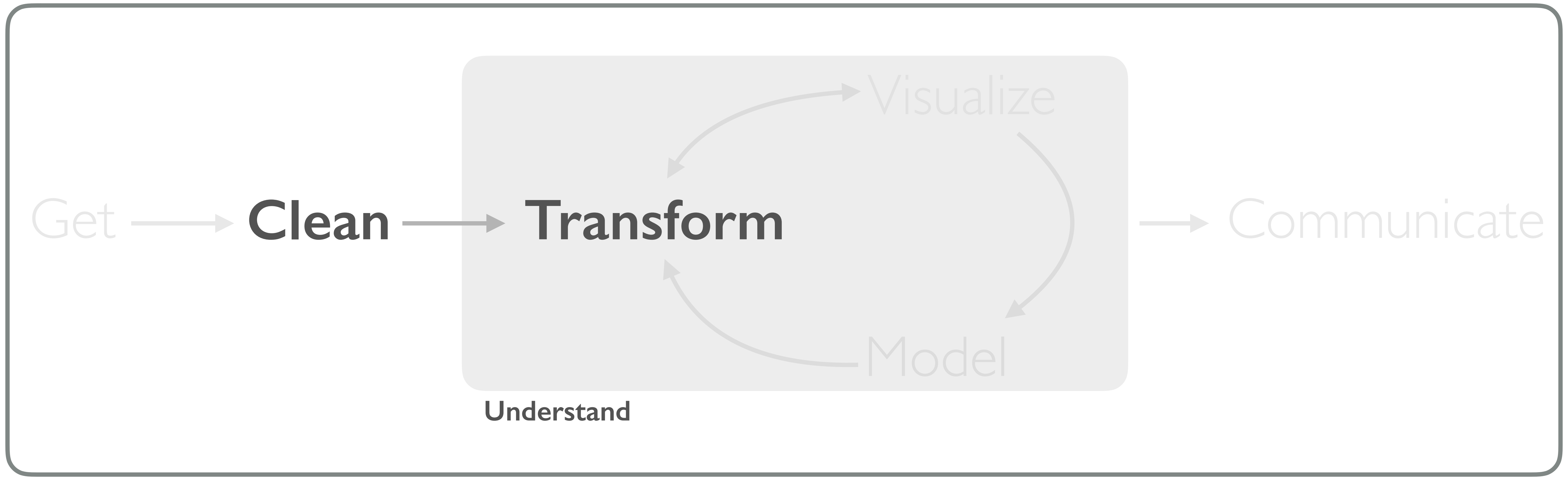
DATE-TIMES



Program

†A modified version of Hadley Wickham's analytic process

DATES

- Dates come in many different forms:
  - `2017/02/03`
  - `February 3, 2017`
  - `03-Feb-2017`
- Working with dates in R can be a bit convoluted and cumbersome
- The `lubridate` package allows us to easily handle/manipulate date-time variables

PARTS OF DATES

```
ymd_hms("2011-06-04 12:00:00", tz = "Pacific/Auckland")
```

Year

Month

Day

Hour

Minutes

Seconds

Time Zone

PARTS OF DATES

```
ymd_hms("2011-06-04 12:00:00", tz = "Pacific/Auckland")
```

Year

Month

Day

Hour

Minutes

Seconds

Time Zone

Plus, we can determine the day of the week and compute duration statistics

PREREQUISITES

PREREQUISITES

- Re-start your R session
  - **Windows:** Ctrl+Shift+F10
  - **Mac:** Command+Shift+F10
- Make sure your working directory is set to the course folder

PACKAGE PREREQUISITE

```
library(lubridate)
library(tidyverse)
```

DATA PREREQUISITE

```
nycflights13::weather
nycflights13::flights
nycflights13::airlines
```

CREATING DATES

CREATING DATES

```
ymd(“2017/02/03”)
[1] "2017-02-03"

mdy("February 3, 2017")
[1] “2017-02-03"

dmy("03-Feb-2017")
[1] "2017-02-03"
```

- 2017/02/03
- February 3, 2016
- 03-Feb-2016

The format of the date determines the function call. Easy to remember since the function call is based on order of year (y), month (m), and day (d)

