L11 Dates and Times

Data Science I (STAT 301-1)

Shay Lebovitz

Contents

Overview

The goal of this lab is to learn and understand how to work with dates and times in R (which is surprisingly complex). The `lubridate` package makes it easier to work with dates and times within R in an intuitive framework with methods that can handle time zones, leap days, daylight savings times, and other issues related to time. Note that `lubridate` is part of the tidyverse, but it is not considered "core" tidyverse (and must be installed separately).

These resources may help:

- `lubridate` tidyverse homepage
- `lubridate` vignette (highly recommend)

Datasets

We will be utilizing the familiar `flights` dataset from `nycflights13` (see `?flights` for documentation). **You will need to recreate and work with the `flights_dt` dataset found in the chapter.**

Exercises

Please complete the following exercises. Be sure that your solutions are clearly indicated and that the document is neatly formatted.

```
library(lubridate)
library(nycflights13)
library(tidyverse)
```

Load Packages

Exercise 1 (Website: 16.2.4 Ex. 1)

What happens if you try to parse a string that contains invalid dates, like this one?

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

You get a warning message that it failed to parse the string.

Exercise 2 (Website: 16.2.4 Ex. 3)

Use the appropriate `lubridate` function to parse each of the following dates:

```
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

mdy(d1)
```

```
## [1] "2010-01-01"
```

```
ymd(d2)
```

```
## [1] "2015-03-07"
```

```
dmy(d3)
```

```
## [1] "2017-06-06"
```

```
mdy(d4)
```

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

```
mdy(d5)
```

```
## [1] "2014-12-30"
```

Exercise 3 (Website: 16.3.4 Ex. 2)

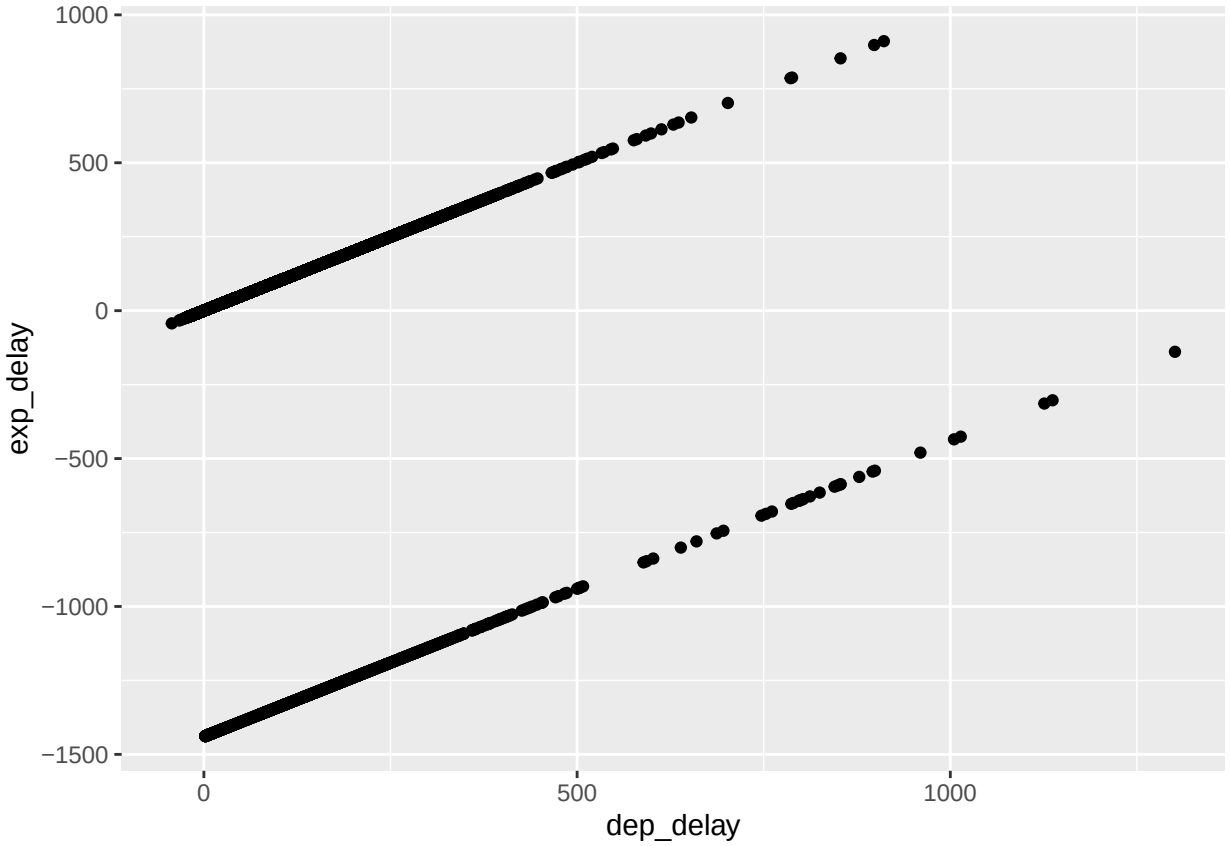
Compare `dep_time`, `sched_dep_time` and `dep_delay`. Are they consistent? Explain your findings.

```
make_datetime_100 <- function(year, month, day, time) {
  make_datetime(year, month, day, time %/% 100, time %% 100)
}

flights_dt <- flights %>%
  filter(!is.na(dep_time), !is.na(arr_time)) %>%
  mutate(
    dep_time = make_datetime_100(year, month, day, dep_time),
    arr_time = make_datetime_100(year, month, day, arr_time),
    sched_dep_time = make_datetime_100(year, month, day, sched_dep_time),
    sched_arr_time = make_datetime_100(year, month, day, sched_arr_time)
  ) %>%
  select(origin, dest, ends_with("delay"), ends_with("time"))

flights_dt %>%
```

```
  select(dep_time, sched_dep_time, dep_delay) %>%
  mutate(exp_delay = as.numeric((dep_time - sched_dep_time)/60)) %>%
  ggplot(aes(x = dep_delay, y = exp_delay)) +
  geom_point()
```

