Assignment 5

Adrian Bracher (Matr. Nr. 01637180)

28.04.2021

Contents

1 Dates and Times in R

1.1 Specifying dates

In this exercise we learn how to interpret a string as a Date.

```
# The date R 3.0.0 was released
x <- "2013-04-03"

# Examine structure of x
str(x)
```

```
##  chr "2013-04-03"
```

```
# Use as.Date() to interpret x as a date
x_date <- as.Date(x)

# Examine structure of x_date
str(x_date)
```

```
##  Date[1:1], format: "2013-04-03"
```

```
# Store April 10 2014 as a Date
april_10_2014 <- as.Date("2014-04-10")
```

1.2 Automatic import

We use the library “anytime” to automatically parse different date formats.

```
# Load the readr package
library(readr)

# Use read_csv() to import rversions.csv
releases <- read_csv("rversions.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   major = col_double(),
##   minor = col_double(),
##   patch = col_double(),
##   date = col_date(format = ""),
##   datetime = col_datetime(format = ""),
##   time = col_time(format = ""),
##   type = col_character()
## )
```

```
# Examine the structure of the date column
str(releases$date)
```

```
##  Date[1:105], format: "1997-12-04" "1997-12-21" "1998-01-10" "1998-03-14" "1998-05-02" ...
```

```
# Load the anytime package
library(anytime)

# Various ways of writing Sep 10 2009
sep_10_2009 <- c("September 10 2009", "2009-09-10", "10 Sep 2009", "09-10-2009")
```

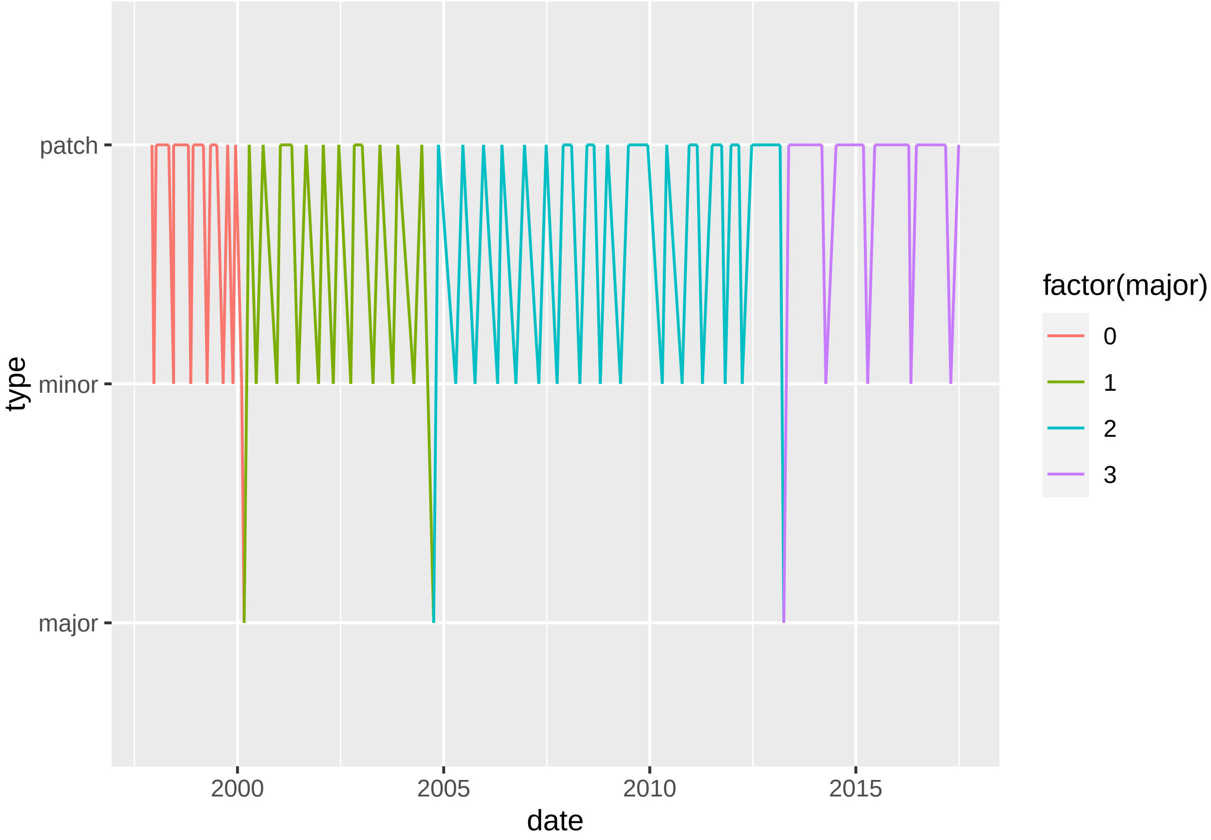
```
# Use anytime() to parse sep_10_2009
anytime(sep_10_2009)
```

```
## [1] "2009-09-10 CEST" "2009-09-10 CEST" "2009-09-10 CEST" "2009-09-10 CEST"
```

1.3 Plotting

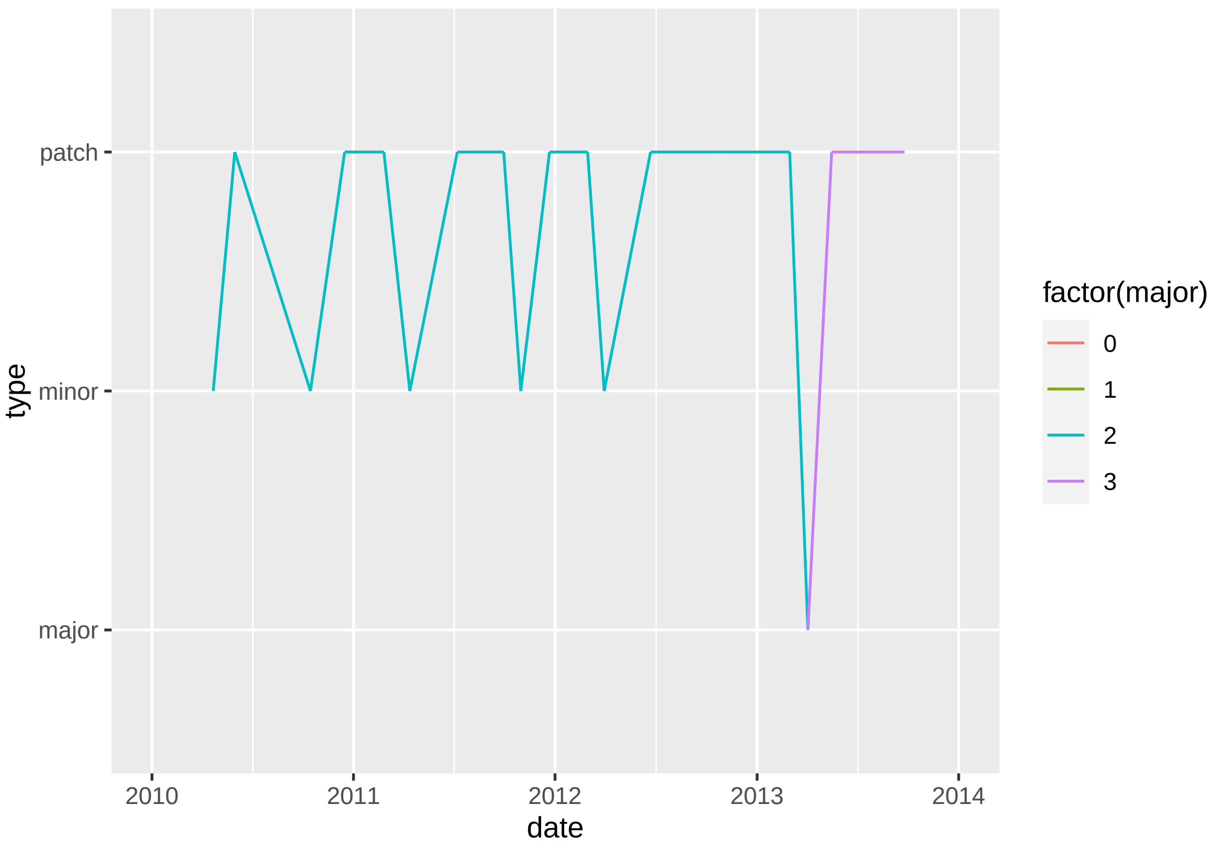
In this exercise we plot data of type Date. We also learn how to limit, set breaks and label formats.

```
library(ggplot2)

# Set the x axis to the date column
ggplot(releases, aes(x = date, y = type)) +
  geom_line(aes(group = 1, color = factor(major)))
```