CREATING DATES

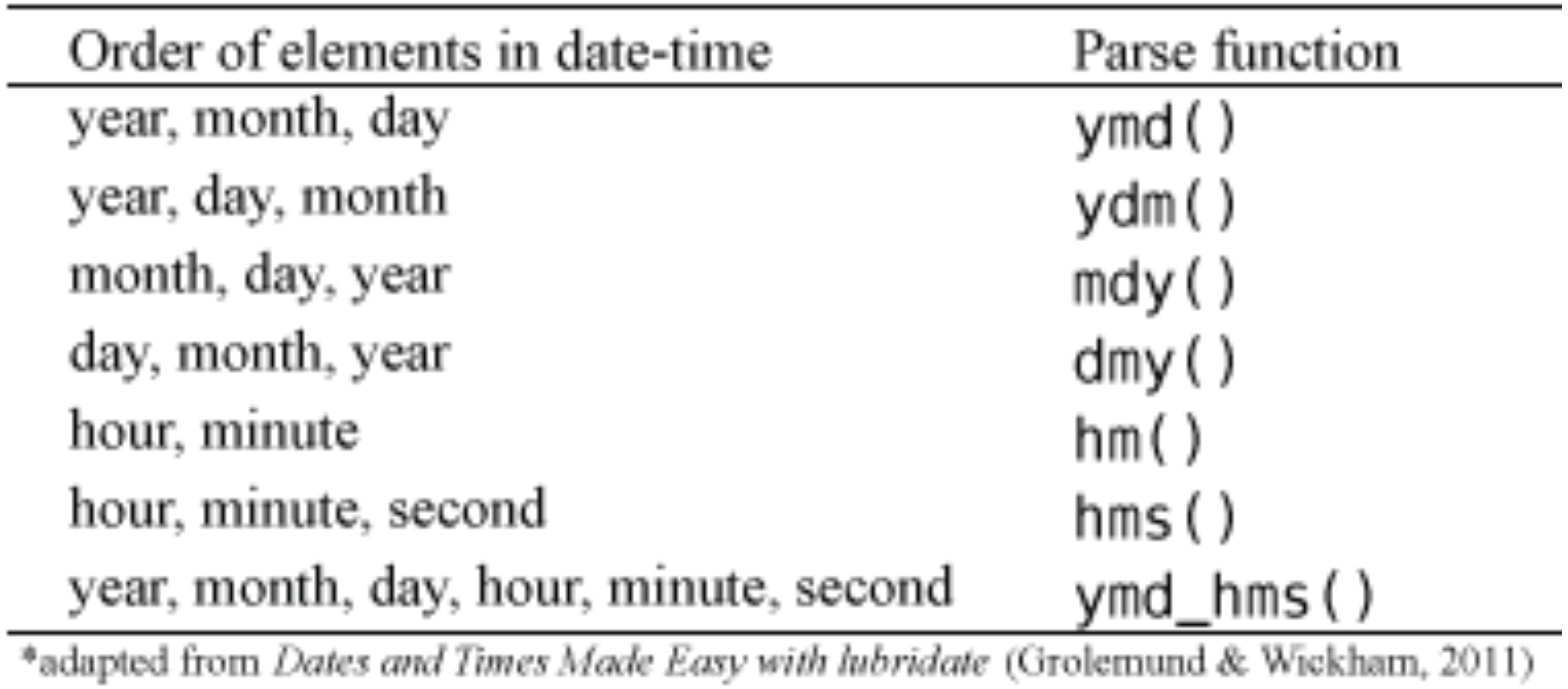
```
ymd_h("2017-02-03 2")
[1] "2017-02-03 02:00:00 UTC"

ymd_hm("2017-02-03 2:15")
[1] "2017-02-03 02:15:00 UTC"

ymd_hms("2017-02-03 2:15:45")
[1] "2017-02-03 02:15:45 UTC"
```

We can even extend this to account for time using

`_hms()`

CREATING DATES

`lubridates` *parsing functions*



Order of elements in date-time	Parse function
year, month, day	`ymd()`
year, day, month	`ydm()`
month, day, year	`mdy()`
day, month, year	`dmy()`
hour, minute	`hm()`
hour, minute, second	`hms()`
year, month, day, hour, minute, second	`ymd_hms()`

*adapted from *Dates and Times Made Easy with lubridate* (Grolemund & Wickham, 2011)