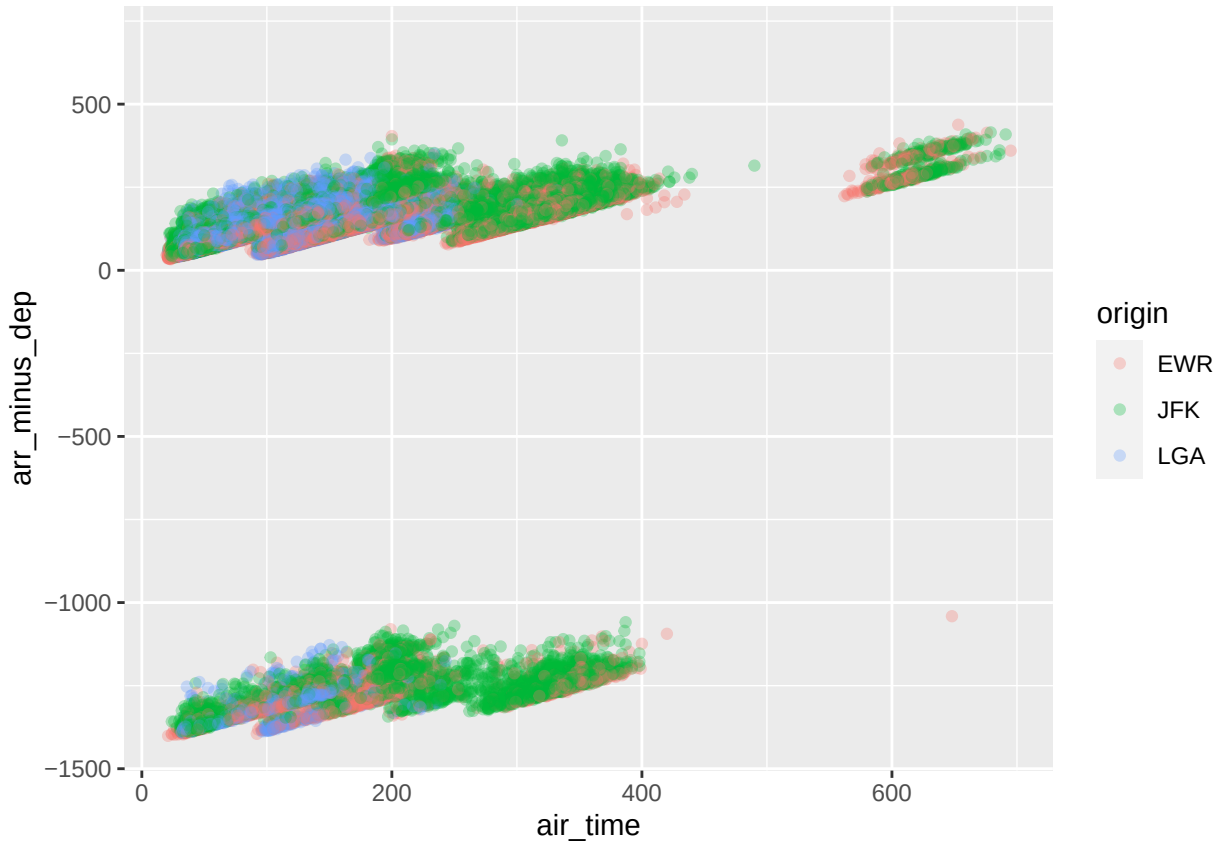


Yes, the departure time, scheduled departure time, and departure delay match up as you'd expect. The issue comes from when a flight gets delayed to a new day.

Exercise 4 (Website: 16.3.4 Ex. 3)

Compare `air_time` with the difference between departure time and arrival time. Explain your findings. (Hint: Consider the location of the airport – this may require joining data frames.)

```
flights_dt %>%
  mutate(arr_minus_dep = as.numeric(arr_time - dep_time)) %>%
  ggplot(aes(x = air_time, y = arr_minus_dep)) +
  geom_point(aes(color = origin), alpha = 0.3)
```