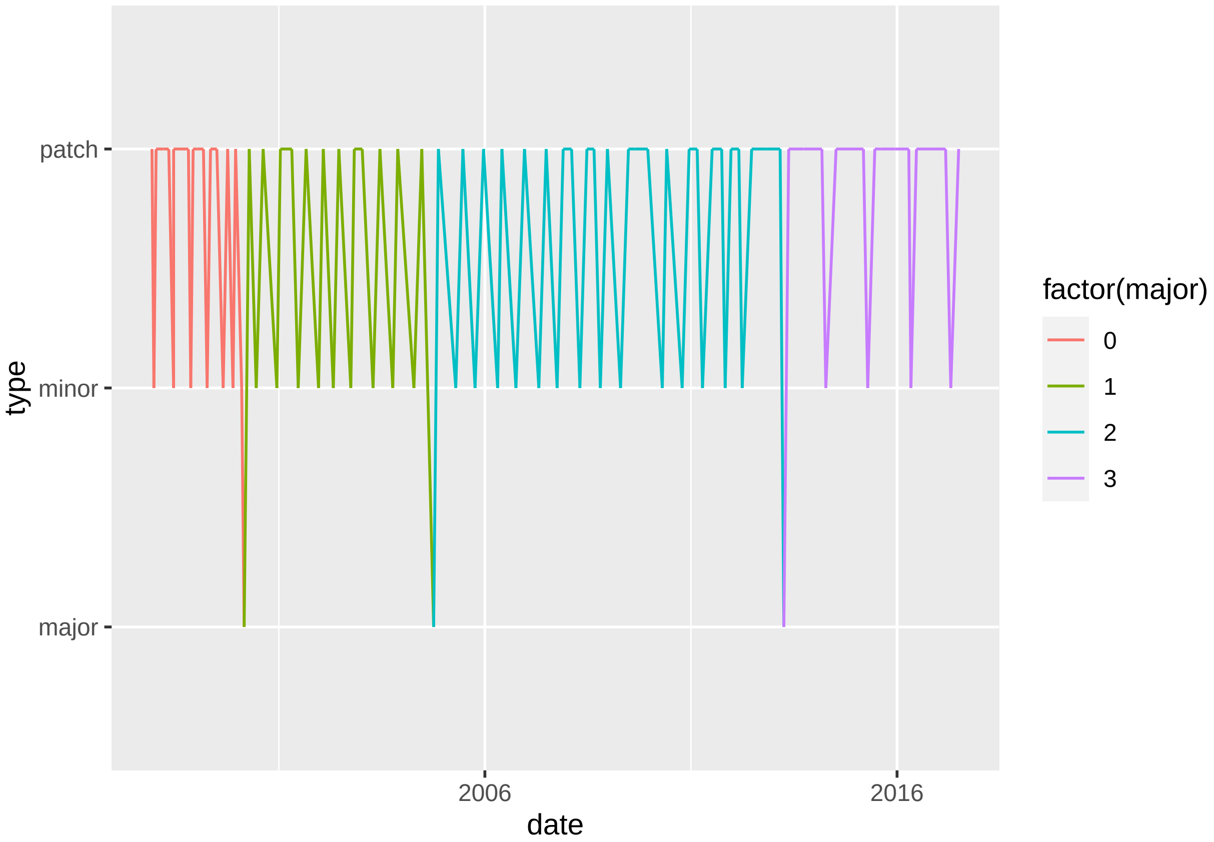


```
# Limit the axis to between 2010-01-01 and 2014-01-01
ggplot(releases, aes(x = date, y = type)) +
  geom_line(aes(group = 1, color = factor(major))) +
  xlim(as.Date("2010-01-01"), as.Date("2014-01-01"))
```

```
## Warning: Removed 87 row(s) containing missing values (geom_path).
```



```
# Specify breaks every ten years and labels with "%Y"
ggplot(releases, aes(x = date, y = type)) +
  geom_line(aes(group = 1, color = factor(major))) +
  scale_x_date(date_breaks = "10 years", date_labels = "%Y")
```



## Arithmetic and logical operators Here we do simple arithmetics on dates.

```
# Find the largest date
last_release_date <- max(releases$date)

# Filter row for last release
last_release <- filter(releases, date == last_release_date)

# Print last_release
last_release

# How long since last release?
Sys.Date() - last_release_date
```

1.4 Getting datetimes into R

In the code below we learn how to handle dates with times, i.e., datetimes.

```
# Use as.POSIXct to enter the datetime
as.POSIXct("2010-10-01 12:12:00")
```

```
## [1] "2010-10-01 12:12:00 CEST"
```

```
# Use as.POSIXct again but set the timezone to `"America/Los_Angeles"`
as.POSIXct("2010-10-01 12:12:00", tz = "America/Los_Angeles")
```

```
## [1] "2010-10-01 12:12:00 PDT"
```

```r
# Use read_csv to import rversions.csv
releases <- read_csv("rversions.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   major = col_double(),
##   minor = col_double(),
##   patch = col_double(),
##   date = col_date(format = ""),
##   datetime = col_datetime(format = ""),
##   time = col_time(format = ""),
##   type = col_character()
## )
```

```r
# Examine structure of datetime column
str(releases$datetime)
```

```
##  POSIXct[1:105], format: "1997-12-04 08:47:58" "1997-12-21 13:09:22" "1998-01-10 00:31:55" ...
```

1.5 Datetimes behave nicely too

In this exercise we learn how to plot and filter datetime.lyr“‘{r} # Import “cran-logs_2015-04-17.csv” with read_csv() logs <- read_csv(“cran-logs_2015-04-17.csv”)

2 Print logs

print(logs)

3 Store the release time as a POSIXct object

release_time <- as.POSIXct(“2015-04-16 07:13:33”, tz = “UTC”)

4 When is the first download of 3.2.0?

logs %>% filter(release_time < datetime, r_version == “3.2.0”)

5 Examine histograms of downloads by version

ggplot(logs, aes(x = datetime)) + geom_histogram() + geom_vline(aes(xintercept = as.numeric(release_time)))+ facet_wrap(~ r_version, ncol = 1)

~~~
## Selecting the right parsing function
The library lubridate offers different functions to parse different date formats. For example dmy parses

```r
library(lubridate)
~~~

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
# Parse x
x <- "2010 September 20th" # 2010-09-20
ymd(x)
```

```
## [1] "2010-09-20"
```

```
# Parse y
y <- "02.01.2010"  # 2010-01-02
dmy(y)
```

```
## [1] "2010-01-02"
```

```
# Parse z
z <- "Sep, 12th 2010 14:00"  # 2010-09-12T14:00
mdy_hm(z)
```

```
## [1] "2010-09-12 14:00:00 UTC"
```

