Dates-Times

Hamid Semiyari

Learning Objectives

- Manipulating dates and times.
- Chapter 16 of RDS.
- Dates and Times Cheat Sheet.
- Lubridate Overview.

Parsing Dates

- The lubridate package has a bunch of convenience functions for working with dates. It is *not* a part of the tidyverse, so you need to load it separately.

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

- There are three main classes for date/time data:
  - `Date` for just the date.
  - `POSIXct` for both the date and the time. "POSIXct" stands for "Portable Operating System Interface Calendar Time" (don't ask me where the "X" comes from). It is a part of a standardized system of representing time across many computing computing platforms.
  - `hms` from the hms R package for just the time. "hms" stands for "hours, minutes, and seconds."
- `today()` will give you the current date in the `Date` class.

```
today()
```

```
[1] "2024-02-13"
```

```
class(today())
```

```
[1] "Date"
```

- `now()` will give you the current date-time in the `POSIXct` class.

```
now()
```

```
[1] "2024-02-13 18:42:24 EST"
```

```
class(now())
```

```
[1] "POSIXct" "POSIXt"
```

- There is no built-in R function to find the current time without the date. But you can use `hms::as_hms(now())` to get the current time.

```
hms::as_hms(now())
```

```
18:42:24.328115
```

```
class(hms::as_hms(now()))
```

```
[1] "hms"      "difftime"
```

Parsing Dates

- You can use `parse_date()`, `parse_datetime()`, and `parse_time()` to parse a date/date-time/time from a string.

```
x <- parse_date("10/11/2020", format = "%m/%d/%Y")
x
```

```
[1] "2020-10-11"
```

```
class(x)
```

```
[1] "Date"
```

```
y <- parse_datetime("10/11/2020 11:59:20", format = "%m/%d/%Y %H:%M:%S")
y
```

```
[1] "2020-10-11 11:59:20 UTC"
```

```
class(y)
```

```
[1] "POSIXct" "POSIXt"
```

```
z <- parse_time("11:59:20", "%H:%M:%S")
z
```

```
11:59:20
```

```
class(z)
```

```
[1] "hms"      "difftime"
```

- lubridate comes with a bunch of helper functions to parse dates more automatically. The helper function name itself specifies the order of the year, month, day, hours, minutes, and seconds.
- To parse dates, look at the help page of

```
help(ymd)
```

```
## Only the order of year, month, and day matters
ymd(c("2011/01-10", "2011-01/10", "20110110"))
```

```
[1] "2011-01-10" "2011-01-10" "2011-01-10"
```

```
mdy(c("01/10/2011", "01 adsl; 10 df 2011", "January 10, 2011"))
```

```
[1] "2011-01-10" "2011-01-10" "2011-01-10"
```

- To parse times, look at the help page of

```
help(ms)
```

```
starting httpd help server ... done
```

```
## only the order of hours, minutes, and seconds matter
hms(c("10:40:10", "10 40 10"))
```

```
[1] "10H 40M 10S" "10H 40M 10S"
```

- Note that `ms()`, `hm()`, and `hms()` won't recognize "-" as a separator because it treats it as negative time. So use `parse_time()` here.

```
ms("10-10")
```

```
[1] "10M -10S"
```

- To parse date-times, look at the help page of

```
help(ymd_hms)
```

- More generally, you can choose the order of elements with `parse_date_time()`, which has a different and easier syntax than `readr::parse_datetime()`.

```
parse_date_time("11, 22, 01 here is a trap! 11/02/2002", orders = "HMSmdy")
```

```
[1] "2002-11-02 11:22:01 UTC"
```

- **Exercise**: Parse the following date-times.

```
"05/26/2004 UTC 11:11:11.444"
"26 2004 05 UTC 11/11/11.444"
```

- **Exercise** (RDS 16.2.4.3): 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
```

Dates from individual components

- If you have a vector of years, months, days, hours, minutes, or seconds, you can use `make_date()` or `make_datetime()` to create dates and date-times.