CREATING DATES

```
flights %>%
  select(year, month, day) %>%
  mutate(date = make_date(year, month, day))
# A tibble: 336,776 × 4
    year month   day       date
   <int> <int> <int>     <date>
1   2013     1     1 2013-01-01
2   2013     1     1 2013-01-01
3   2013     1     1 2013-01-01
4   2013     1     1 2013-01-01
5   2013     1     1 2013-01-01
6   2013     1     1 2013-01-01
7   2013     1     1 2013-01-01
8   2013     1     1 2013-01-01
9   2013     1     1 2013-01-01
10  2013     1     1 2013-01-01
```

Or we can create a date variable from separate year, month, day variables using

`make_date()`

CREATING DATES

```
flights %>%
  select(year, month, day) %>%
  mutate(date = make_date(year, month, day))
# A tibble: 336,776 × 4
    year month   day       date
   <int> <int> <int>     <date>
1   2013     1     1 2013-01-01
2   2013     1     1 2013-01-01
3   2013     1     1 2013-01-01
4   2013     1     1 2013-01-01
5   2013     1     1 2013-01-01
6   2013     1     1 2013-01-01
7   2013     1     1 2013-01-01
8   2013     1     1 2013-01-01
9   2013     1     1 2013-01-01
10  2013     1     1 2013-01-01
```

Or we can create a date variable from separate year, month, day variables using

`make_date()`

As with most functions these follow a specific order (`?make_date`)

CREATING DATES

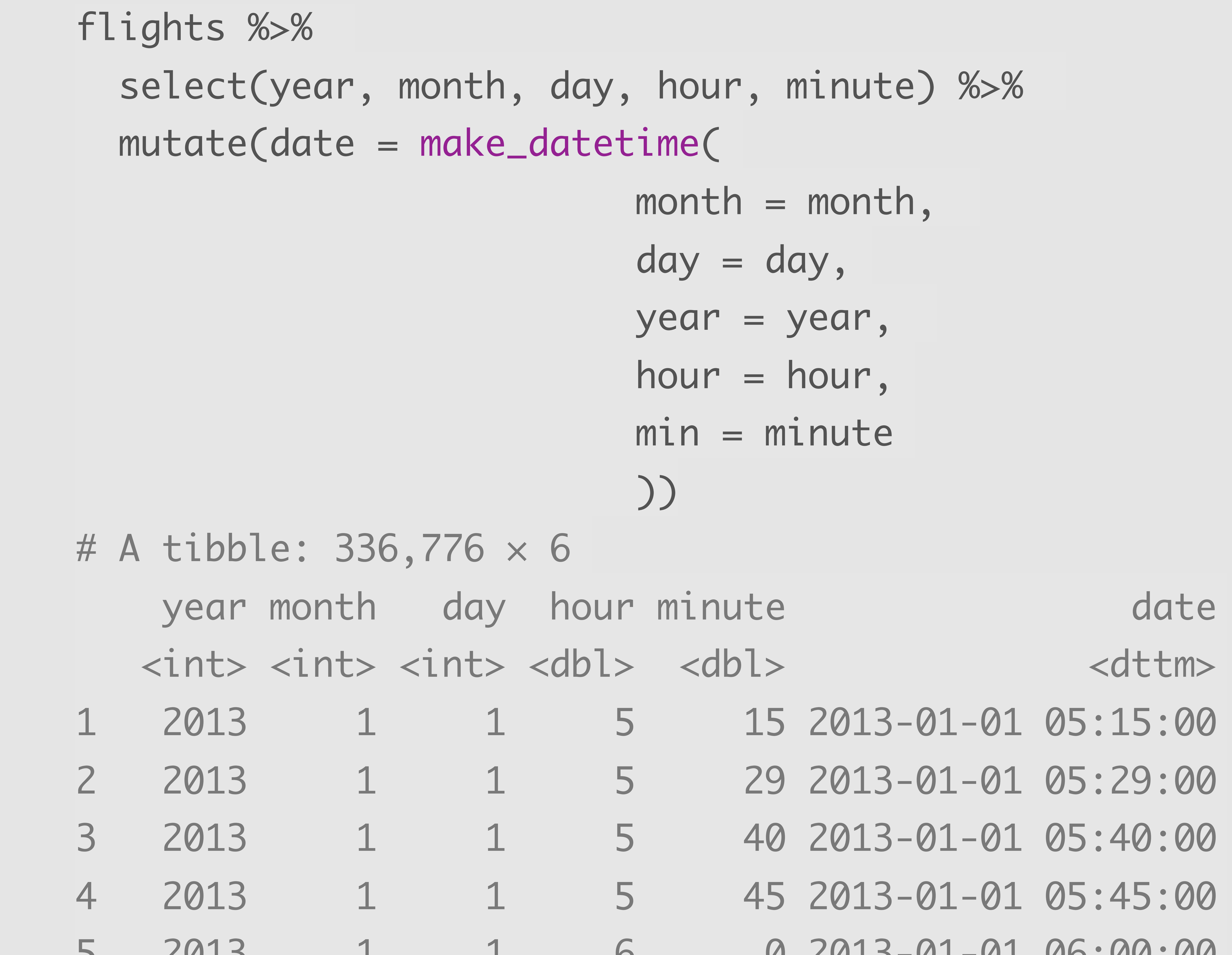
```
flights %>%
  select(year, month, day) %>%
  mutate(date = make_date(month = month,
                          day = day,
                          year = year))
# A tibble: 336,776 × 4
   year month   day       date
  <int> <int> <int>     <date>
1  2013     1     1 2013-01-01
2  2013     1     1 2013-01-01
3  2013     1     1 2013-01-01
4  2013     1     1 2013-01-01
5  2013     1     1 2013-01-01
6  2013     1     1 2013-01-01
7  2013     1     1 2013-01-01
8  2013     1     1 2013-01-01
```