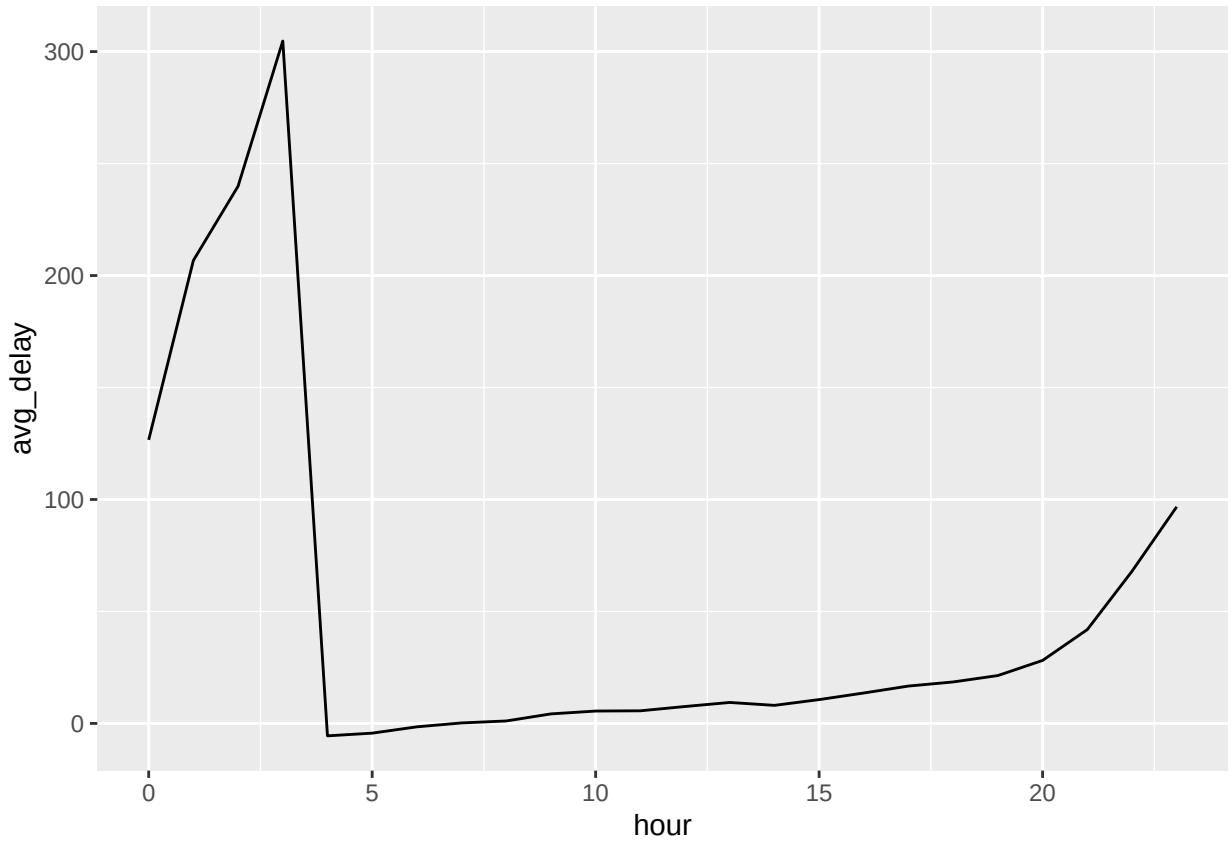


There isn't really a clear relationship between the arrival minus departure time and the air time. Again, we see negative values for when the flight flew overnight. But there is no definitive linear relationship like we saw in question 3. This