```
make_date(year = 1981, month = 6, day = 25)
```

```
[1] "1981-06-25"
```

```
make_datetime(year = 1972, month = 2, day = 22, hour = 10, min = 9, sec = 01)
```

```
[1] "1972-02-22 10:09:01 UTC"
```

- nycflights13 example:

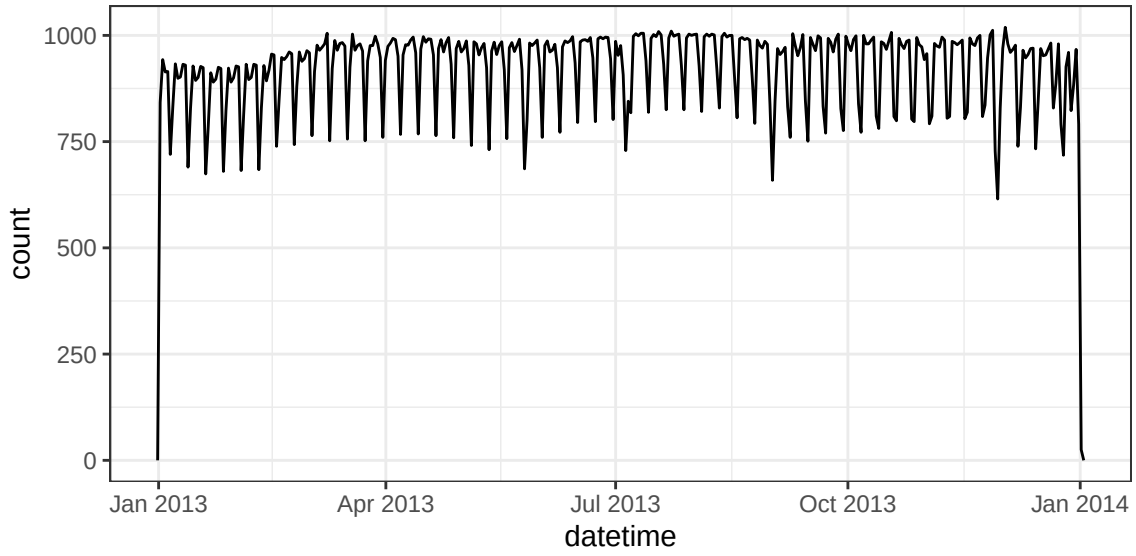
```
library(nycflights13)
data("flights")
flights %>%
  mutate(datetime = make_datetime(year   = year,
                                  month  = month,
                                  day    = day,
                                  hour   = hour,
                                  min = minute)) ->
  flights
  select(flights, datetime)
```

```
# A tibble: 336,776 x 1
   datetime
   <dttm>
 1 2013-01-01 05:15:00
 2 2013-01-01 05:29:00
 3 2013-01-01 05:40:00
```

```
 4 2013-01-01 05:45:00
 5 2013-01-01 06:00:00
 6 2013-01-01 05:58:00
 7 2013-01-01 06:00:00
 8 2013-01-01 06:00:00
 9 2013-01-01 06:00:00
10 2013-01-01 06:00:00
# i 336,766 more rows
```

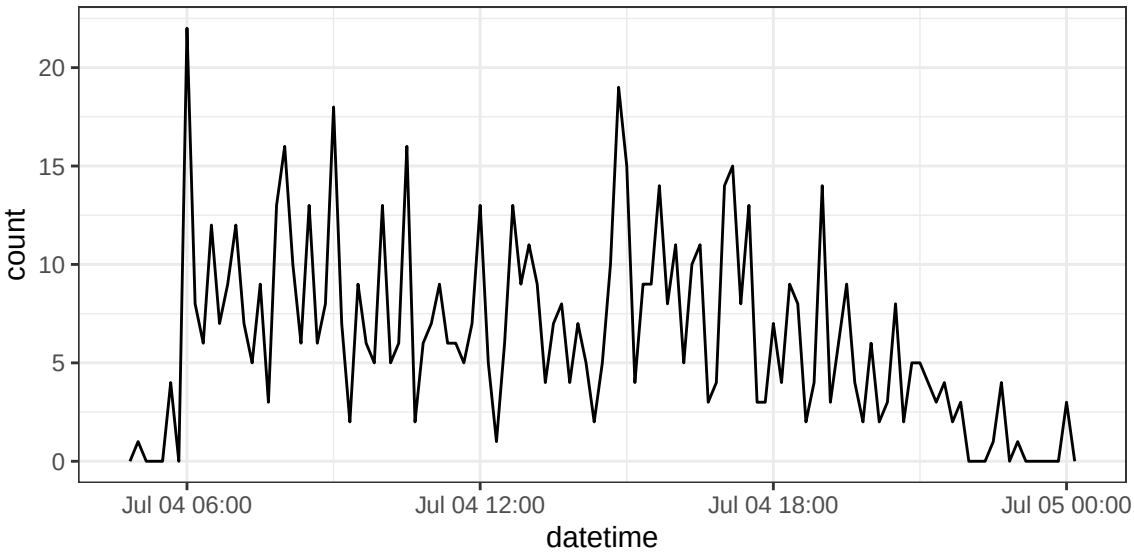
- Having it in the date-time format makes it easier to plot.