Or we can create a date variable from separate year, month, day variables using

`make_date()`

Can insert arguments in any order as long as you define the parameters

CREATING DATES



```
flights %>%
  select(year, month, day, hour, minute) %>%
  mutate(date = make_datetime(
                        month = month,
                        day = day,
                        year = year,
                        hour = hour,
                        min = minute
                        ))
# A tibble: 336,776 × 6
    year month   day  hour minute                date
   <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
```

We can extend this to include time components with

`make_datetime()`

YOUR TURN!

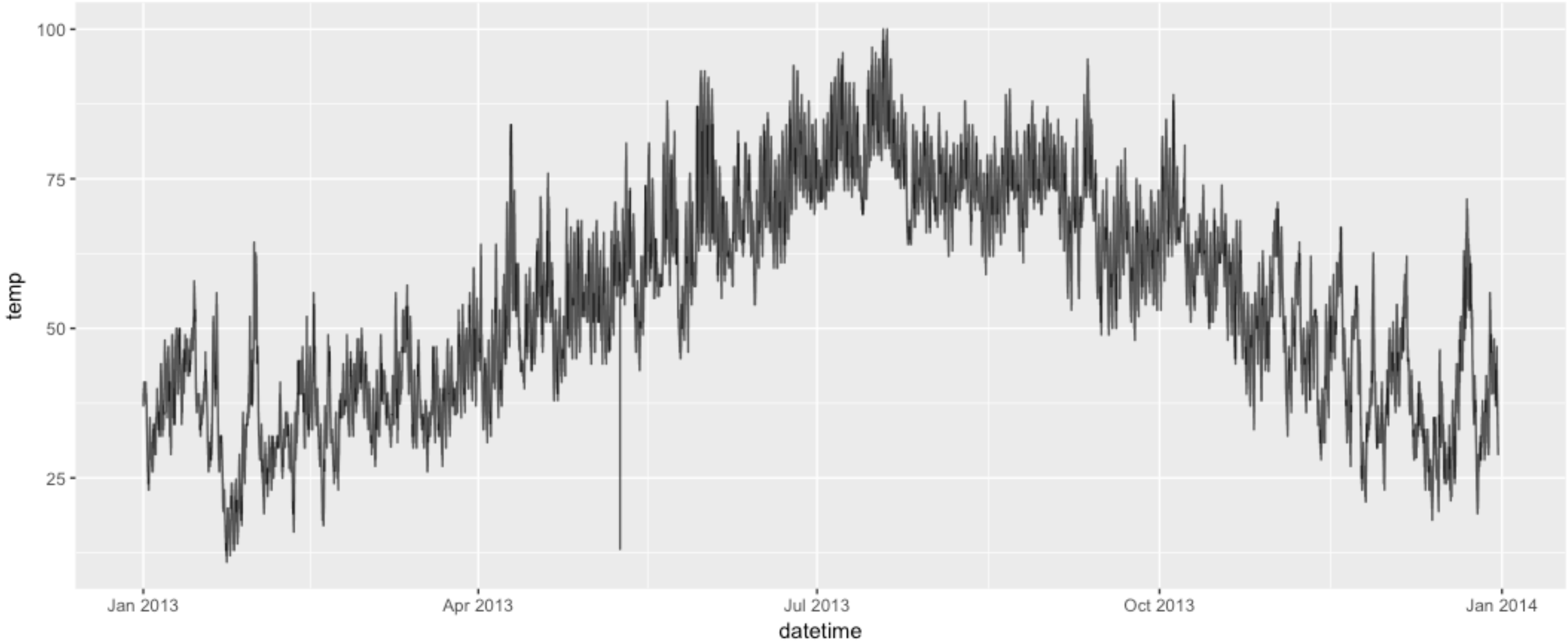
1. *Using the* **`nycflights13::weather`** *data, create a new "datetime" variable that combines the* **`year`**, **`month`**, **`day`**, *and* **`hour`** *variables.*