5.1 Specifying an order with 'parse_date_time()'

For even weirder date formats we can use the function parse_date_time with the option orders. Non-specified values are set to 01.

```
# Specify an order string to parse x
x <- "Monday June 1st 2010 at 4pm"
parse_date_time(x, orders = "Amdy_Ip")
```

```
## Warning: All formats failed to parse. No formats found.
```

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

```
# Specify order to include both "mdy" and "dmy"
two_orders <- c("October 7, 2001", "October 13, 2002", "April 13, 2003",
  "17 April 2005", "23 April 2017")
parse_date_time(two_orders, orders = c("mdy", "dmy"))
```

```
## [1] "2001-10-07 UTC" "2002-10-13 UTC" "2003-04-13 UTC" "2005-04-17 UTC"
## [5] "2017-04-23 UTC"
```

```
# Specify order to include "dOmY", "OmY" and "Y"
short_dates <- c("11 December 1282", "May 1372", "1253")
parse_date_time(short_dates, orders = c("dOmY", "OmY", "Y"))
```

```
## [1] "1282-12-11 UTC" "1372-05-01 UTC" "1253-01-01 UTC"
```

5.2 Import daily weather data

We use functions from dplyr like mutate to make sure dates are interpreted as date type.

```
library(lubridate)
library(readr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)

# Import CSV with read_csv()
akl_daily_raw <- read_csv("akl_weather_daily.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   date = col_character(),
##   max_temp = col_double(),
##   min_temp = col_double(),
##   mean_temp = col_double(),
##   mean_rh = col_double(),
##   events = col_character(),
##   cloud_cover = col_double()
## )
```

```
# Print akl_daily_raw
akl_daily_raw
```

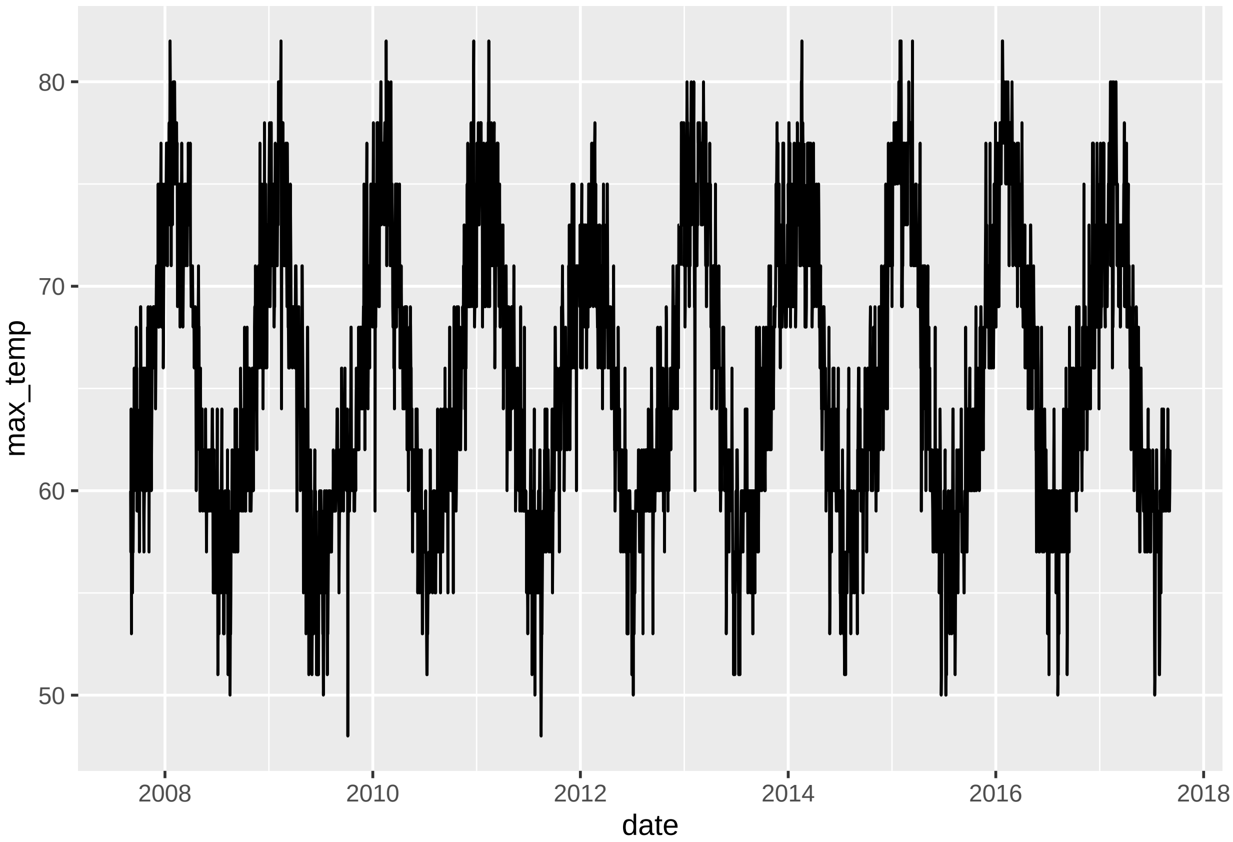
```
## # A tibble: 3,661 x 7
##    date      max_temp min_temp mean_temp mean_rh events cloud_cover
##    <chr>        <dbl>    <dbl>     <dbl>   <dbl> <chr>        <dbl>
##  1 2007-9-1        60       51        56      75 <NA>             4
##  2 2007-9-2        60       53        56      82 Rain             4
##  3 2007-9-3        57       51        54      78 <NA>             6
##  4 2007-9-4        64       50        57      80 Rain             6
##  5 2007-9-5        53       48        50      90 Rain             7
##  6 2007-9-6        57       42        50      69 <NA>             1
##  7 2007-9-7        59       41        50      77 <NA>             4
##  8 2007-9-8        59       46        52      80 <NA>             5
##  9 2007-9-9        55       50        52      88 Rain             7
## 10 2007-9-10       59       50        54      82 Rain             4
## # ... with 3,651 more rows
```

```
# Parse date
akl_daily <- akl_daily_raw %>%
  mutate(date = as.Date(date))

# Print akl_daily
akl_daily
```

```
## # A tibble: 3,661 x 7
##    date       max_temp min_temp mean_temp mean_rh events cloud_cover
##    <date>        <dbl>    <dbl>     <dbl>   <dbl> <chr>        <dbl>
##  1 2007-09-01       60       51        56      75 <NA>             4
##  2 2007-09-02       60       53        56      82 Rain             4
##  3 2007-09-03       57       51        54      78 <NA>             6
##  4 2007-09-04       64       50        57      80 Rain             6
##  5 2007-09-05       53       48        50      90 Rain             7
##  6 2007-09-06       57       42        50      69 <NA>             1
##  7 2007-09-07       59       41        50      77 <NA>             4
##  8 2007-09-08       59       46        52      80 <NA>             5
##  9 2007-09-09       55       50        52      88 Rain             7
## 10 2007-09-10       59       50        54      82 Rain             4
## # ... with 3,651 more rows
```

```
# Plot to check work
ggplot(akl_daily, aes(x = date, y = max_temp)) +
  geom_line()
```

```
## Warning: Removed 1 row(s) containing missing values (geom_path).
```



## Import hourly weather data In this exercise we import a csv with columns containing year, month and day information and we use make_date() to combine them to a Date type.

```
library(lubridate)
library(readr)
library(dplyr)
library(ggplot2)

# Import "akl_weather_hourly_2016.csv"
akl_hourly_raw <-  read_csv("akl_weather_hourly_2016.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   year = col_double(),
##   month = col_double(),
##   mday = col_double(),
##   time = col_time(format = ""),
##   temperature = col_double(),
##   weather = col_character(),
##   conditions = col_character(),
##   events = col_character(),
##   humidity = col_double(),
##   date_utc = col_datetime(format = "")
## )
```