```
ggplot(flights, aes(x = datetime)) +
  geom_freqpoly(bins = 365)
```



- It makes it easier to filter by date

```
flights %>%
  filter(as_date(datetime) == ymd(20130704)) %>%
  ggplot(aes(x = datetime)) +
  geom_freqpoly(binwidth = 600)
```



- I used `as_date()` in the previous example. This function will try to coerce an object to a date. Sometimes successfully! It is particularly useful for extracting the date component of a `POSIXct` object.
- `as_datetime()` tries to coerce an object to a `POSIXct` object.
- **Exercise**: Create a date variable from the following data frame. Then filter out all rows before Feb 1, 2010. If you finish early, try converting the month variable to the numeric representation of the month.

```
fake <- tribble(~year, ~month, ~day, ~month_num,
                ##----/-------/----------------
                2018,  "Oct",  1,    10,
                2011,  "Nov",  2,    11,
                2019,  "Dec",  3,    12,
                2010,  "JAN",  5,     1,
                1999,  "MAr",  1,     3,
                1987,  "ApR",  3,     4,
                2020,  "maY",  2,     5,
                2010,  "May",  4,     5)
```

Extracting Components

- `year()` extracts the year.
- `month()` extracts the month.

- `week()` extracts the week.
- `mday()` extracts the day of the month (1, 2, 3, …).
- `wday()` extracts the day of the week (Saturday, Sunday, Monday …).
- `yday()` extracts the day of the year (1, 2, 3, …)
- `hour()` extracts the hour.
- `minute()` extract the minute.
- `second()` extracts the second.

```
ddat <- mdy_hms("01/02/1970 03:51:44")
ddat
```

```
[1] "1970-01-02 03:51:44 UTC"
```

```
year(ddat)
```

```
[1] 1970
```

```
month(ddat, label = TRUE)
```

```
[1] Jan
12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Sep < ... < Dec
```

```
week(ddat)
```

```
[1] 1
```

```
mday(ddat)
```

```
[1] 2
```

```
wday(ddat, label = TRUE)
```

```
[1] Fri
Levels: Sun < Mon < Tue < Wed < Thu < Fri < Sat
```

```
    yday(ddat)
```

```
[1] 2
```

```
    hour(ddat)
```

```
[1] 3
```

```
    minute(ddat)
```

```
[1] 51
```

```
    second(ddat)
```

```
[1] 44
```

- **Exercise**: Load the `wmata_ridership` data frame into R from https://dcgerard.github.io/stat_412_612/data/wmata_ridership.csv. For each month, calculate the proportion of rides made on a given day of the month. Then make box plots of the proportions of ridership vs day of the weak. But exclude any days from 2004.
- You can overwrite components.

```
ddat <- mdy_hms("01/02/1970 03:51:44")
ddat
```

```
[1] "1970-01-02 03:51:44 UTC"
```

```
year(ddat) <- 1988
ddat
```

```
[1] "1988-01-02 03:51:44 UTC"
```

- To create a new date with the updated component, rather than overwrite a component, use `update()`.

