Time-Series data

Teodor Chakarov

2022-04-14

Contents

Tutorium in R

Exercise: Time-Series data - Number 5

By: Teodor Chakarov 12141198

Date and times in R

```
# Load the readr package
library(readr)
library(anytime)
library(ggplot2)
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(magrittr)
library(lubridate)
```

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

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

```
releases <- read_csv("releases.csv")
```

```
## Rows: 105 Columns: 7
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
```

```
## chr  (1): type
## dbl  (3): major, minor, patch
## dttm (1): datetime
## date (1): date
## time (1): time
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

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

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

Dates

```
##  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")
str(april_10_2014)
```

```
##  Date[1:1], format: "2014-04-10"
```

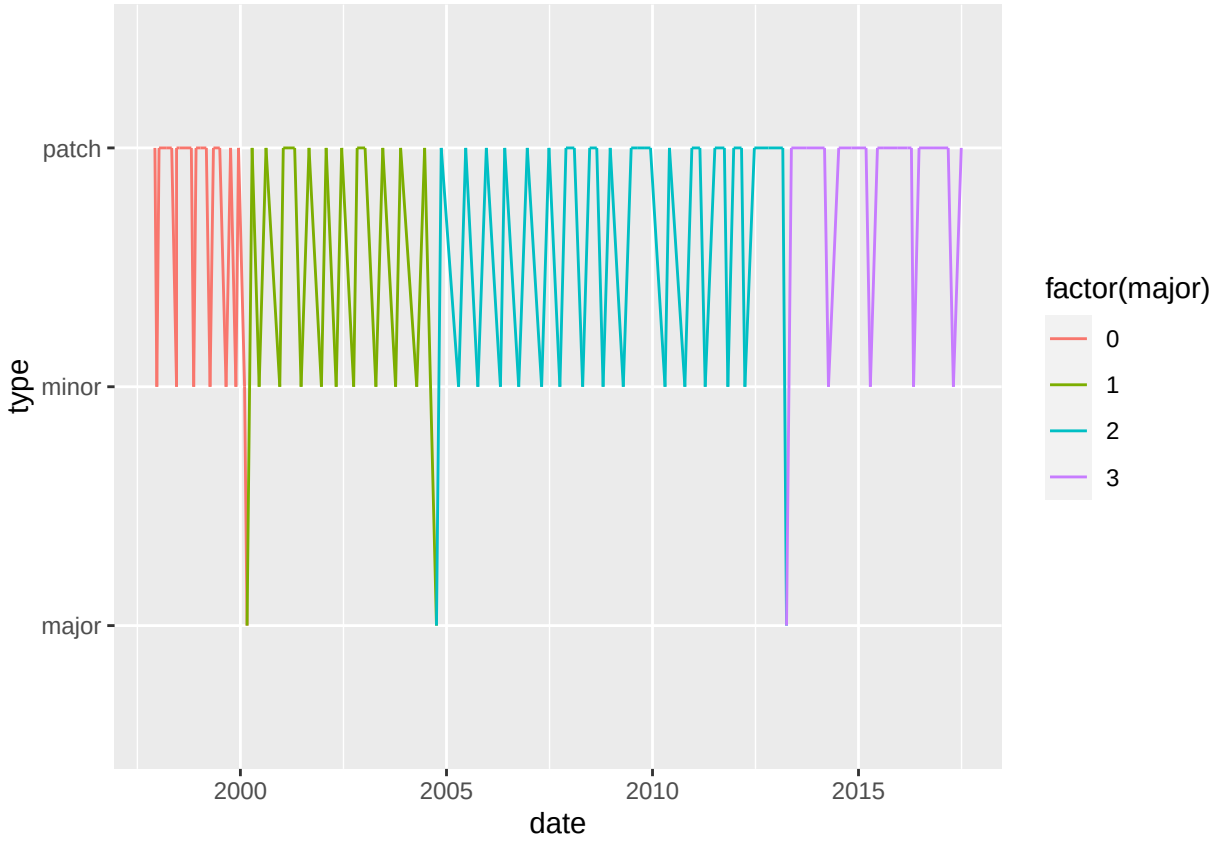
```
# Examine the structure of the date column
str(releases$date)
```

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