2. *Plot temp on the y-axis and date time on the x-axis*

3. *Can you spot the abnormally high and low temps?*

SOLUTION

```
weather %>%
  mutate(datetime = make_datetime(year, month, day, hour)) %>%
  ggplot(aes(datetime, temp)) +
  geom_line(alpha = .7)
```

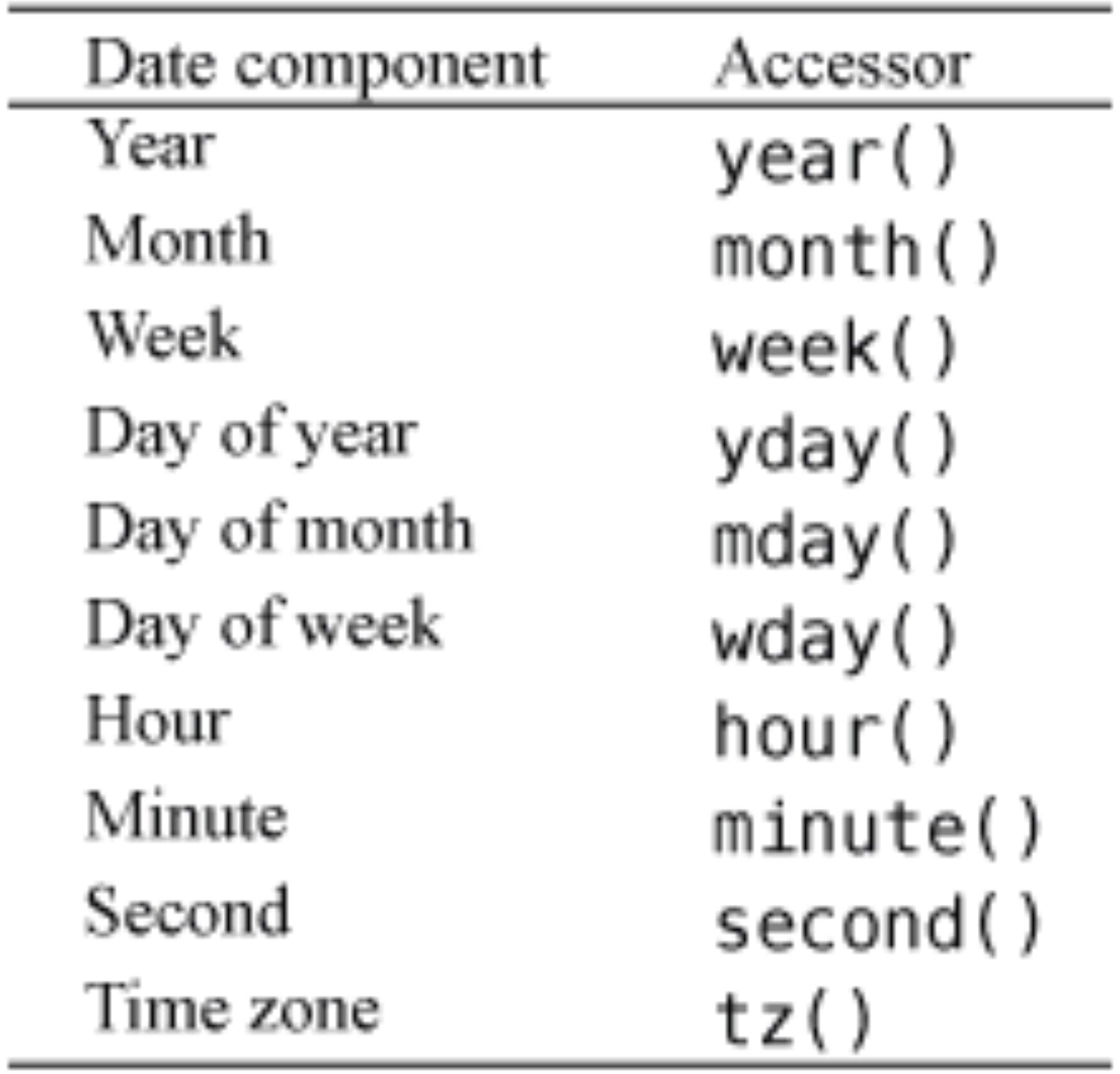


EXTRACTING COMPONENTS

EXTRACTING DATE-TIME COMPONENTS

```
datetime <- ymd_hms("2017-02-03 12:34:56”)

year(datetime)
[1] 2017

month(datetime)
[1] 2