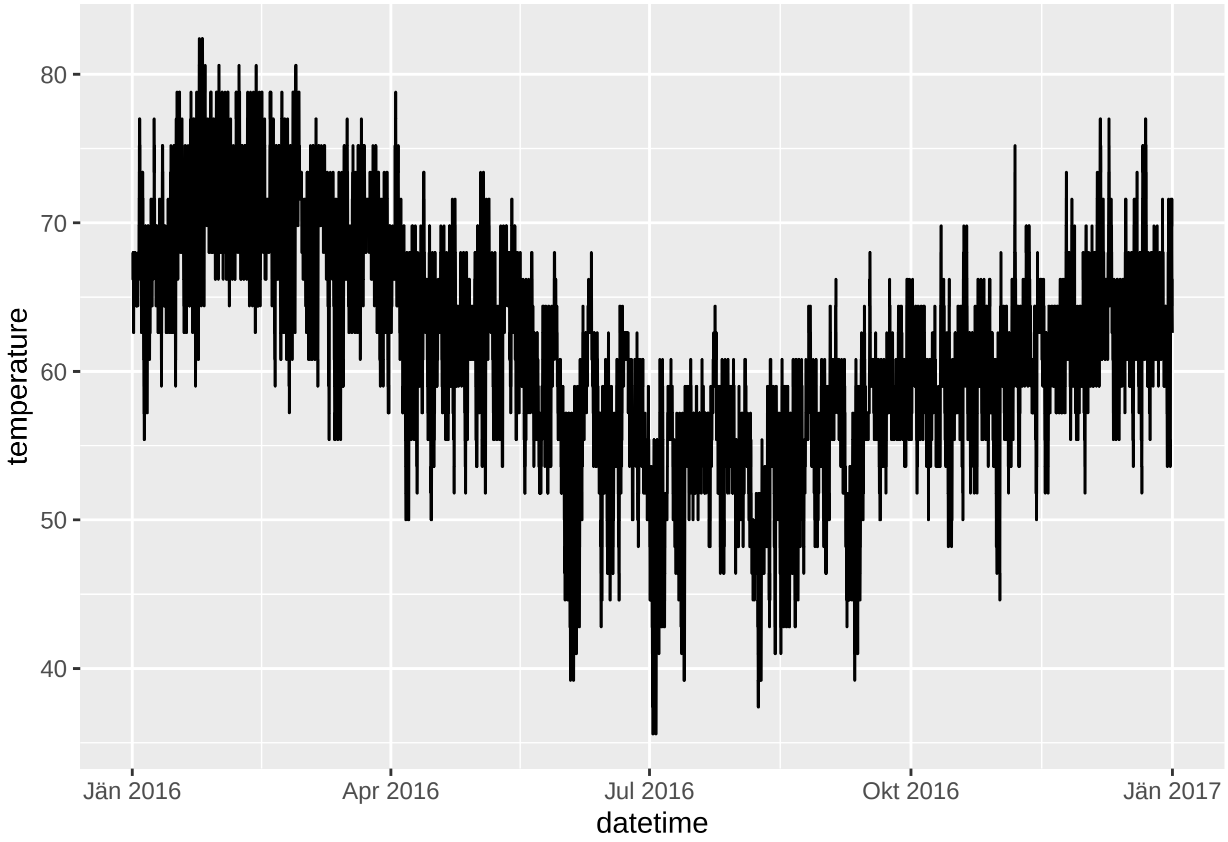
```r
# Print akl_hourly_raw
akl_hourly_raw
```

```
## # A tibble: 17,454 x 10
##     year month  mday time   temperature weather conditions       events humidity
##    <dbl> <dbl> <dbl> <time>       <dbl> <chr>   <chr>            <chr>     <dbl>
##  1  2016     1     1 00:00           68   Clear   Clear            <NA>         68
##  2  2016     1     1 00:30           68   Clear   Clear            <NA>         68
##  3  2016     1     1 01:00           68   Clear   Clear            <NA>         73
##  4  2016     1     1 01:30           68   Clear   Clear            <NA>         68
##  5  2016     1     1 02:00           68   Clear   Clear            <NA>         68
##  6  2016     1     1 02:30           68   Clear   Clear            <NA>         68
##  7  2016     1     1 03:00           68   Clear   Clear            <NA>         68
##  8  2016     1     1 03:30           68   Cloudy  Partly Cloudy    <NA>         68
##  9  2016     1     1 04:00           68   Cloudy  Scattered Clouds <NA>         68
## 10  2016     1     1 04:30           66.2 Cloudy  Partly Cloudy    <NA>         73
## # ... with 17,444 more rows, and 1 more variable: date_utc <dttm>
```

```r
# Use make_date() to combine year, month and mday
akl_hourly  <- akl_hourly_raw  %>%
  mutate(date = make_date(year = year, month = month, day = mday))

# Parse datetime_string
akl_hourly <- akl_hourly  %>%
  mutate(
    datetime_string = paste(date, time, sep = "T"),
    datetime = ymd_hms(datetime_string)
  )

# Print date, time and datetime columns of akl_hourly
akl_hourly %>% select(date, time, datetime)
```

```
## # A tibble: 17,454 x 3
##    date       time   datetime
##    <date>     <time> <dttm>
##  1 2016-01-01 00:00  2016-01-01 00:00:00
##  2 2016-01-01 00:30  2016-01-01 00:30:00
##  3 2016-01-01 01:00  2016-01-01 01:00:00
##  4 2016-01-01 01:30  2016-01-01 01:30:00
##  5 2016-01-01 02:00  2016-01-01 02:00:00
##  6 2016-01-01 02:30  2016-01-01 02:30:00
##  7 2016-01-01 03:00  2016-01-01 03:00:00
##  8 2016-01-01 03:30  2016-01-01 03:30:00
##  9 2016-01-01 04:00  2016-01-01 04:00:00
## 10 2016-01-01 04:30  2016-01-01 04:30:00
## # ... with 17,444 more rows
```

```
# Plot to check work
ggplot(akl_hourly, aes(x = datetime, y = temperature)) +
  geom_line()
```



## What can you extract? The lubridate functions year(), month(), day(), etc. can be used to extract information from a datetime value.

```
# Examine the head() of release_time
head(month(release_time))

# Examine the head() of the months of release_time
head(ymd_hms(release_time))

# Extract the month of releases
month(release_time) %>% table()

# Extract the year of releases
year(release_time) %>% table()

# How often is the hour before 12 (noon)?
mean(hour(release_time) < 12)

# How often is the release in am?
mean(am(release_time))
```

5.3 Adding useful labels

The function wday can be used to extract the weekday from a date.