```
# 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)
```

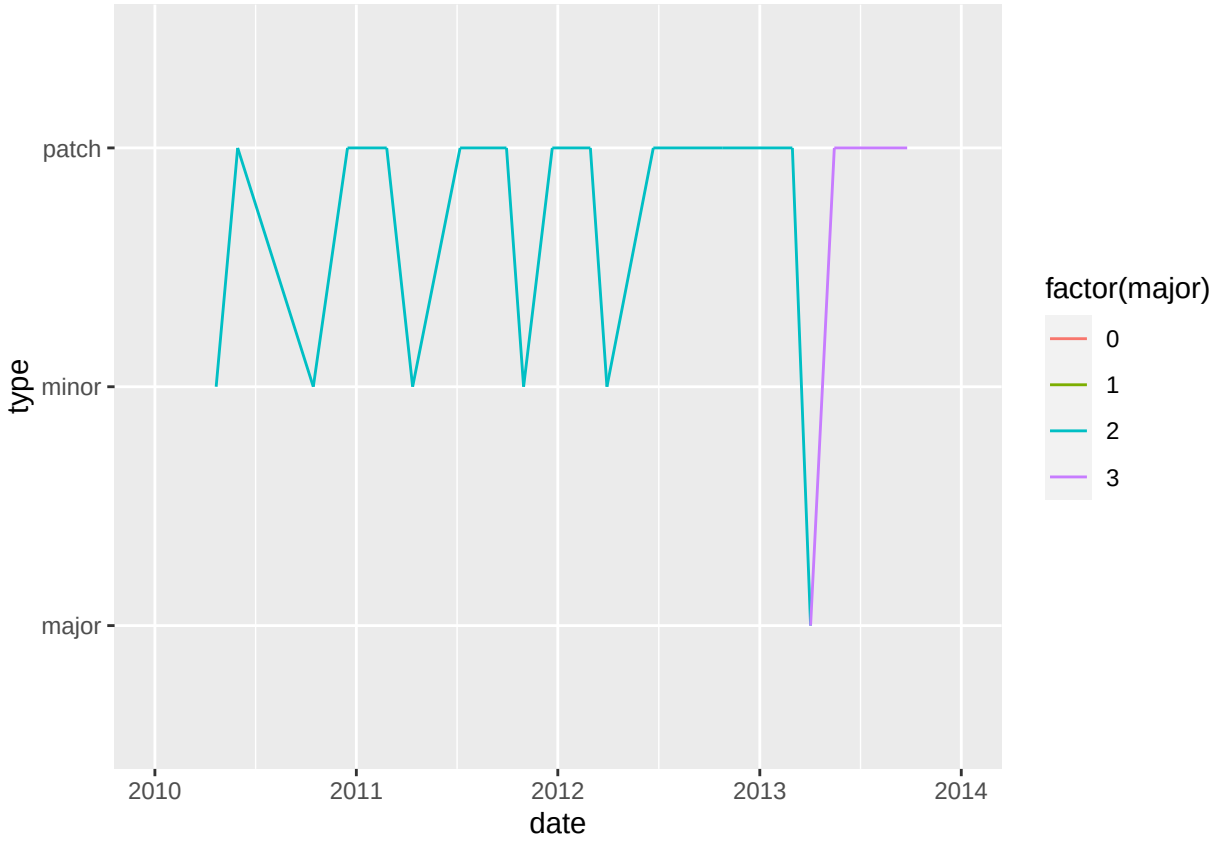
```
## [1] "2009-09-10 CEST" "2009-09-10 CEST" "2009-09-10 CEST" "2009-09-10 CEST"
```

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

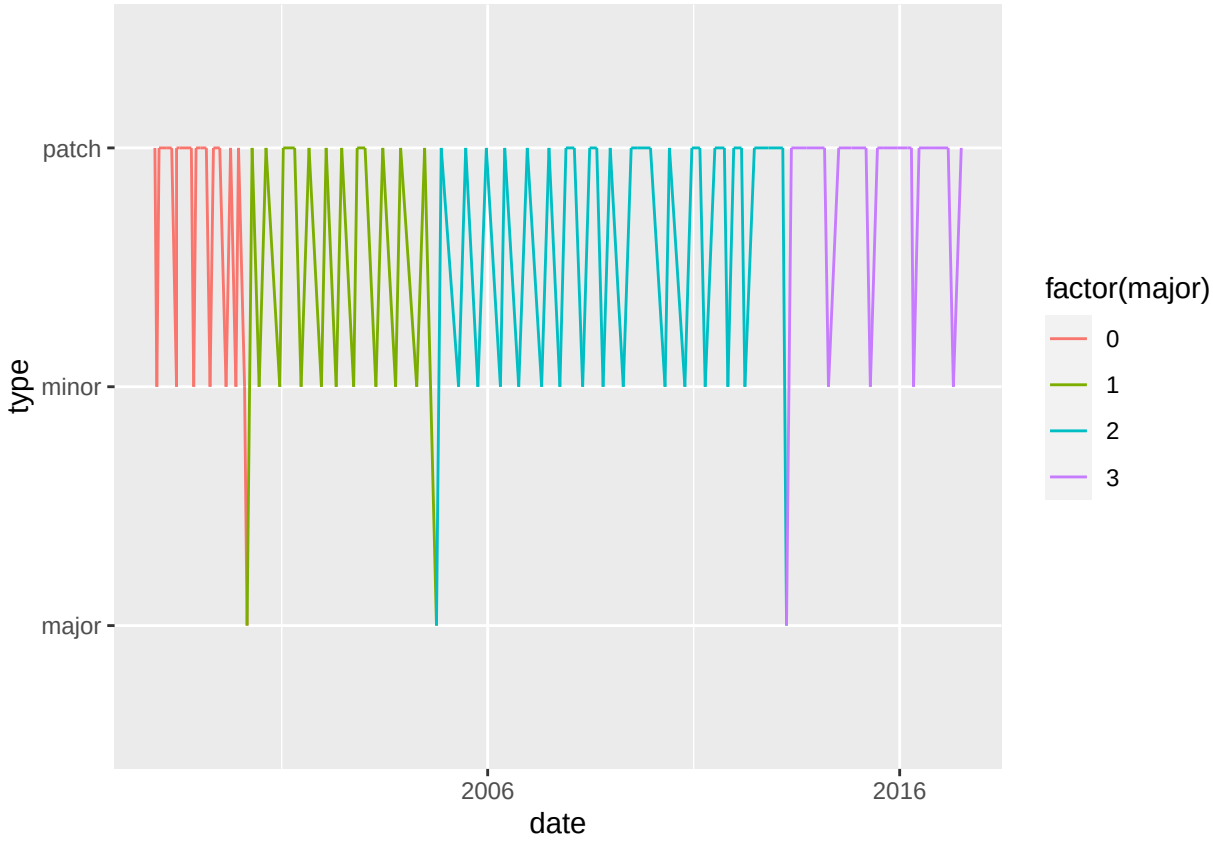



```
# Limit the axis to between 2010-01-01 and 2014-01-01
ggplot(releases, aes(x = date, y = type)) +
  geom_line(aes(group = 1, color = factor(major))) +
  xlim(as.Date("2010-01-01"), as.Date("2014-01-01"))
```