mday(datetime)
[1] 3

yday(datetime)
[1] 34

wday(datetime, label = TRUE, abbr = FALSE)
[1] Friday
Levels: Sunday < Monday < Tuesday < Wednesday <
```

- Now that we have dates, how can we extract pieces of these dates?

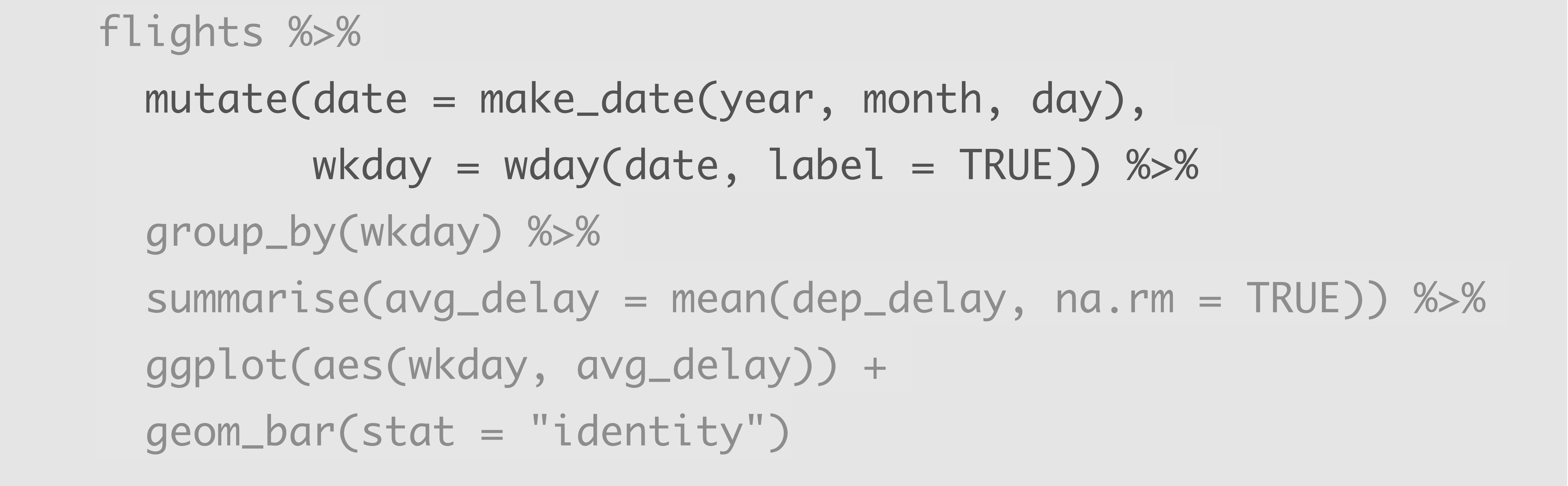


Date component	Accessor
Year	`year()`
Month	`month()`
Week	`week()`
Day of year	`yday()`
Day of month	`mday()`
Day of week	`wday()`
Hour	`hour()`
Minute	`minute()`
Second	`second()`
Time zone	`tz()`