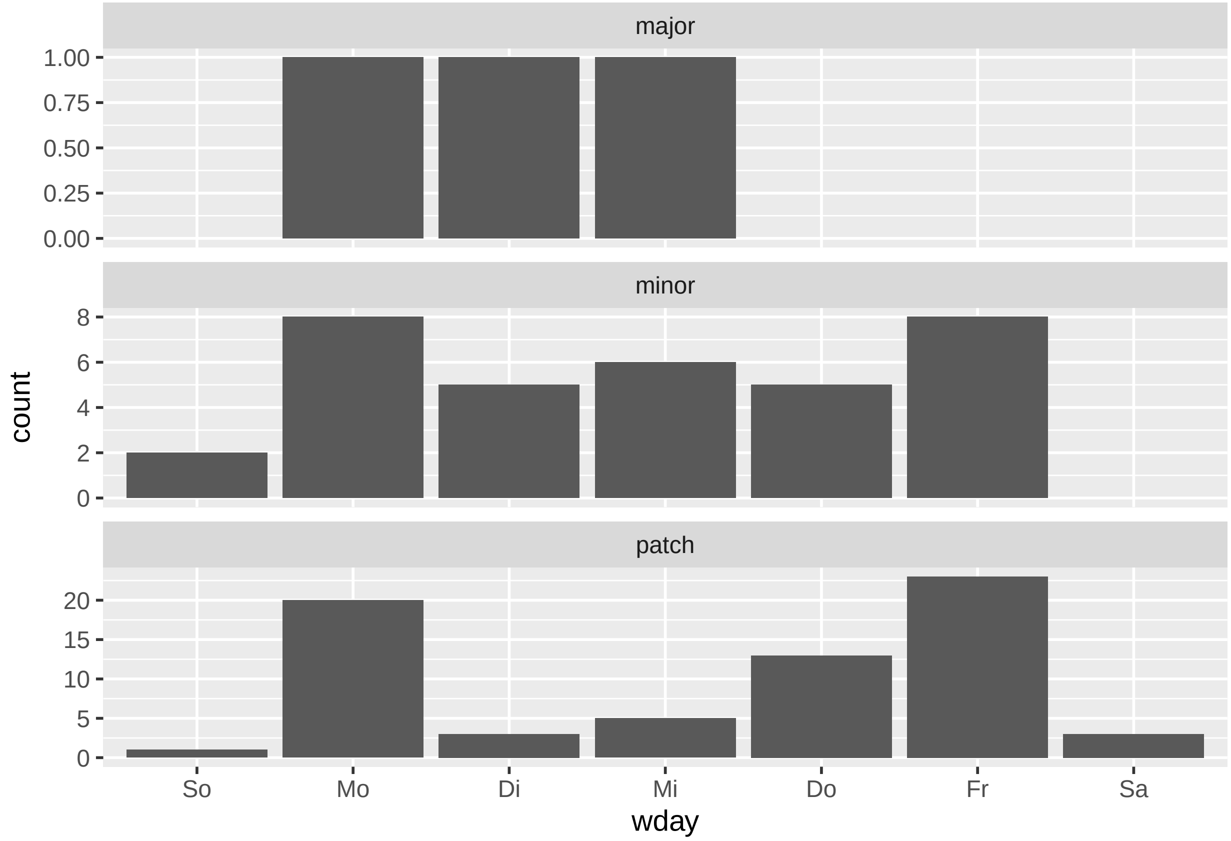
```
library(ggplot2)

# Use wday() to tabulate release by day of the week
wday(releases$datetime) %>% table()
```

```
## .
##  1  2  3  4  5  6  7
##  3 29  9 12 18 31  3
```

```
# Add label = TRUE to make table more readable
wday(releases$datetime, label=TRUE) %>% table()
```

```
## .
## So Mo Di Mi Do Fr Sa
##  3 29  9 12 18 31  3
```

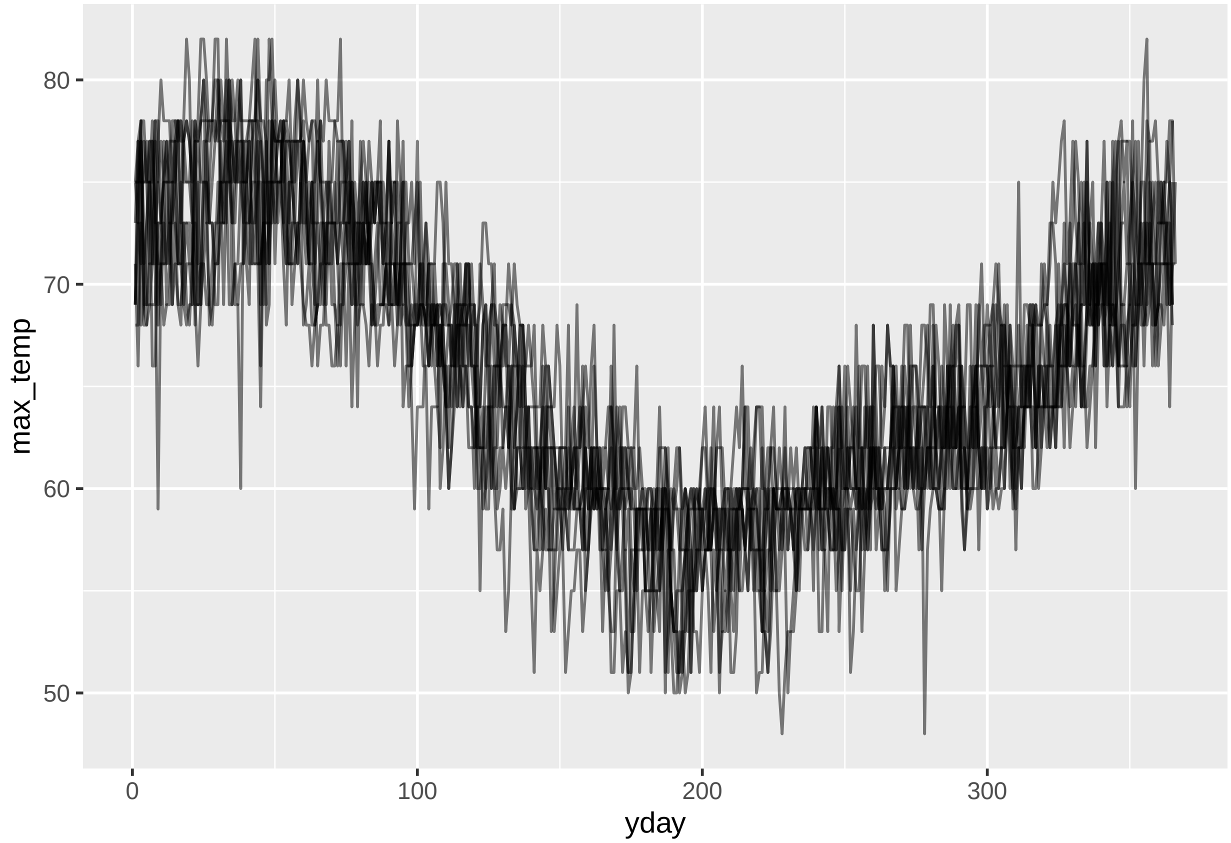
```
# Create column wday to hold labelled week days
releases$wday <- wday(releases$datetime, label=TRUE)

# Plot barchart of weekday by type of release
ggplot(releases, aes(wday)) +
  geom_bar() +
  facet_wrap(~ type, ncol = 1, scale = "free_y")
```