```
## 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")
```




```
# 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
```

```
## Time difference of 1761 days
```

```

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

Times

```
## [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"
```

```
# 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" ...
```

```
# Import csv
logs <- read_csv("logs.csv")
```

```
## Rows: 100000 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (2): r_version, country
## dttm (1): datetime
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# Print logs
print(logs)
```

```
## # A tibble: 100,000 x 3
##    datetime            r_version country
##    <dttm>              <chr>     <chr>
##  1 2015-04-16 22:40:19 3.1.3     CO
##  2 2015-04-16 09:11:04 3.1.3     GB
##  3 2015-04-16 17:12:37 3.1.3     DE
##  4 2015-04-18 12:34:43 3.2.0     GB
##  5 2015-04-16 04:49:18 3.1.3     PE
##  6 2015-04-16 06:40:44 3.1.3     TW
##  7 2015-04-16 00:21:36 3.1.3     US
##  8 2015-04-16 10:27:23 3.1.3     US
##  9 2015-04-16 01:59:43 3.1.3     SG
## 10 2015-04-18 15:41:32 3.2.0     CA
## # ... with 99,990 more rows
```

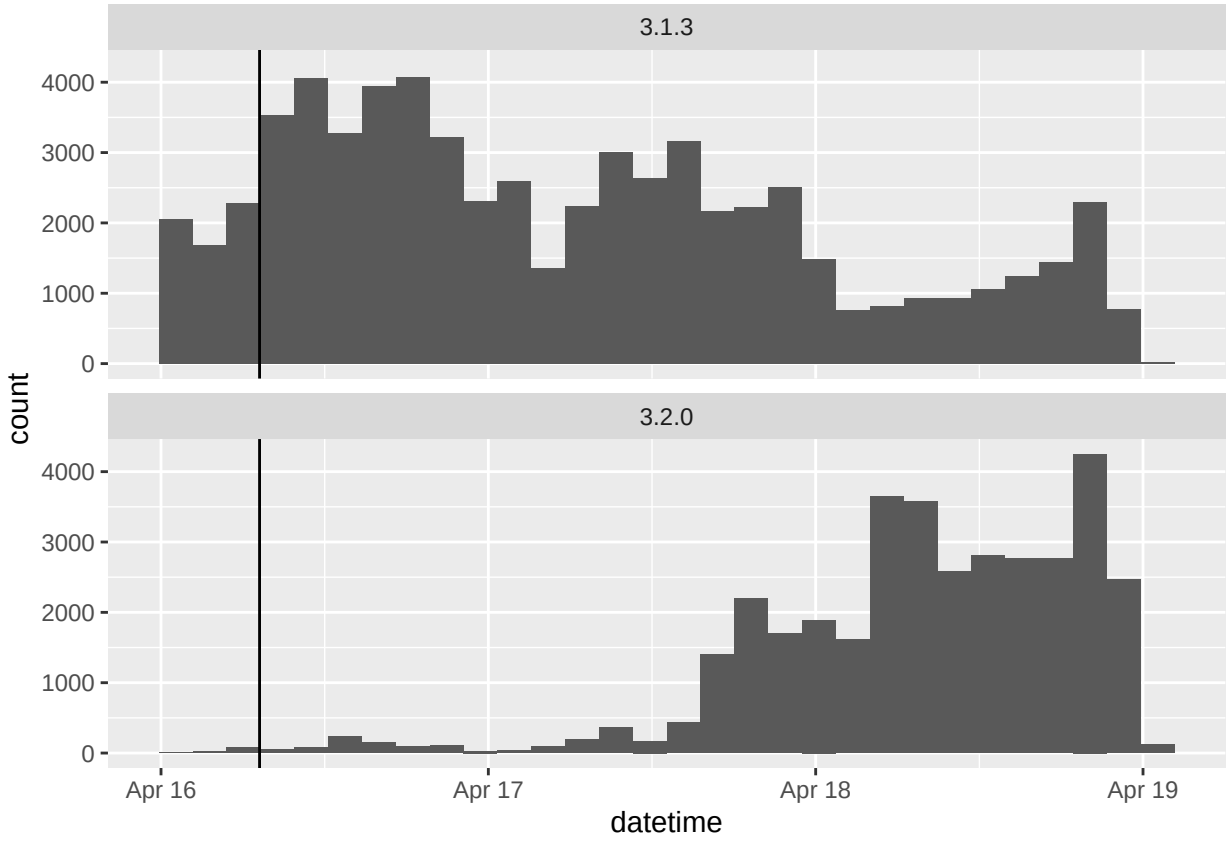
```
# Store the release time as a POSIXct object
release_time <- as.POSIXct("2015-04-16 07:13:33", tz = "UTC")

# When is the first download of 3.2.0?
logs %>%
  filter(datetime > release_time,
    r_version == "3.2.0")
```

```
## # A tibble: 35,826 x 3
##    datetime            r_version country
##    <dttm>              <chr>     <chr>
##  1 2015-04-18 12:34:43 3.2.0     GB
##  2 2015-04-18 15:41:32 3.2.0     CA
##  3 2015-04-18 14:58:41 3.2.0     IE
##  4 2015-04-18 16:44:45 3.2.0     US
##  5 2015-04-18 04:34:35 3.2.0     US
##  6 2015-04-18 22:29:45 3.2.0     CH
##  7 2015-04-17 16:21:06 3.2.0     US
##  8 2015-04-18 20:34:57 3.2.0     AT
##  9 2015-04-17 18:23:19 3.2.0     US
## 10 2015-04-18 03:00:31 3.2.0     US
```

```
## # ... with 35,816 more rows
```

```
# 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)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




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

Manipulating data with lubridate

```
## [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"
```

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

```
## [1] "2010-06-01 16:00:00 UTC"
```

```
# 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"
```

Working with AKL weather data

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

