
Hadley Wickham &
Garrett Grolemund

R for Data Science, Chapter 17

Dates and Times

There are three types of date/time data that refer to an instant in time:

- A date. Tibbles print this as $< date >$.
- A time within a day. Tibbles print this as $< time >$.
- A date-time is a date plus a time: it uniquely identifies an instant in time (typically to the nearest second). Tibbles print this as $< dttm >$.
- Use the `lubridate` package to work with dates and times.
- Use functions `today()` and `now()` to get the current date or date-time.

```
today()
#> [1] "2025-09-17"
now()
#> [1] "2025-09-17 23:12:07 UTC"
```

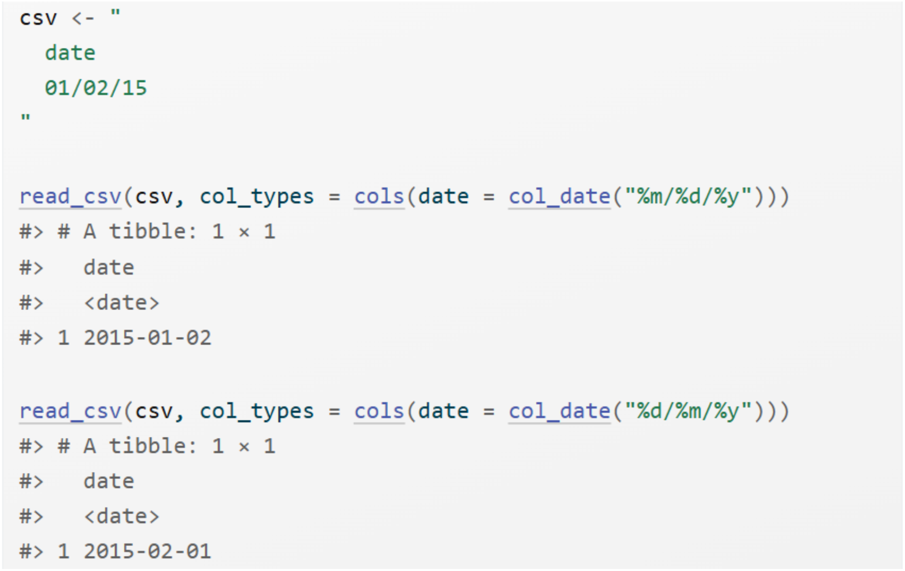
Creating date/time

During import

- If your CSV contains an ISO8601 date or date-time, `readr` will automatically parse it.
- ISO8601 is a standard way to represent dates and times: YYYY-MM-DD for dates, and YYYY-MM-DDTHH:MM:SS for date-times.
- You can also use `col_type = col_date()` or `col_datetime()` to specify that a column should be parsed as a date or date-time.
- R4DS Table 17.1: All date formats understood by readr

Creating date/time

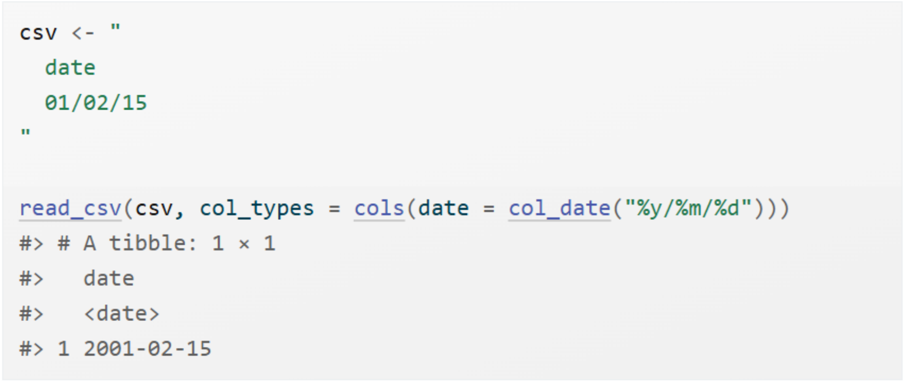
During import



```
csv <- "
  date
  01/02/15
"

read_csv(csv, col_types = cols(date = col_date("%m/%d/%y")))
#> # A tibble: 1 × 1
#>   date
#>   <date>
#> 1 2015-01-02

read_csv(csv, col_types = cols(date = col_date("%d/%m/%y")))
#> # A tibble: 1 × 1
#>   date
#>   <date>
#> 1 2015-02-01
```