## Extracting for plotting In the code below we plot extracted date data

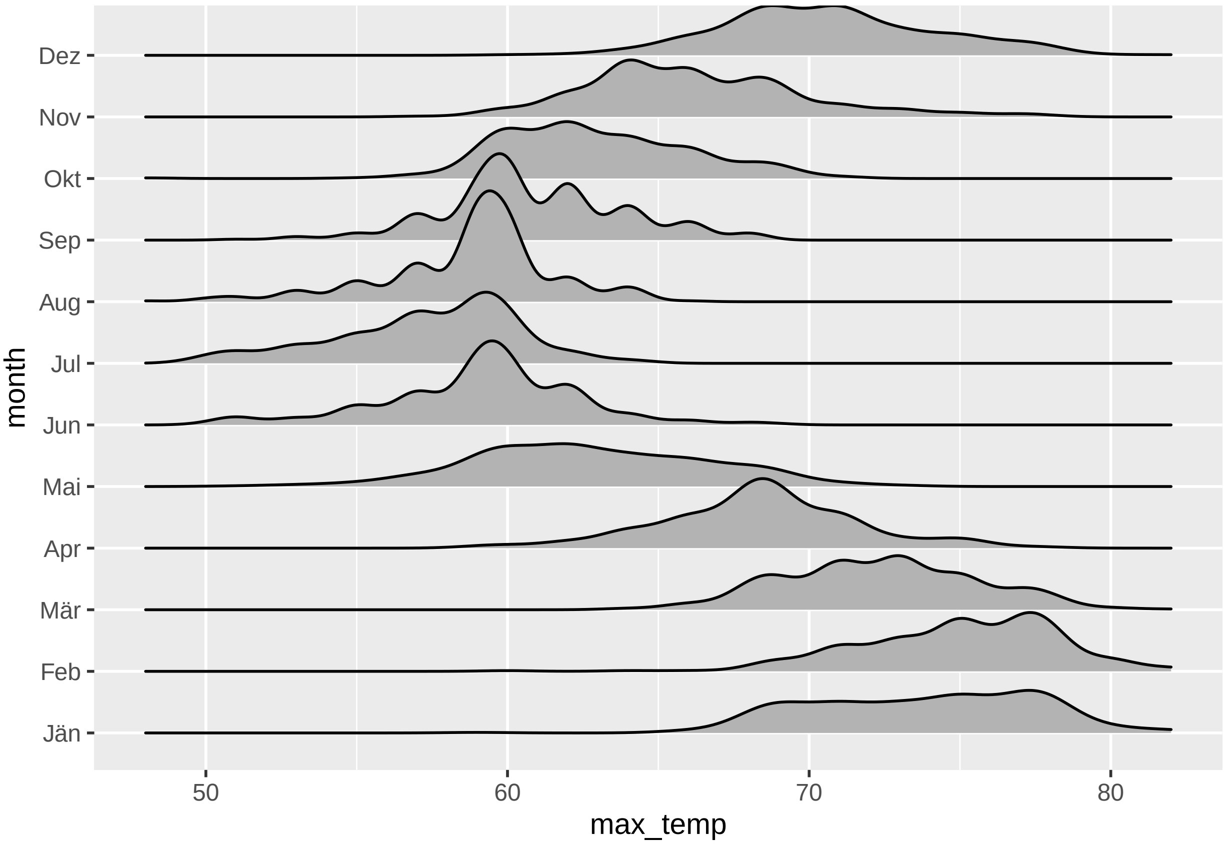
```r
library(ggplot2)
library(dplyr)
library(ggridges)

# Add columns for year, yday and month
akl_daily <- akl_daily %>%
  mutate(
    year = year(date),
    yday = yday(date),
    month = month(date, label=TRUE))

# Plot max_temp by yday for all years
ggplot(akl_daily, aes(x = yday, y = max_temp)) +
  geom_line(aes(group = year), alpha = 0.5)
```

```
## Warning: Removed 1 row(s) containing missing values (geom_path).
```



```r
# Examine distribution of max_temp by month
ggplot(akl_daily, aes(x = max_temp, y = month, height = ..density..)) +
  geom_density_ridges(stat = "density")
```

```
## Warning: Removed 10 rows containing non-finite values (stat_density).
```



## Extracting for filtering and summarising We can also use extraction in combination with dplyr functions such as filter, summarise, group_by and mutate.

```r
# Create new columns hour, month and rainy
akl_hourly <- akl_hourly %>%
  mutate(
    hour = hour(datetime),
    month = month(datetime, label=TRUE),
    rainy = weather == "Precipitation"
  )

# Filter for hours between 8am and 10pm (inclusive)
akl_day <- akl_hourly %>%
  filter(hour >= 8, hour <= 22)

# Summarise for each date if there is any rain
rainy_days <- akl_day %>%
  group_by(month, date) %>%
  summarise(
    any_rain = any(rainy)
  )
```

```
## `summarise()` has grouped output by 'month'. You can override using the `.groups` argument.
```

```r
# Summarise for each month, the number of days with rain
rainy_days %>%
  summarise(
    days_rainy = sum(any_rain)
  )
```

```
## # A tibble: 12 x 2
##    month days_rainy
##    <ord>      <int>
##  1 Jän           15
##  2 Feb           13
##  3 Mär           12
##  4 Apr           15
##  5 Mai           21
##  6 Jun           19
##  7 Jul           22
##  8 Aug           16
##  9 Sep           25
## 10 Okt           20
## 11 Nov           19
## 12 Dez           11
```

5.4 Practice rounding

In this exercise we learn how to round date to the nearest value with round_date(), round up with ceiling_date() and round down with floor_date().

```
r_3_4_1 <- ymd_hms("2016-05-03 07:13:28 UTC")

# Round down to day
floor_date(r_3_4_1, unit = "day")
```

```
## [1] "2016-05-03 UTC"
```

```
# Round to nearest 5 minutes
round_date(r_3_4_1, unit = "5 minutes")
```

```
## [1] "2016-05-03 07:15:00 UTC"
```

```
# Round up to week
ceiling_date(r_3_4_1, unit = "week")
```

```
## [1] "2016-05-08 UTC"
```

```
# Subtract r_3_4_1 rounded down to day
r_3_4_1 - floor_date(r_3_4_1, unit = "day")
```

```
## Time difference of 7.224444 hours
```

5.5 Rounding with the weather data

We use our previously acquired rounding skills on weather data.

```
# Create day_hour, datetime rounded down to hour
akl_hourly <- akl_hourly %>%
  mutate(
    day_hour = floor_date(datetime, unit = "hour")
  )

# Count observations per hour
akl_hourly %>%
  count(day_hour)
```

```
## # A tibble: 8,770 x 2
##    day_hour                n
##    <dttm>              <int>
##  1 2016-01-01 00:00:00     2
##  2 2016-01-01 01:00:00     2
##  3 2016-01-01 02:00:00     2
##  4 2016-01-01 03:00:00     2
##  5 2016-01-01 04:00:00     2
##  6 2016-01-01 05:00:00     2
##  7 2016-01-01 06:00:00     2
##  8 2016-01-01 07:00:00     2
##  9 2016-01-01 08:00:00     2
## 10 2016-01-01 09:00:00     2
## # ... with 8,760 more rows
```

```
# Find day_hours with n != 2
akl_hourly %>%
  count(day_hour) %>%
  filter(n != 2) %>%
  arrange(desc(n))
```

```
## # A tibble: 92 x 2
##    day_hour                n
##    <dttm>              <int>
##  1 2016-04-03 02:00:00     4
##  2 2016-09-25 00:00:00     4
##  3 2016-06-26 09:00:00     1
##  4 2016-09-01 23:00:00     1
##  5 2016-09-02 01:00:00     1
##  6 2016-09-04 11:00:00     1
##  7 2016-09-04 16:00:00     1
##  8 2016-09-04 17:00:00     1
##  9 2016-09-05 00:00:00     1
## 10 2016-09-05 15:00:00     1
## # ... with 82 more rows
```

6 Arithmetic with Dates and Times

6.1 How long has it been?

In this exercise we calculate how many days and seconds passed since the first man stepped on the moon with the function difftime().

```
# The date of landing and moment of step
date_landing <- mdy("July 20, 1969")
moment_step <- mdy_hms("July 20, 1969, 02:56:15", tz = "UTC")

# How many days since the first man on the moon?
difftime(today(), date_landing, units = "days")
```

```
## Time difference of 18910 days
```

```
# How many seconds since the first man on the moon?
difftime(now(), moment_step, units = "secs")
```

```
## Time difference of 1633881229 secs
```