```
## Rows: 3661 Columns: 7
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): date, events
## dbl (5): max_temp, min_temp, mean_temp, mean_rh, cloud_cover
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
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
```

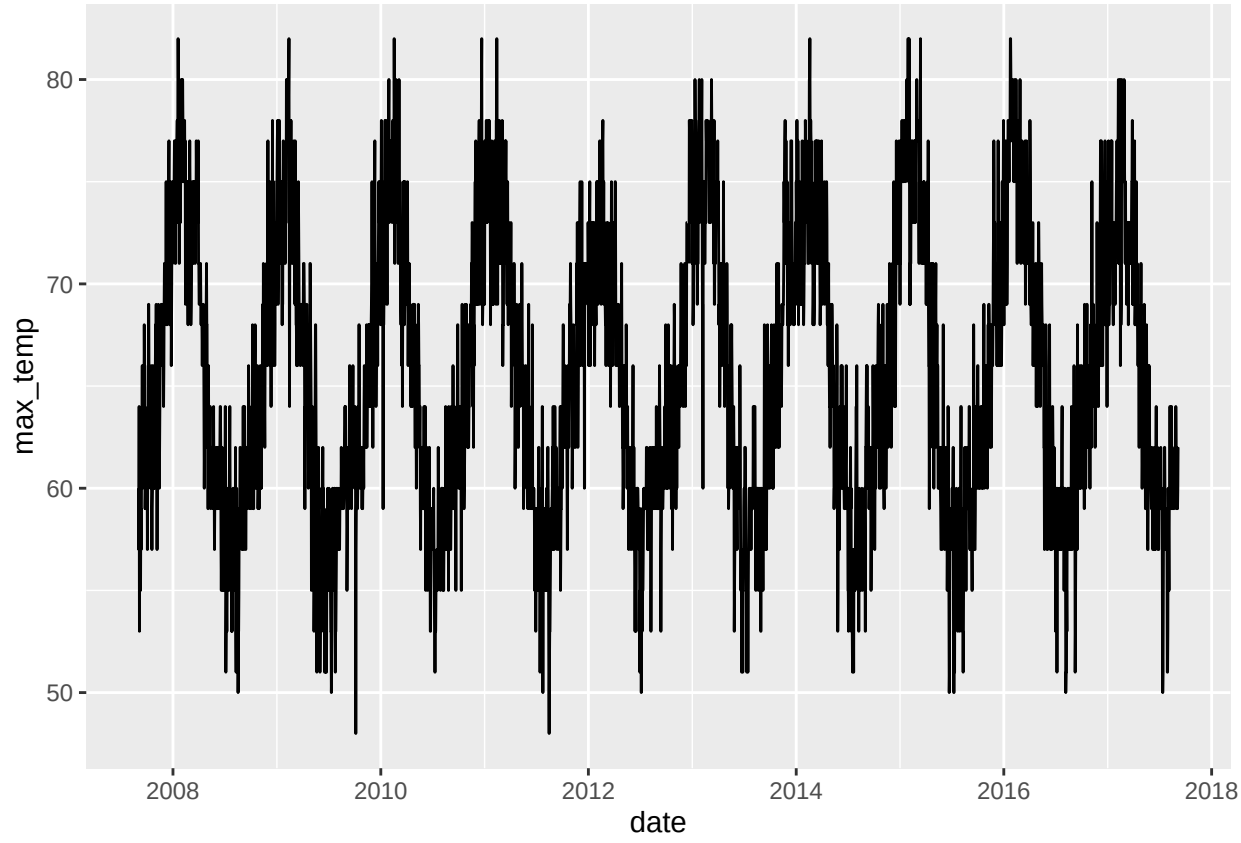
```
# Parse date
akl_daily <- akl_daily_raw %>%
  mutate(date = ymd(date))
```

```
# Print akl_daily
print(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 "akl_weather_hourly_2016.csv"
akl_hourly_raw <- read_csv("hourly.csv")
```