Creating date/time

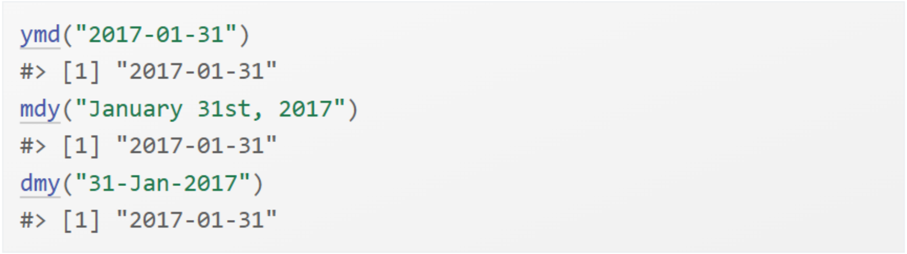
During import



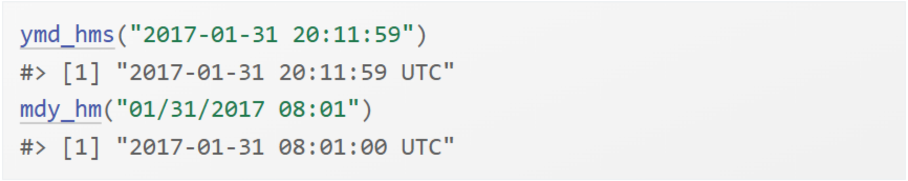
```
csv <- "
  date
  01/02/15
"

read_csv(csv, col_types = cols(date = col_date("%y/%m/%d")))
#> # A tibble: 1 × 1
#>   date
#>   <date>
#> 1 2001-02-15
```

Creating date/time

From strings: Lubridate functions



```
ymd("2017-01-31")
#> [1] "2017-01-31"
mdy("January 31st, 2017")
#> [1] "2017-01-31"
dmy("31-Jan-2017")
#> [1] "2017-01-31"
```



```
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"
```

Creating date/time

From individual components

```
flights |>
  select(year, month, day, hour, minute) |>
  mutate(departure = make_datetime(year, month, day, hour, minute))
#> # A tibble: 336,776 × 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
#> 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
#> # ℹ 336,770 more rows
```

Creating date/time

From other types

```
as_datetime(today())
#> [1] "2025-09-17 UTC"
as_date(now())
#> [1] "2025-09-17"
```

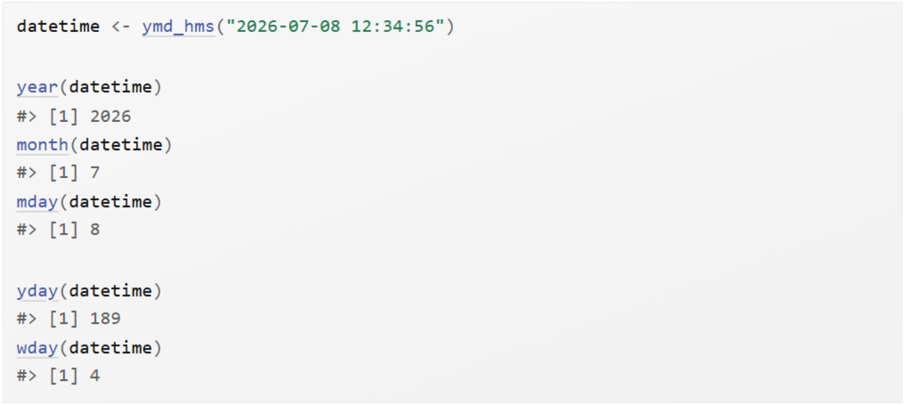
Numeric values:

- `as_datetime()` converts numeric values as seconds since 1970-01-01.
- `as_date()` converts numeric values as days since 1970-01-01.

```
as_datetime(60 * 60 * 10)
#> [1] "1970-01-01 10:00:00 UTC"
as_date(365 * 10 + 2)
#> [1] "1980-01-01"
```

Extracting components

Date components:



```
datetime <- ymd_hms("2026-07-08 12:34:56")

year(datetime)
#> [1] 2026
month(datetime)
#> [1] 7
mday(datetime)
#> [1] 8

yday(datetime)
#> [1] 189
wday(datetime)
#> [1] 4
```