6.2 How many seconds are in a day?

Similarily to the last exercise we use difftime to calculate datetime differences.

```
# Three dates
mar_11 <- ymd_hms("2017-03-11 12:00:00",
  tz = "America/Los_Angeles")
mar_12 <- ymd_hms("2017-03-12 12:00:00",
  tz = "America/Los_Angeles")
mar_13 <- ymd_hms("2017-03-13 12:00:00",
  tz = "America/Los_Angeles")

# Difference between mar_13 and mar_12 in seconds
difftime(mar_13, mar_12, units = "secs")
```

```
## Time difference of 86400 secs
```

```
# Difference between mar_12 and mar_11 in seconds
difftime(mar_12, mar_11, units = "secs")
```

```
## Time difference of 82800 secs
```

6.3 Adding or subtracting a time span to a datetime

In the code below we add and subtract periods and durations to datetime values.

```
# Add a period of one week to mon_2pm
mon_2pm <- dmy_hm("27 Aug 2018 14:00")
mon_2pm + weeks(1)
```

```
## [1] "2018-09-03 14:00:00 UTC"
```

```
# Add a duration of 81 hours to tue_9am
tue_9am <- dmy_hm("28 Aug 2018 9:00")
tue_9am + dhours(81)
```

```
## [1] "2018-08-31 18:00:00 UTC"
```

```
# Subtract a period of five years from today()
today() - years(5)
```

```
## [1] "2016-04-28"
```

```
# Subtract a duration of five years from today()
today() - dyears(5)
```

```
## [1] "2016-04-27 18:00:00 UTC"
```

6.4 Arithmetic with timespans

Durations can be added and subtracted and even multiplied by numbers.

```
# Time of North American Eclipse 2017
eclipse_2017 <- ymd_hms("2017-08-21 18:26:40")

# Duration of 29 days, 12 hours, 44 mins and 3 secs
synodic <- ddays(29) + dhours(12) + dminutes(44) + dseconds(3)

# 223 synodic months
saros <- 223*synodic
```

```
# Add saros to eclipse_2017
eclipse_2017 + saros
```

```
## [1] "2035-09-02 02:09:49 UTC"
```

6.5 Generating sequences of datetimes

It's also possible to create sequences of datetimes like below with *:* * duration.

```
# Add a period of 8 hours to today
today_8am <- today() + hours(8)

# Sequence of two weeks from 1 to 26
every_two_weeks <- 1:26 * weeks(2)

# Create datetime for every two weeks for a year
every_two_weeks + today_8am
```

```
##  [1] "2021-05-12 08:00:00 UTC" "2021-05-26 08:00:00 UTC"
##  [3] "2021-06-09 08:00:00 UTC" "2021-06-23 08:00:00 UTC"
##  [5] "2021-07-07 08:00:00 UTC" "2021-07-21 08:00:00 UTC"
##  [7] "2021-08-04 08:00:00 UTC" "2021-08-18 08:00:00 UTC"
##  [9] "2021-09-01 08:00:00 UTC" "2021-09-15 08:00:00 UTC"
## [11] "2021-09-29 08:00:00 UTC" "2021-10-13 08:00:00 UTC"
## [13] "2021-10-27 08:00:00 UTC" "2021-11-10 08:00:00 UTC"
## [15] "2021-11-24 08:00:00 UTC" "2021-12-08 08:00:00 UTC"
## [17] "2021-12-22 08:00:00 UTC" "2022-01-05 08:00:00 UTC"
## [19] "2022-01-19 08:00:00 UTC" "2022-02-02 08:00:00 UTC"
## [21] "2022-02-16 08:00:00 UTC" "2022-03-02 08:00:00 UTC"
## [23] "2022-03-16 08:00:00 UTC" "2022-03-30 08:00:00 UTC"
## [25] "2022-04-13 08:00:00 UTC" "2022-04-27 08:00:00 UTC"
```

6.6 The tricky thing about months

Due to the ambivalent nature of month addition/subtraction there are different implementations. Either with a simple "+" or with tidyverse's %m+%.

```
jan_31 = as.Date("2000-01-31")
# A sequence of 1 to 12 periods of 1 month
month_seq <- 1:12 * months(1)

# Add 1 to 12 months to jan_31
jan_31 + month_seq
```

```
##  [1] NA           "2000-03-31" NA           "2000-05-31" NA
##  [6] "2000-07-31" "2000-08-31" NA           "2000-10-31" NA
## [11] "2000-12-31" "2001-01-31"
```

```
# Replace + with %m+%
jan_31 %m+% month_seq
```

```
##  [1] "2000-02-29" "2000-03-31" "2000-04-30" "2000-05-31" "2000-06-30"
##  [6] "2000-07-31" "2000-08-31" "2000-09-30" "2000-10-31" "2000-11-30"
## [11] "2000-12-31" "2001-01-31"
```

```
# Replace + with %m-%
jan_31 %m-% month_seq
```

```
##  [1] "1999-12-31" "1999-11-30" "1999-10-31" "1999-09-30" "1999-08-31"
##  [6] "1999-07-31" "1999-06-30" "1999-05-31" "1999-04-30" "1999-03-31"
## [11] "1999-02-28" "1999-01-31"
```

6.7 Examining intervals. Reigns of kings and queens

Intervals can be created with the operator %–%.

```
# Print monarchs
monarchs

# Create an interval for reign
monarchs <- monarchs %>%
  mutate(reign = from %--% to)

# Find the length of reign, and arrange
monarchs %>%
  mutate(length = int_length(reign)) %>%
  arrange(desc(length)) %>%
  select(name, length, dominion)
```

6.8 Comparing intervals and datetimes

We use the keyword %within% wo check if a date is inside an interval.

```
# Print halleys
halleys

# New column for interval from start to end date
halleys <- halleys %>%
  mutate(visible = start_date %--% end_date)

# The visitation of 1066
halleys_1066 <- halleys[14, ]

# Monarchs in power on perihelion date
monarchs %>%
  filter(halleys_1066$perihelion_date %within% reign) %>%
  select(name, from, to, dominion)

# Monarchs whose reign overlaps visible time
monarchs %>%
  filter(int_overlaps(halleys_1066$visible, reign)) %>%
  select(name, from, to, dominion)
```

6.9 Converting to durations and periods

Conversion can be done with as.duration and as.period.

```
# New columns for duration and period
monarchs <- monarchs %>%
  mutate(
    duration = as.duration(reign),
    period = as.period(reign))

# Examine results
monarchs %>%
  select(name, duration, period)
```

7 Problems in practice

7.1 Setting the timezone

The function force_tz() can be used to set the timezone of a datetime.

```
# Game2: CAN vs NZL in Edmonton
game2 <- mdy_hm("June 11 2015 19:00")

# Game3: CHN vs NZL in Winnipeg
game3 <- mdy_hm("June 15 2015 18:30")

# Set the timezone to "America/Edmonton"
game2_local <- force_tz(game2, tzone = "America/Edmonton")
game2_local
```

```
## [1] "2015-06-11 19:00:00 MDT"
```

```
# Set the timezone to "America/Winnipeg"
game3_local <- force_tz(game3, tzone = "America/Winnipeg")
game3_local
```

```
## [1] "2015-06-15 18:30:00 CDT"
```

```
# How long does the team have to rest?
as.period(game2_local %--% game3_local)
```

```
## [1] "3d 22H 30M 0S"
```

7.2 Viewing in a timezone

The function with_tz() can be used to not set but only view a datetime in another timezone.

```
# What time is game2_local in NZ?
with_tz(game2_local, tzone = "Pacific/Auckland")
```

```
## [1] "2015-06-12 13:00:00 NZST"
```

```
# What time is game2_local in Corvallis, Oregon?
with_tz(game2_local, tzone = "America/Los_Angeles")
```

```
## [1] "2015-06-11 18:00:00 PDT"
```

```
# What time is game3_local in NZ?
with_tz(game3_local, tzone = "Pacific/Auckland")
```

```
## [1] "2015-06-16 11:30:00 NZST"
```