```
## Rows: 17454 Columns: 10
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (3): weather, conditions, events
## dbl  (5): year, month, mday, temperature, humidity
## dttm (1): date_utc
## time (1): time
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
akl_hourly_raw
```

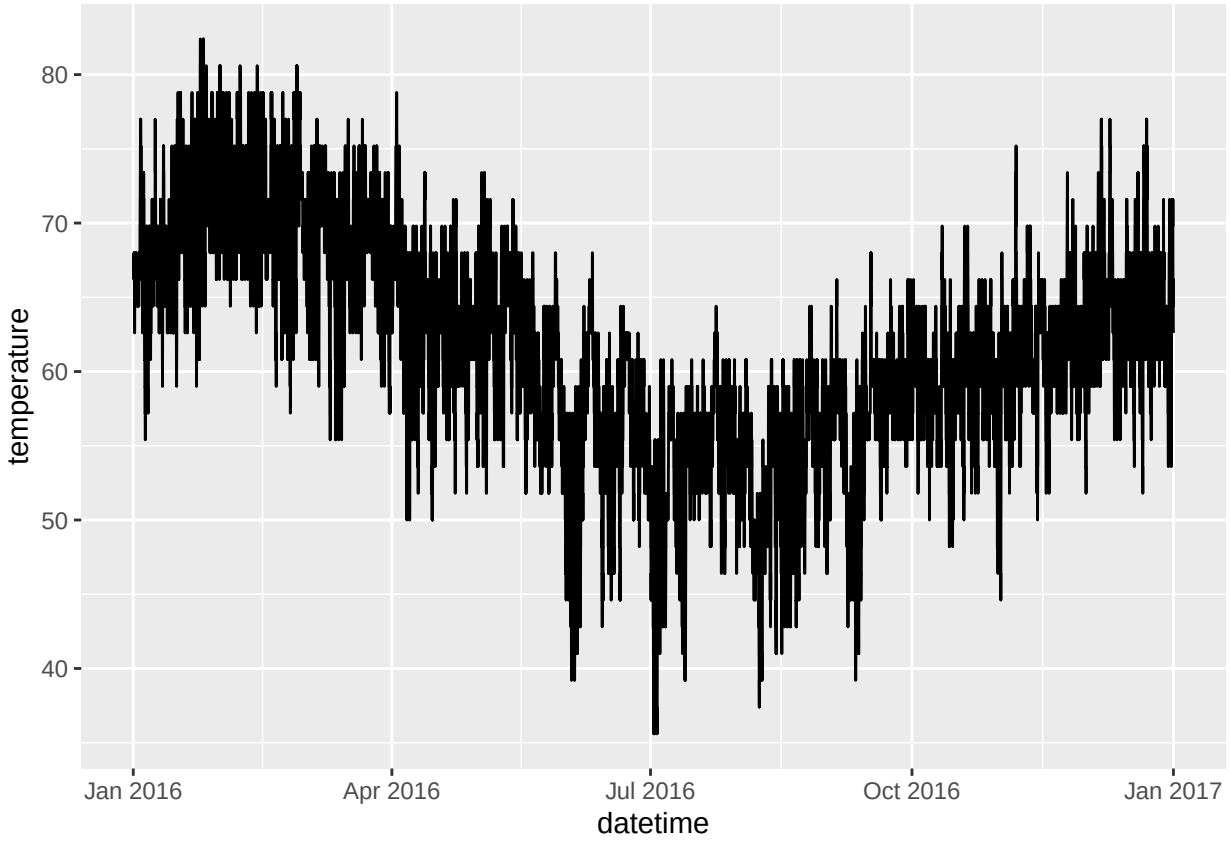
```
## # A tibble: 17,454 x 10
##     year month  mday time   temperature weather conditions       events humidity
##    <dbl> <dbl> <dbl> <time>       <dbl> <chr>   <chr>            <chr>     <dbl>
##  1  2016     1     1 00:00         68   Clear   Clear            <NA>         68
##  2  2016     1     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>
```

```
# 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()
```




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

Extracting parts of the date-time

```
## [1] "2015-04-16 07:13:33 UTC"
```

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

```
## [1] 4
```

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

```
## .
## 4
## 1
```

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

```
## .
```

```
## 2015
##    1
```

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

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

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

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

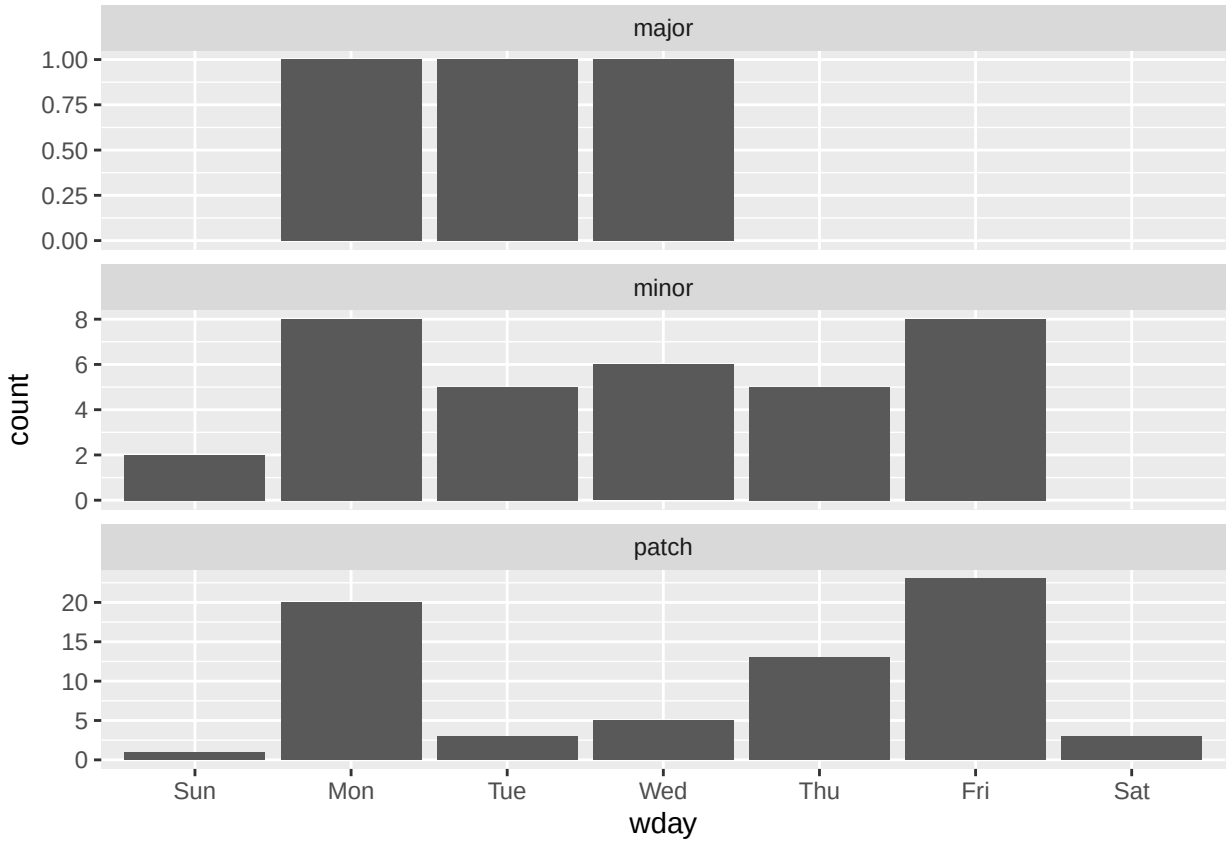
```
# Use wday() to tabulate release by day of the week
wday(releases$datetime) %>% table()
```