Extracting components

Date components:

```
month(datetime, label = TRUE)
#> [1] Jul
#> 12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Sep < ... < Dec
wday(datetime, label = TRUE, abbr = FALSE)
#> [1] Wednesday
#> 7 Levels: Sunday < Monday < Tuesday < Wednesday < Thursday < ... < Saturday
```

Extracting components

Date components:

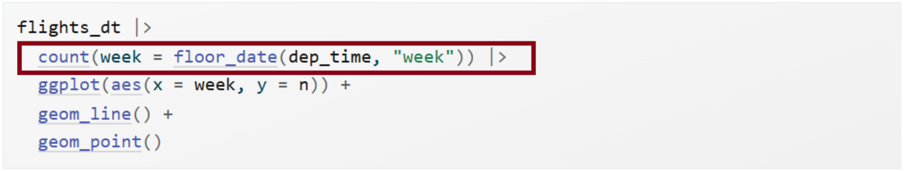
```r
flights_dt |>
  mutate(wday = wday(dep_time, label = TRUE)) |>
  ggplot(aes(x = wday)) +
  geom_bar()
```

```r
flights_dt |>
  mutate(minute = minute(dep_time)) |>
  group_by(minute) |>
  summarize(
    avg_delay = mean(dep_delay, na.rm = TRUE),
    n = n()
  ) |>
  ggplot(aes(x = minute, y = avg_delay)) +
  geom_line()
```

Extracting components

Rounding:

- floor_date(), ceiling_date(), round_date()

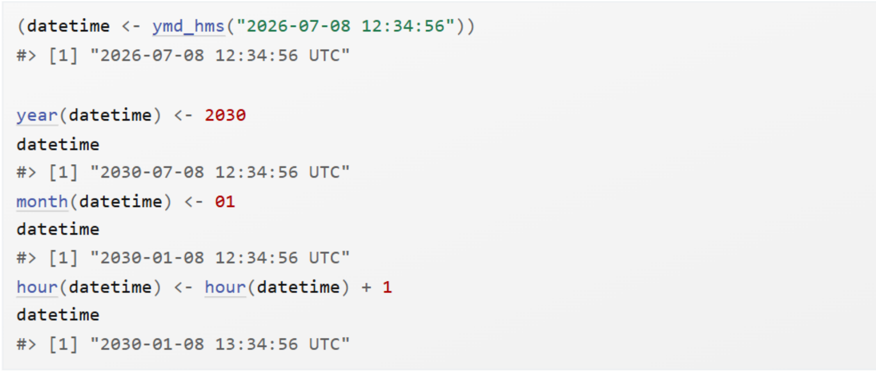


```
flights_dt |>
  count(week = floor_date(dep_time, "week")) |>
  ggplot(aes(x = week, y = n)) +
  geom_line() +
  geom_point()
```

Extracting components

Modifying components:

- Directly assign a new value to a component: `year(x) <- 2024`



```
(datetime <- ymd_hms("2026-07-08 12:34:56"))
#> [1] "2026-07-08 12:34:56 UTC"

year(datetime) <- 2030
datetime
#> [1] "2030-07-08 12:34:56 UTC"
month(datetime) <- 01
datetime
#> [1] "2030-01-08 12:34:56 UTC"
hour(datetime) <- hour(datetime) + 1
datetime
#> [1] "2030-01-08 13:34:56 UTC"
```

Extracting components

Modifying components:

- Use `update()` to modify multiple components at once:

```
update(datetime, year = 2030, month = 2, mday = 2, hour = 2)
#> [1] "2030-02-02 02:34:56 UTC"
```

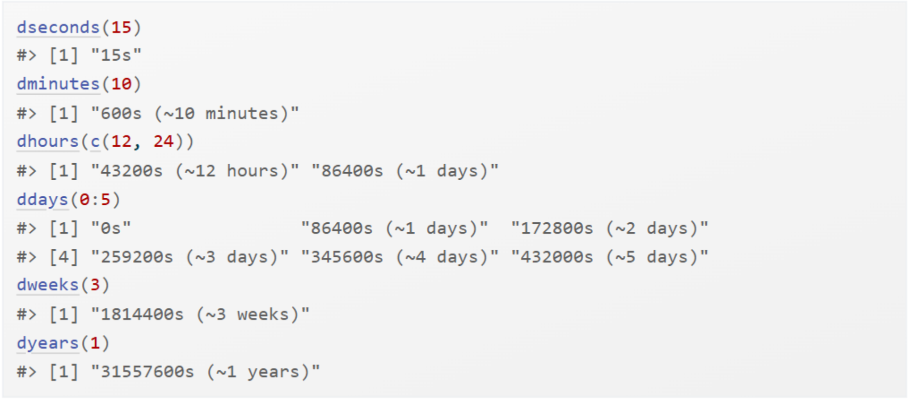
- Note that if you set a component to a
  value that is out of range, it will roll over to the next largest component.

```
update(ymd("2023-02-01"), mday = 30)
#> [1] "2023-03-02"
update(ymd("2023-02-01"), hour = 400)
#> [1] "2023-02-17 16:00:00 UTC"
```