is likely because of different taxiing times at different airports. It looks like `LGA` tends to have shorter flights (more domestic) whereas `EWR` and `JFK` tend to have longer flights (international).

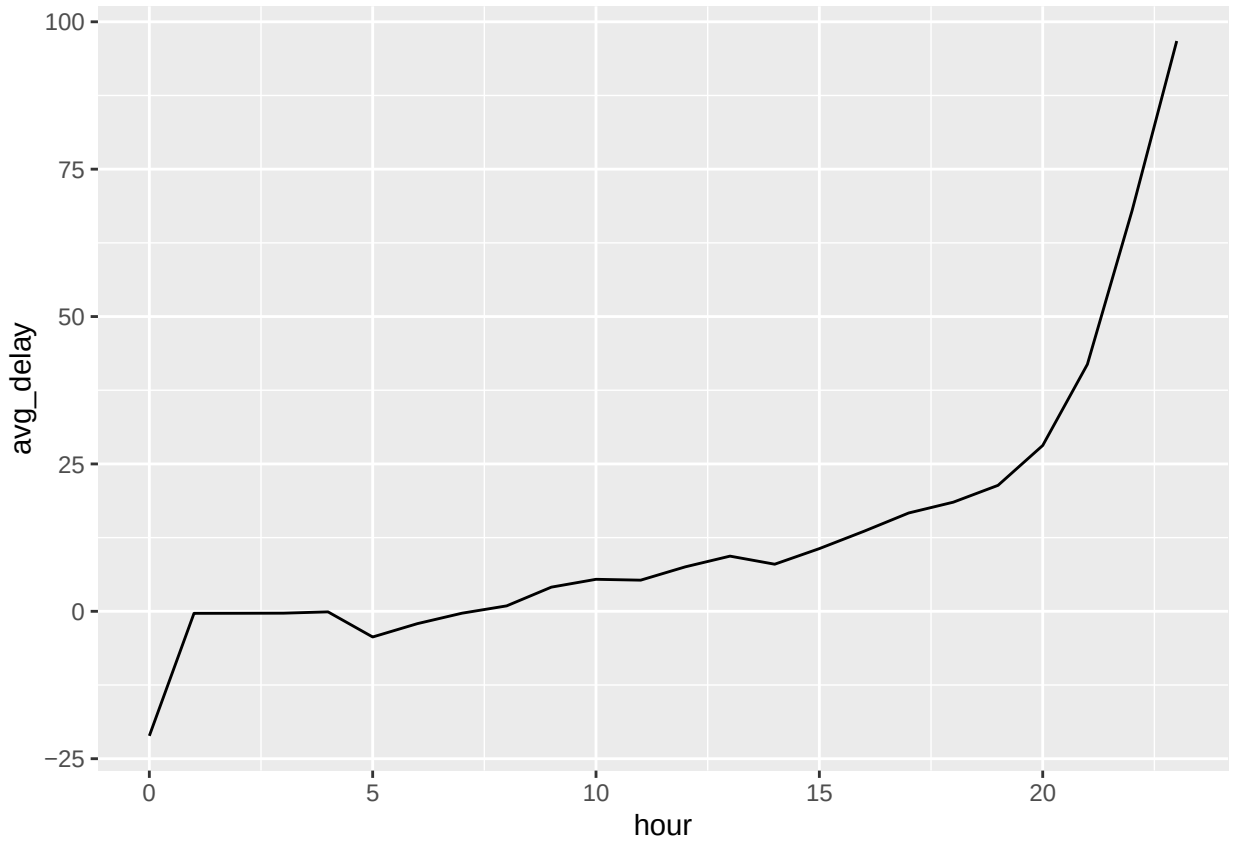
Exercise 5 (Website: 16.3.4 Ex. 4)