```
## .
##  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()
```

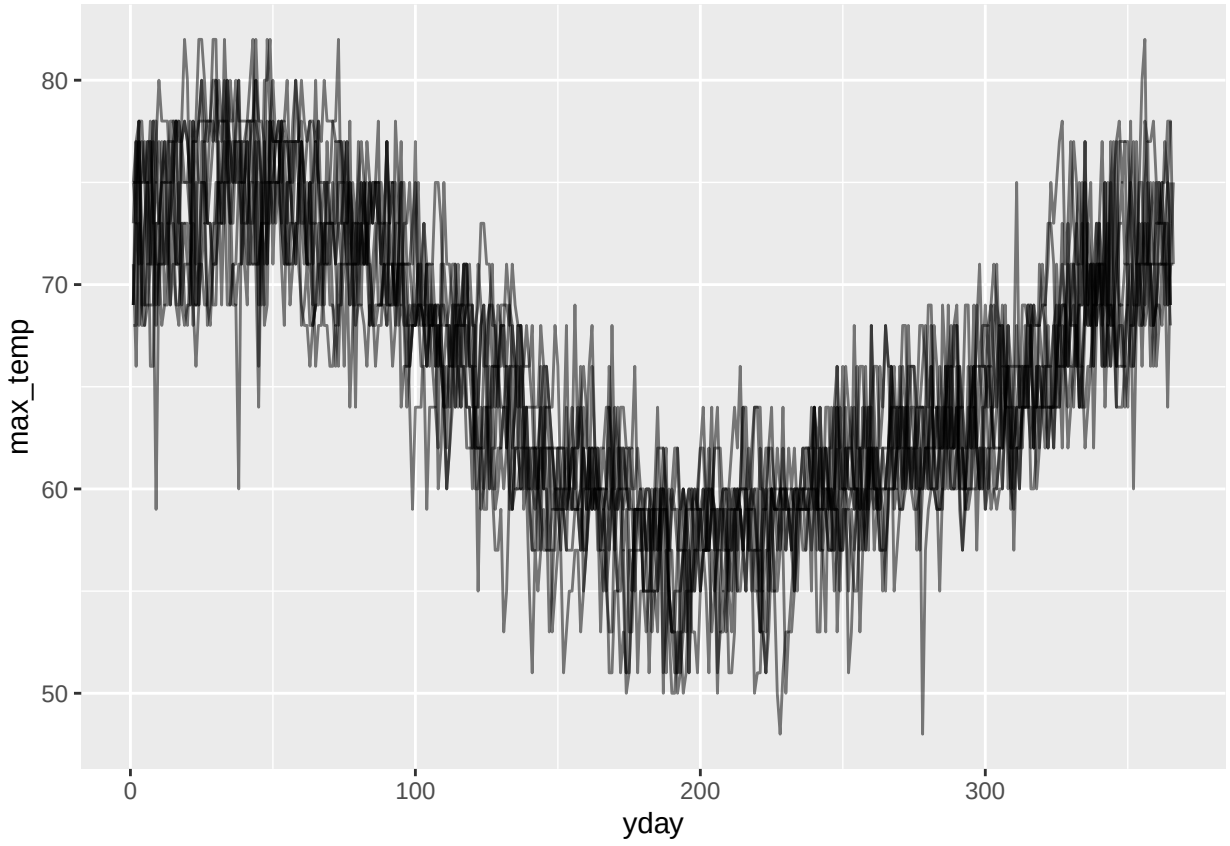
```
## .
## Sun Mon Tue Wed Thu Fri Sat
##   3  29   9  12  18  31   3
```

```
# 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")
```



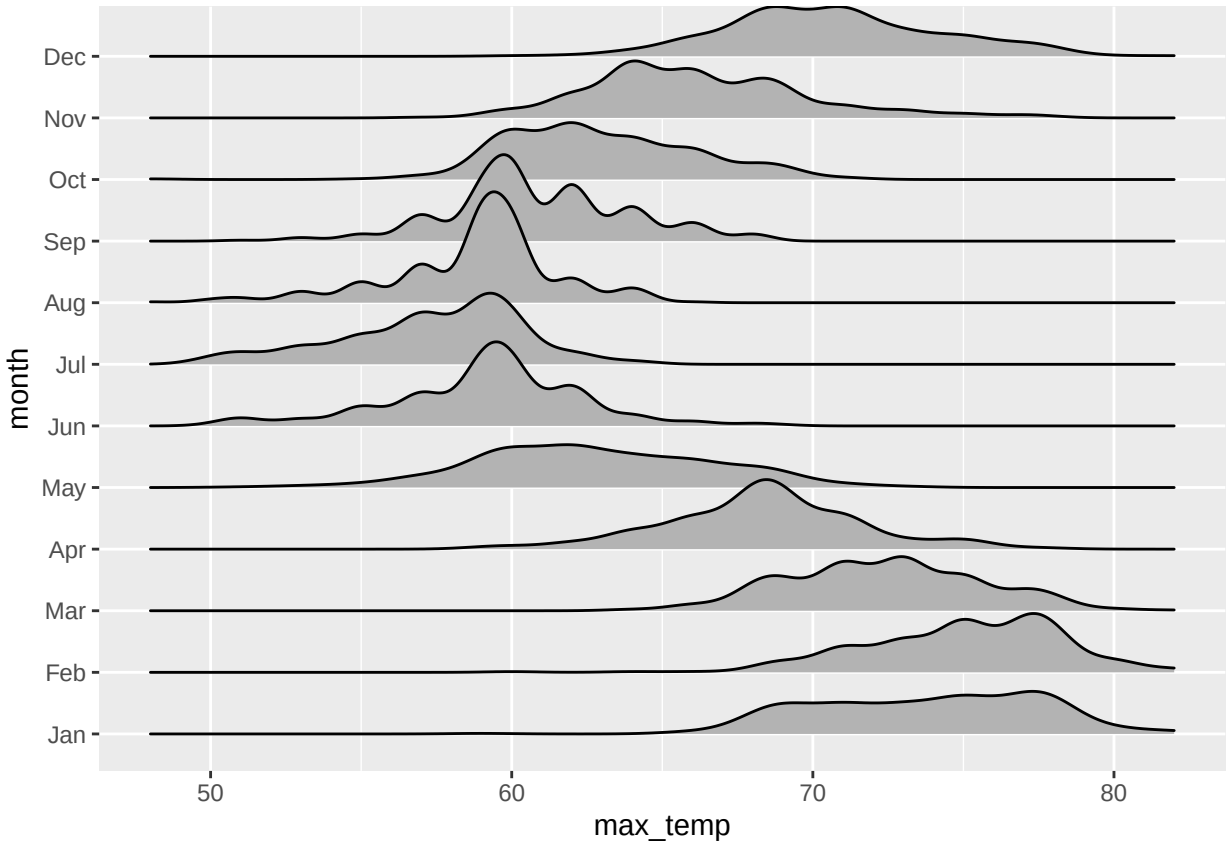