7.3 Timezones in the weather data

In this exercise we practice the use of force_tz().

```
# Examine datetime and date_utc columns
head(akl_hourly$datetime)
```

```
## [1] "2016-01-01 00:00:00 UTC" "2016-01-01 00:30:00 UTC"
## [3] "2016-01-01 01:00:00 UTC" "2016-01-01 01:30:00 UTC"
## [5] "2016-01-01 02:00:00 UTC" "2016-01-01 02:30:00 UTC"
```

```
head(akl_hourly$date_utc)
```

```
## [1] "2015-12-31 11:00:00 UTC" "2015-12-31 11:30:00 UTC"
## [3] "2015-12-31 12:00:00 UTC" "2015-12-31 12:30:00 UTC"
## [5] "2015-12-31 13:00:00 UTC" "2015-12-31 13:30:00 UTC"
```

```
# Force datetime to Pacific/Auckland
akl_hourly <- akl_hourly %>%
  mutate(
    datetime = force_tz(datetime, tzone = "Pacific/Auckland"))

# Reexamine datetime
head(akl_hourly$datetime)
```

```
## [1] "2016-01-01 00:00:00 NZDT" "2016-01-01 00:30:00 NZDT"
## [3] "2016-01-01 01:00:00 NZDT" "2016-01-01 01:30:00 NZDT"
## [5] "2016-01-01 02:00:00 NZDT" "2016-01-01 02:30:00 NZDT"
```

```
# Are datetime and date_utc the same moments
table(akl_hourly$datetime - akl_hourly$date_utc)
```

```
##
## -82800      0   3600
##      2  17450      2
```

7.4 Times without dates

In this exercise we learn that a hms class exists, which contains only time.

```
# Import auckland hourly data
akl_hourly <- read_csv("akl_weather_hourly_2016.csv")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   year = col_double(),
##   month = col_double(),
##   mday = col_double(),
##   time = col_time(format = ""),
##   temperature = col_double(),
##   weather = col_character(),
##   conditions = col_character(),
##   events = col_character(),
##   humidity = col_double(),
##   date_utc = col_datetime(format = "")
## )
```

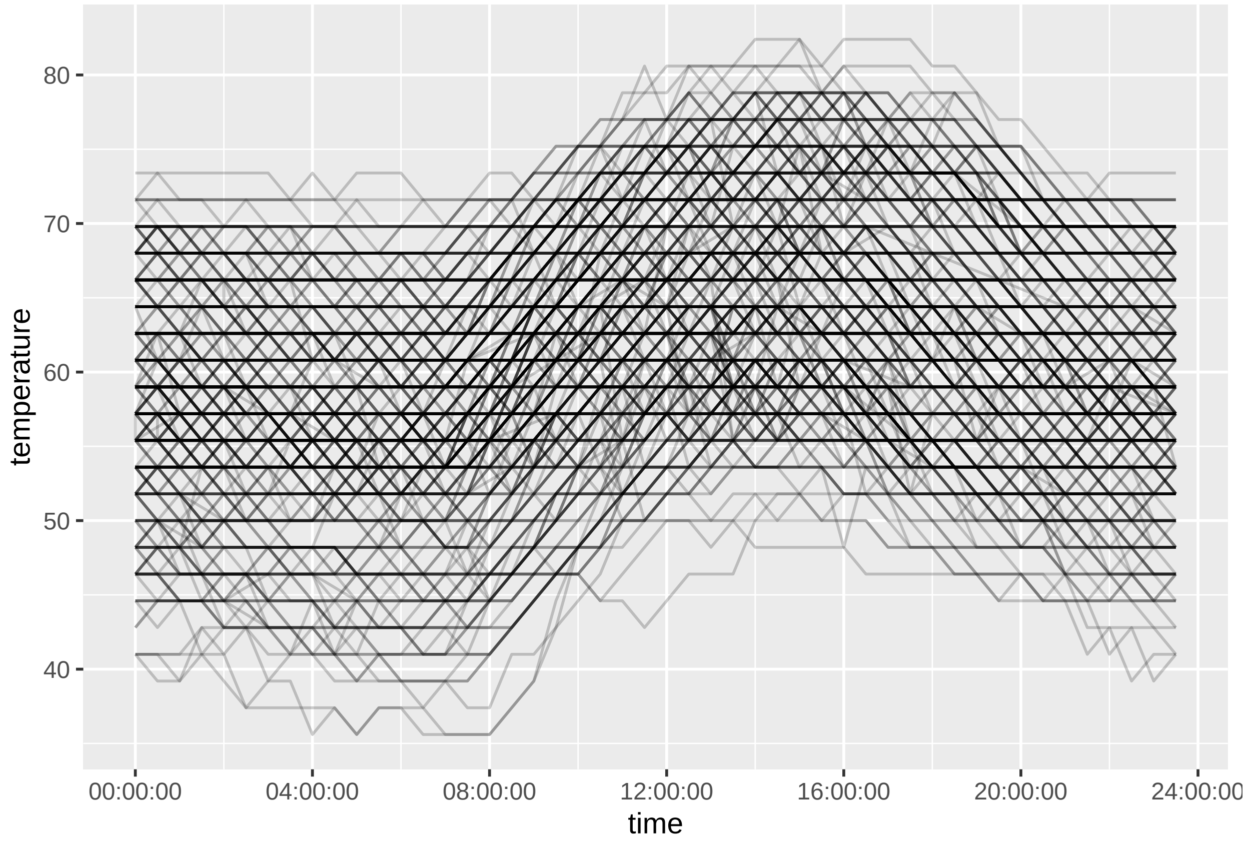
```
# Examine structure of time column
str(akl_hourly$time)
```

```
##  'hms' num [1:17454] 00:00:00 00:30:00 01:00:00 01:30:00 ...
##  - attr(*, "units")= chr "secs"
```

```
# Examine head of time column
head(akl_hourly$time)
```

```
## 00:00:00
## 00:30:00
## 01:00:00
## 01:30:00
## 02:00:00
## 02:30:00
```

```
# A plot using just time
ggplot(akl_hourly, aes(x = time, y = temperature)) +
  geom_line(aes(group = make_date(year, month, mday)), alpha = 0.2)
```



## Fast parsing with fasttime In the code below we compare the speed of ymd_hms and fastPOSIXct.

```
library(microbenchmark)
library(fasttime)

# Examine structure of dates
str(dates)

# Use fastPOSIXct() to parse dates
fastPOSIXct(dates) %>% str()

# Compare speed of fastPOSIXct() to ymd_hms()
microbenchmark(
  ymd_hms = ymd_hms(dates),
  fasttime = fastPOSIXct(dates),
  times = 20)
```

7.5 Fast parsing with libridate::fast_strptime

The function fast-strptime is even faster, but a explicit format has to be specified.

```
# Head of dates
head(dates)

# Parse dates with fast_strptime
fast_strptime(dates,
    format = "%Y-%m-%dT%H:%M:%SZ") %>% str()

# Comparse speed to ymd_hms() and fasttime
microbenchmark(
  ymd_hms = ymd_hms(dates),
  fasttime = fastPOSIXct(dates),
  fast_strptime = fast_strptime(dates,
    format = "%Y-%m-%dT%H:%M:%SZ"),
  times = 20)
```

7.6 Outputting pretty dates and times

The function stamp() can be used to output dates and takes a format string and outputs a function.

```
# Create a stamp based on "Saturday, Jan 1, 2000"
date_stamp <- stamp("Saturday, Jan 1, 2000")

# Print date_stamp
print(date_stamp)

# Call date_stamp on today()
date_stamp(today())

# Create and call a stamp based on "12/31/1999"
stamp("12/31/1999")(today())

# Use string finished for stamp()
stamp(finished)(today())
```