How does the average delay time change over the course of a day? Should you use `dep_time` or `sched_dep_time` to assess this? Explain your choice.

```
flights_dt %>%
  mutate(hour = hour(dep_time)) %>%
  group_by(hour) %>%
  summarise(avg_delay = mean(dep_delay, na.rm = TRUE)) %>%
  ggplot(aes(x = hour, y = avg_delay)) +
  geom_line()
```




```
flights_dt %>%
  mutate(hour = hour(dep_time)) %>%
  group_by(hour) %>%
  summarise(
    avg_delay = mean(as.numeric((dep_time - sched_dep_time)/60), na.rm = TRUE)
    ) %>%
  ggplot(aes(x = hour, y = avg_delay)) +
  geom_line()
```




```
flights_dt %>%
  arrange(dep_delay)
```

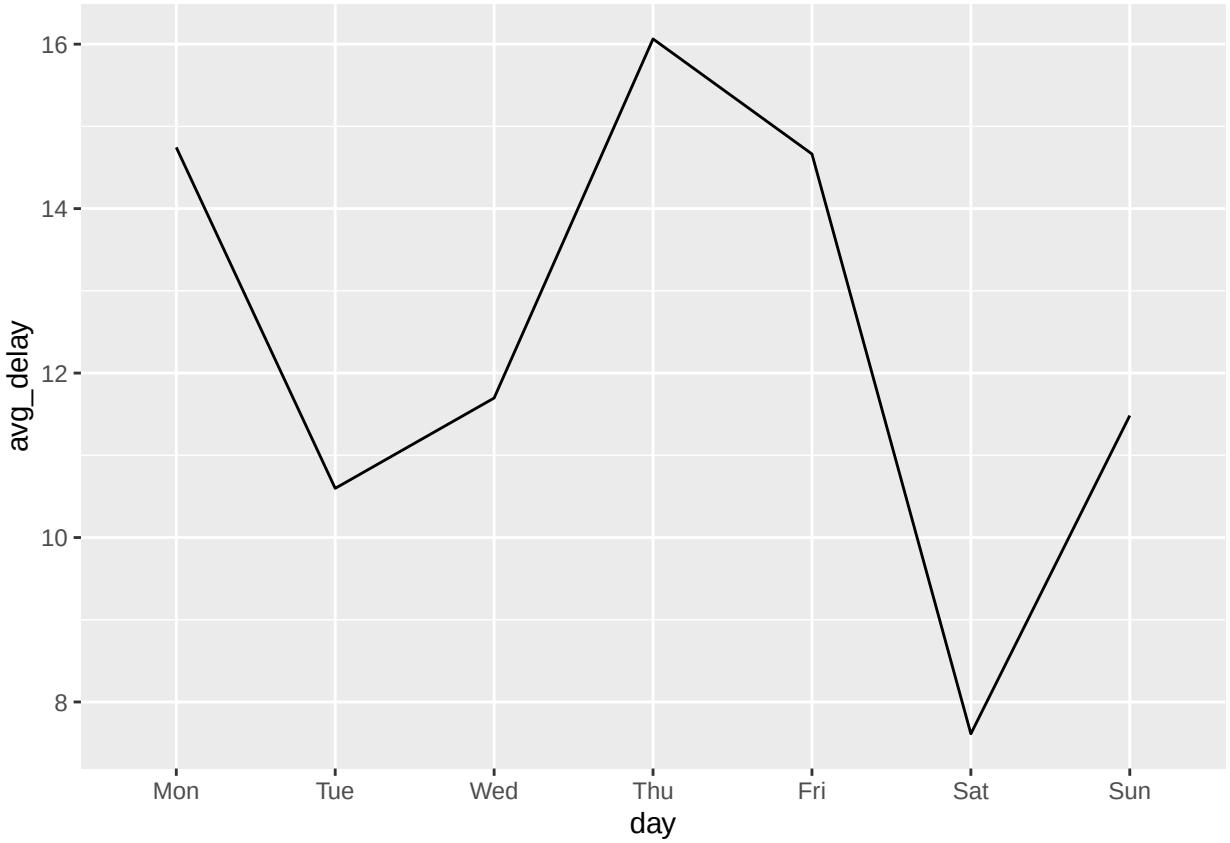
```
## # A tibble: 328,063 x 9
##    origin dest  dep_delay arr_delay dep_time            sched_dep_time
##    <chr>  <chr>     <dbl>     <dbl> <dttm>              <dttm>
##  1 JFK    DEN         -43        48 2013-12-07 20:40:00 2013-12-07 21:23:00
##  2 LGA    MSY         -33       -58 2013-02-03 20:22:00 2013-02-03 20:55:00
##  3 LGA    IAD         -32       -10 2013-11-10 14:08:00 2013-11-10 14:40:00
##  4 LGA    TPA         -30       -10 2013-01-11 19:00:00 2013-01-11 19:30:00
##  5 LGA    DEN         -27       -10 2013-01-29 17:03:00 2013-01-29 17:30:00
##  6 LGA    DTW         -26         7 2013-08-09 07:29:00 2013-08-09 07:55:00
##  7 EWR    TYS         -25         0 2013-10-23 19:07:00 2013-10-23 19:32:00
##  8 LGA    DTW         -25       -37 2013-03-30 20:30:00 2013-03-30 20:55:00
##  9 JFK    BUF         -24       -30 2013-03-02 14:31:00 2013-03-02 14:55:00
## 10 LGA    FLL         -24       -44 2013-05-05 09:34:00 2013-05-05 09:58:00
## # ... with 328,053 more rows, and 3 more variables: arr_time <dttm>,
## #   sched_arr_time <dttm>, air_time <dbl>
```