```
# Add columns for year, yday and month
akl_daily <- akl_daily %>%
  mutate(
    year = year(date),
    yday = yday(date),
    month = month(date, label = TRUE))

# 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).
```




```
library(ggridges)
# 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 stigmatization

```
# 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.
```

```
# 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 Jan           15
##  2 Feb           13
##  3 Mar           12
##  4 Apr           15
##  5 May           21
##  6 Jun           19
##  7 Jul           22
##  8 Aug           16
##  9 Sep           25
## 10 Oct           20
## 11 Nov           19
## 12 Dec           11
```

```
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
```

Rounding the 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
```

Taking differences of datetimes

datetime_1 - datetime_2: subtraction for time elapsed datetime_1 + (2 * timespan): addition and multiplication for generating new datetimes in the past or future timespan1 / timespan2: division for change of units

```
# 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 19273 days
```

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

```
## Time difference of 1665241399 secs
```

```
# 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
```

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

Timespan:

```
## [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 + hours(81)
```

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

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

```
## [1] "2017-04-26"
```

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

```
## [1] "2017-04-25 18:00:00 UTC"
```

```
# 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
```

Arithmetic operations with date-time

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

```
# 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
today_8am + every_two_weeks
```

```
##  [1] "2022-05-10 08:00:00 UTC" "2022-05-24 08:00:00 UTC"
##  [3] "2022-06-07 08:00:00 UTC" "2022-06-21 08:00:00 UTC"
##  [5] "2022-07-05 08:00:00 UTC" "2022-07-19 08:00:00 UTC"
##  [7] "2022-08-02 08:00:00 UTC" "2022-08-16 08:00:00 UTC"
##  [9] "2022-08-30 08:00:00 UTC" "2022-09-13 08:00:00 UTC"
## [11] "2022-09-27 08:00:00 UTC" "2022-10-11 08:00:00 UTC"
## [13] "2022-10-25 08:00:00 UTC" "2022-11-08 08:00:00 UTC"
## [15] "2022-11-22 08:00:00 UTC" "2022-12-06 08:00:00 UTC"
## [17] "2022-12-20 08:00:00 UTC" "2023-01-03 08:00:00 UTC"
## [19] "2023-01-17 08:00:00 UTC" "2023-01-31 08:00:00 UTC"
## [21] "2023-02-14 08:00:00 UTC" "2023-02-28 08:00:00 UTC"
## [23] "2023-03-14 08:00:00 UTC" "2023-03-28 08:00:00 UTC"
## [25] "2023-04-11 08:00:00 UTC" "2023-04-25 08:00:00 UTC"
```

```
jan_31 <- ymd("2020-01-31")
# 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           "2020-03-31" NA           "2020-05-31" NA
##  [6] "2020-07-31" "2020-08-31" NA           "2020-10-31" NA
## [11] "2020-12-31" "2021-01-31"
```

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

```
##  [1] "2020-02-29" "2020-03-31" "2020-04-30" "2020-05-31" "2020-06-30"
##  [6] "2020-07-31" "2020-08-31" "2020-09-30" "2020-10-31" "2020-11-30"
## [11] "2020-12-31" "2021-01-31"
```

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

```
##  [1] "2019-12-31" "2019-11-30" "2019-10-31" "2019-09-30" "2019-08-31"
##  [6] "2019-07-31" "2019-06-30" "2019-05-31" "2019-04-30" "2019-03-31"
## [11] "2019-02-28" "2019-01-31"
```

```
# 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)
```

```
# 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)
```

Intervals:

```
# 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
```

Time Zones

```
## [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"
```

```
# 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"
```

```
# Examine datetime and date_utc
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
```

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

```
## Rows: 17454 Columns: 10
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (3): weather, conditions, events
## dbl  (5): year, month, mday, temperature, humidity
## dttm (1): date_utc
## time (1): time
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