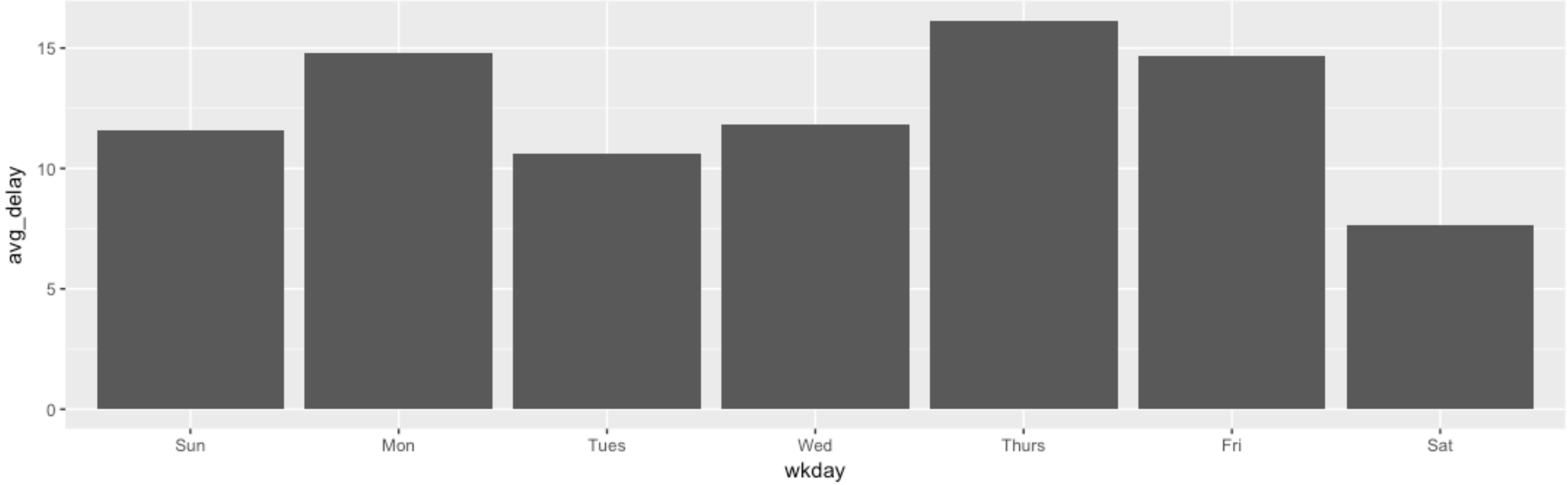
*adapted from *Dates and Times Made Easy with lubridate* (Grolemund & Wickham, 2011)

EXTRACTING DATE-TIME COMPONENTS

How can we use this? Here we use `wday()` to analyze the average departure delay by weekday.



```
flights %>%
  mutate(date = make_date(year, month, day),
         wkday = wday(date, label = TRUE)) %>%
  group_by(wkday) %>%
  summarise(avg_delay = mean(dep_delay, na.rm = TRUE)) %>%
  ggplot(aes(wkday, avg_delay)) +
  geom_bar(stat = "identity")
```



YOUR TURN!

1. *Using the* **`nycflights13::weather`** *data, can you identify the day of the week (i.e. Monday, Tuesday) that was the hottest day? What about the coldest day?*

SOLUTION

```
weather %>%
  mutate(date = make_date(year, month, day),
         wkday = wday(date, label = TRUE)) %>%
  select(date, wkday, temp) %>%
  slice(c(which.max(temp), which.min(temp)))
```

```
# A tibble: 2 × 3
        date wkday   temp
      <date> <ord>  <dbl>
1 2013-07-18 Thurs 100.04
2 2013-01-23   Wed  10.94
```

TIME SPANS

TIME SPANS

- Next you'll learn about how arithmetic with dates works, including subtraction, addition, and division. Along the way, you'll learn about three important classes that represent time spans:
  - **durations**: represent an exact number of seconds.
  - **periods**: represent human units like weeks and months.