Because of the reasons mentioned in question 3 (negative delay times when delayed across midnight), using `dep_delay` is better than using the difference of `dep_time` and `sched_dep_time`. We see that night flights (10pm - 4am) have significantly higher average delays than anything during the day.

Exercise 6 (Website: 16.3.4 Ex. 5)

On what day of the week should you leave if you want to minimize the chance of a delay?

```
flights_dt %>%
  mutate(day = wday(dep_time, label = T, week_start = 1)) %>%
  group_by(day) %>%
  summarize(avg_delay = mean(dep_delay, na.rm = TRUE)) %>%
  ggplot(aes(x = day, y = avg_delay)) +
  geom_line(aes(group = 1))
```



Saturday has the lowest average delay of all days of the week.

Exercise 7 (Website: 16.4.5 Ex. 3)

Create a tibble containing vectors of (1) every month in the current year (i.e., "January"), (2) the date of the first day of every month (i.e., "2019-01-01"), and (3) the day of the week that each first day falls on (i.e., "Tuesday").

```
first_day <- ymd('2020-01-01') + months(0:11)
months <- month(first_day, label = TRUE, abbr = FALSE)
day_of_week <- weekdays(first_day)
(myTibble <- tibble(months, first_day, day_of_week))
```

```
## # A tibble: 12 x 3
##    months    first_day  day_of_week
##    <ord>     <date>     <chr>
##  1 January   2020-01-01 Wednesday
##  2 February  2020-02-01 Saturday
##  3 March     2020-03-01 Sunday
##  4 April     2020-04-01 Wednesday
```

```
##  5 May       2020-05-01 Friday
##  6 June      2020-06-01 Monday
##  7 July      2020-07-01 Wednesday
##  8 August    2020-08-01 Saturday
##  9 September 2020-09-01 Tuesday
## 10 October   2020-10-01 Thursday
## 11 November  2020-11-01 Sunday
## 12 December  2020-12-01 Tuesday
```

Exercise 8 (Website: 16.4.5 Ex. 4)

Write a function that returns how old you are in years, given your birthday (as `"YYYY-MM-DD"`).

```
my_bd <- function(birthday) {
  birthday <- ymd(birthday)
  as.numeric(as.duration(today() - birthday), "years") %/% 1
}
my_bd('1999-03-04')
```

```
## [1] 21
```

```
my_bd('2001-12-19')
```

```
## [1] 19
```