Time Spans

Durations:

- A duration is an exact number of seconds.
- Convert to a duration using `as.duration()` or...



```
dseconds(15)
#> [1] "15s"
dminutes(10)
#> [1] "600s (~10 minutes)"
dhours(c(12, 24))
#> [1] "43200s (~12 hours)" "86400s (~1 days)"
ddays(0:5)
#> [1] "0s"                 "86400s (~1 days)"  "172800s (~2 days)"
#> [4] "259200s (~3 days)" "345600s (~4 days)" "432000s (~5 days)"
dweeks(3)
#> [1] "1814400s (~3 weeks)"
dyears(1)
#> [1] "31557600s (~1 years)"
```

Time Spans

Arithmetic with durations:

```
2 * dyears(1)
#> [1] "63115200s (~2 years)"
dyears(1) + dweeks(12) + dhours(15)
#> [1] "38869200s (~1.23 years)"
```

```
tomorrow <- today() + ddays(1)
last_year <- today() - dyears(1)
```

Time Spans

Periods:

- Uses "human times" like weeks, months, years.

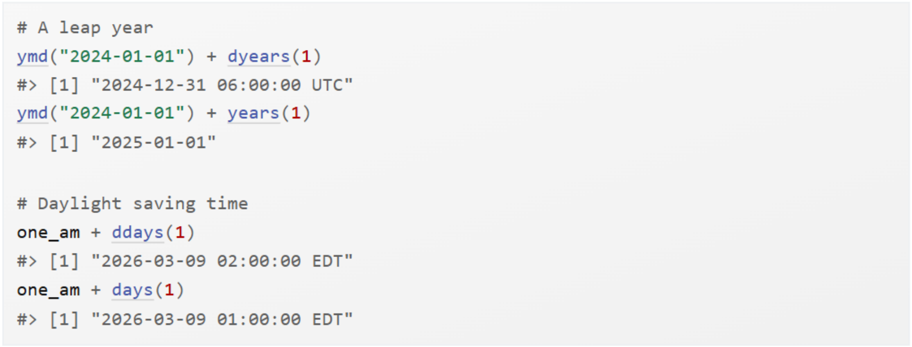
```
hours(c(12, 24))
#> [1] "12H 0M 0S" "24H 0M 0S"
days(7)
#> [1] "7d 0H 0M 0S"
months(1:6)
#> [1] "1m 0d 0H 0M 0S" "2m 0d 0H 0M 0S" "3m 0d 0H 0M 0S" "4m 0d 0H 0M 0S"
#> [5] "5m 0d 0H 0M 0S" "6m 0d 0H 0M 0S"
```

Arithmetic with periods:

```
10 * (months(6) + days(1))
#> [1] "60m 10d 0H 0M 0S"
days(50) + hours(25) + minutes(2)
#> [1] "50d 25H 2M 0S"
```

Time Spans

Durations vs Periods:



```
# A leap year
ymd("2024-01-01") + dyears(1)
#> [1] "2024-12-31 06:00:00 UTC"
ymd("2024-01-01") + years(1)
#> [1] "2025-01-01"

# Daylight saving time
one_am + ddays(1)
#> [1] "2026-03-09 02:00:00 EDT"
one_am + days(1)
#> [1] "2026-03-09 01:00:00 EDT"
```

Time Spans

Intervals:

- Create by using %--% to create an interval between two date-times.

```
y2023 <- ymd("2023-01-01") %--% ymd("2024-01-01")
y2024 <- ymd("2024-01-01") %--% ymd("2025-01-01")

y2023
#> [1] 2023-01-01 UTC--2024-01-01 UTC
y2024
#> [1] 2024-01-01 UTC--2025-01-01 UTC
```