DURATIONS

```
b_age <- today() - ymd(19800824)
b_age

as.duration(b_age)
# Time difference of 13288 days
```

- When you subtract two dates, you get a **difftime** object

DURATIONS

```
b_age <- today() - ymd(19800824)
b_age

as.duration(b_age)
# Time difference of 13288 days

dseconds(15)
dminutes(10)
dhours(c(12, 24))
ddays(0:5)
dweeks(3)
dyears(1)
```

- When you subtract two dates, you get a **difftime** object
- There are also a handful of **d_** functions that provide convenient **d**uration constructors

TRY THESE

DURATIONS

```
b_age <- today() - ymd(19800824)
b_age

as.duration(b_age)
# Time difference of 13288 days

dseconds(15)
dminutes(10)
dhours(c(12, 24))
ddays(0:5)
dweeks(3)

new_age <- b_age + dyears(1) + dweeks(3) +
ddays(4)
as.duration(new_age)
[1] "1181779200s (~37.45 years)"
```

- When you subtract two dates, you get a **difftime** object
- There are also a handful of **d_** functions that provide convenient duration constructors
- We can use these to perform mathematical operations on existing **difftime** objects

PERIODS

- However, durations represent exact number of seconds and do not consider daylight savings time or time zone differences.
- For more accurate calendar and clock representations use **periods**

TRY THESE

```
seconds(15)
minutes(10)
hours(c(12, 24))
days(0:5)
weeks(3)
months(1:6)
years(1)
```

PERIODS

```
seconds(15)
minutes(10)
hours(c(12, 24))
days(0:5)
weeks(3)
months(1:6)
years(1)

new_age <- ymd(19800824) + years(37)
as.duration(today() - new_age)
[1] "19526400s (~32.29 weeks)”

ymd_hms("2016-02-28 23:59:59") + minutes(1)
ymd_hms("2015-02-28 23:59:59") + minutes(1)
```

- However, durations represent exact number of seconds and do not consider daylight savings time or time zone differences.
- For more accurate calendar and clock representations use **periods**
- Periods get applied to **date-time** objects, not difftime objects

← TRY THESE

CHALLENGE

CHALLENGE

1. *If you look at the* **`arr_time`** *and* **`dep_time`** *variables in the* **`flights`** *data you'll notice that some flights arrive before they depart. These are overnight flights.*

2. *Can you figure out a way to adjust the* **`arr_time`** *and* **`sched_arr_time`** *for these flights so that they are recorded as these times for one day after the* **`dep_time`***?*

```
flights %>%
  filter(arr_time < dep_time) %>%
  select(dep_time, arr_time, sched_arr_time)
```

```
# A tibble: 10,633 × 3
   dep_time arr_time sched_arr_time
      <int>    <int>          <int>
1      1929        3              7
2      1939       29           2151
3      2058        8           2359
4      2102      146            158
5      2108       25             39
6      2120       16             18
7      2121        6           2323
8      2128       26             50
9      2134       20           2352
10     2136       25             39
# ... with 10,623 more rows
```

SOLUTION

```
flights %>%
  filter(arr_time < dep_time) %>%
  select(dep_time, arr_time, sched_arr_time) %>%
  mutate(
    overnight = arr_time < dep_time,
    arr_time = arr_time + days(overnight * 1),
    sched_arr_time = sched_arr_time + days(overnight * 1)
  )
# A tibble: 10,633 × 4
   dep_time      arr_time sched_arr_time overnight
      <int>  <S4: Period>   <S4: Period>     <lgl>
1      1929   1d 0H 0M 3S    1d 0H 0M 7S      TRUE
2      1939  1d 0H 0M 29S 1d 0H 0M 2151S      TRUE
3      2058   1d 0H 0M 8S 1d 0H 0M 2359S      TRUE
4      2102 1d 0H 0M 146S  1d 0H 0M 158S      TRUE
5      2108  1d 0H 0M 25S   1d 0H 0M 39S      TRUE
```

WHAT TO REMEMBER

FUNCTIONS TO REMEMBER

Operator/Function	Description
`ymd, ymd_hms, dym, etc.`	parsing functions to turn character into a date-time object
`make_date, make_datetime`	parsing functions to turn separate variables into a date-time object
`year, month, mday, wday, hour, minute, etc.`	functions to extract date-time components
`as.duration, dyears, dmonths, ddays, etc.`	functions to work with durations
`years, months, days, hours, minutes, seconds, etc.`	functions to work with time periods