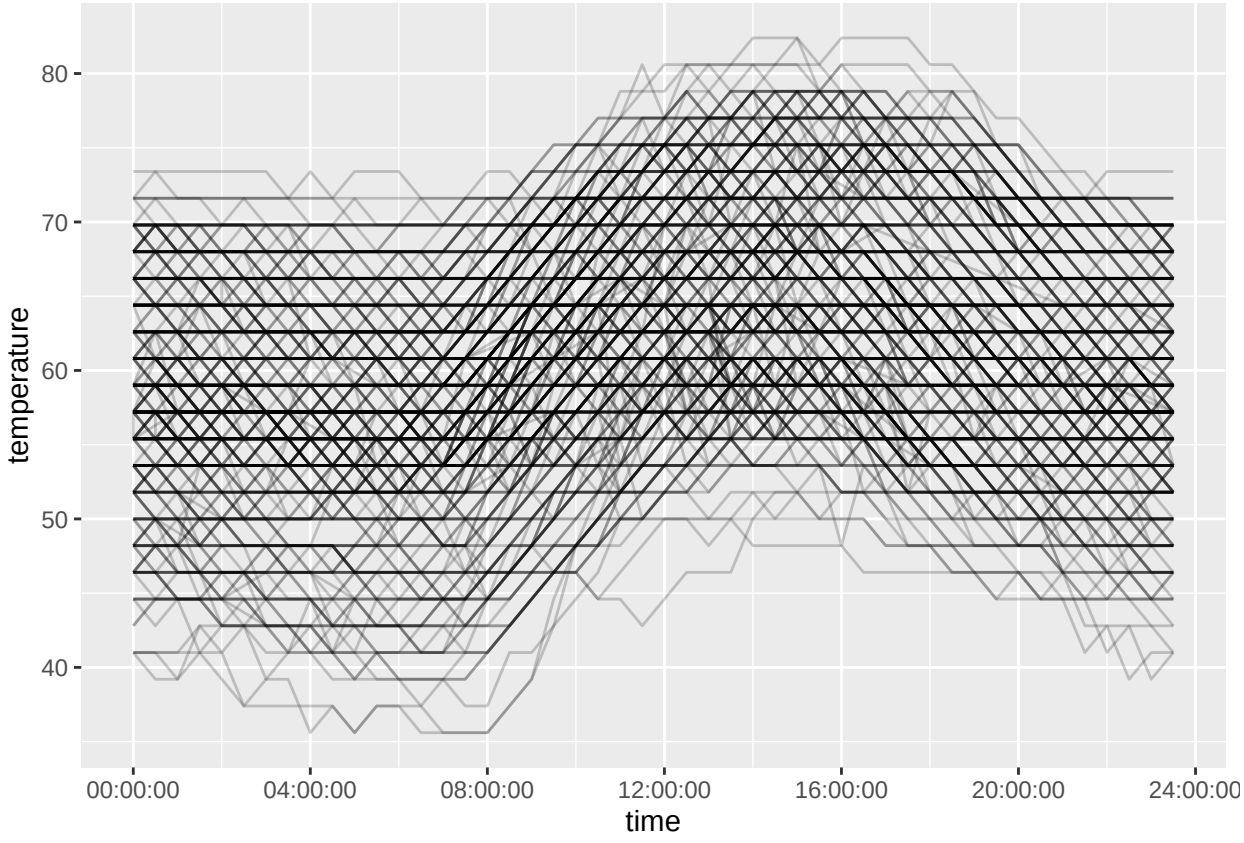
```
# Examine structure of time column
str(akl_hourly$time)
```

```
##  '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)
```




```
library(microbenchmark)
library(fasttime)

dates <- as.character(with_tz(akl_hourly$date_utc, tzone = "Pacific/Auckland"))

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

```
##  chr [1:17454] "2016-01-01 00:00:00" "2016-01-01 00:30:00" ...
```

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

```
##  POSIXct[1:17454], format: "2016-01-01 01:00:00" "2016-01-01 01:30:00" "2016-01-01 02:00:00" ...
```

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

```
## Unit: microseconds
##      expr     min       lq      mean   median       uq     max neval
##   ymd_hms 16335.4 16554.65 17496.795 17169.55 17375.65 23094.2    20
##  fasttime   798.8   806.25   822.175   817.05   832.65   878.6    20
```

```
# Head of dates
head(dates)
```

```
## [1] "2016-01-01 00:00:00" "2016-01-01 00:30:00" "2016-01-01 01:00:00"
## [4] "2016-01-01 01:30:00" "2016-01-01 02:00:00" "2016-01-01 02:30:00"
```

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

```
##  POSIXlt[1:17454], format: "2016-01-01 00:00:00" "2016-01-01 00:30:00" "2016-01-01 01:00:00" ...
```

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

```
## Unit: microseconds
##           expr     min       lq      mean   median       uq     max neval
##        ymd_hms 16363.4 16822.20 17587.840 17180.45 17384.75 22918.1    20
##       fasttime   798.8   817.15   841.395   824.45   856.25  1036.1    20
##  fast_strptime   948.7   975.75  1233.540   997.35  1044.05  5552.0    20
```

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

```
## Multiple formats matched: "%A, %b %d, %Y"(1), "Saturday, Jan %Om, %Y"(1), "Saturday, %Om %d, %Y"(1),
```

```
## Using: "%A, %b %d, %Y"
```

```
# Print date_stamp
date_stamp
```

```
## function (x, locale = "English_United States.1252")
## {
##     {
##         old_lc_time <- Sys.getlocale("LC_TIME")
##         if (old_lc_time != locale) {
##             on.exit(Sys.setlocale("LC_TIME", old_lc_time))
##             Sys.setlocale("LC_TIME", locale)
##         }
##     }
##     format(x, format = "%A, %b %d, %Y")
## }
## <environment: 0x00000000267c1fc8>
```

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

```
## [1] "Tuesday, Apr 26, 2022"
```

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

```
## Multiple formats matched: "%Om/%d/%Y"(1), "%m/%d/%Y"(1)
```

```
## Using: "%Om/%d/%Y"
```

```
## [1] "04/26/2022"
```

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