```
    ddat
```

```
[1] "1988-01-02 03:51:44 UTC"
```

```
    update(ddat, year = 1999)
```

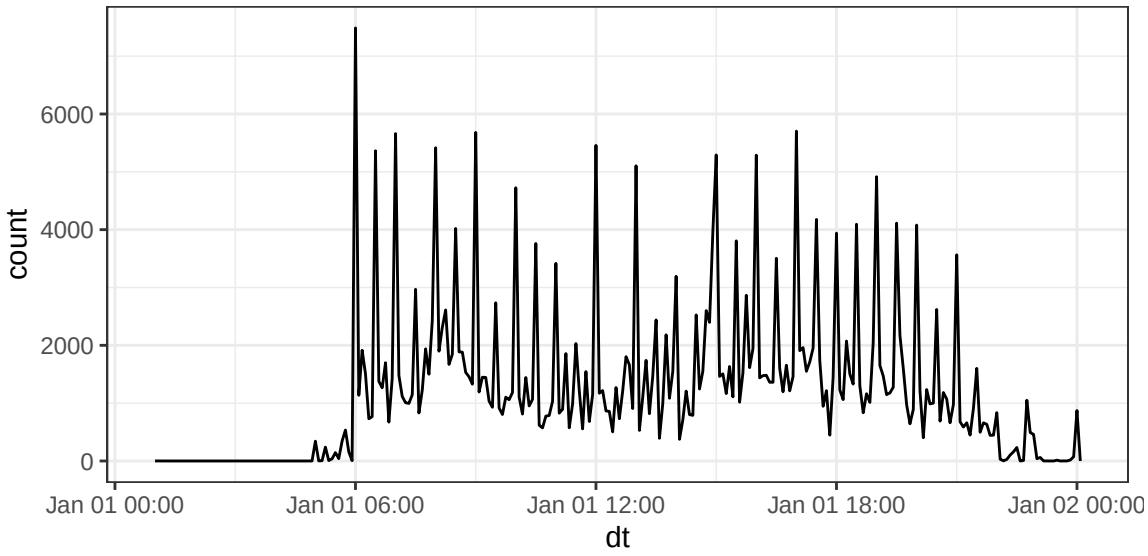
```
[1] "1999-01-02 03:51:44 UTC"
```

```
    ddat ## still 1988
```

```
[1] "1988-01-02 03:51:44 UTC"
```

- The book provides an example of using `update()` on larger elements to see fine scale patterns

```
flights %>%
  mutate(dt = update(datetime, yday = 1)) %>%
  ggplot(aes(x = dt)) +
  geom_freqpoly(binwidth = 300)
```



- You can round components

```
ddat <- mdy_hms("01/02/1970 03:51:44")
ddat
```

```
[1] "1970-01-02 03:51:44 UTC"
```

```
round_date(ddat, unit = "year")
```

```
[1] "1970-01-01 UTC"
```

Time Spans

- To count the number of seconds between two dates, use a `duration`. You can read about durations using

```
help("Duration-class")
```

- You first subtract two dates, then use `as.duration()` to create a duration.
- We can find out how old Patrick Stewart is using durations

```
d1 <- ymd(19400713)
d2 <- today()
agesec <- as.duration(d2 - d1)
agesec
```

```
[1] "2637792000s (~83.59 years)"
```

- You can also create durations from years with `dyears()`, from days with `ddays()`, etc…

```
dyears(1)
```

```
[1] "31557600s (~1 years)"
```

```
ddays(1)
```

```
[1] "86400s (~1 days)"
```

```
dhours(1)
```

```
[1] "3600s (~1 hours)"
```

```
dminutes(1)
```

```
[1] "60s (~1 minutes)"
```

```
dseconds(1)
```

```
[1] "1s"
```

- You can add durations to date-times, but you always add *seconds*, so if there is daylight savings you get weird results (add a day but the time is not the same as the time the previous day).

```
one_pm <- ymd_hms("2016-03-12 13:00:00", tz = "America/New_York")
one_pm
```

```
[1] "2016-03-12 13:00:00 EST"
```

```
one_pm + ddays(1)
```

```
[1] "2016-03-13 14:00:00 EDT"
```

- Adding a `period` takes into account daylight savings.

```
one_pm
```

```
[1] "2016-03-12 13:00:00 EST"
```

```
one_pm + days(1)
```

```
[1] "2016-03-13 13:00:00 EDT"
```

- You can read more about periods with

```
help("Period-class")
```

- Intervals are like durations, but they also have an associated start time. You can read more about intervals with

```
help("Interval-class")
```

- **Exercise**: How long of a time-span is covered in the WMATA ridership dataset?

Time Zones

- Time zones are specified using the `tz` or `tzone` arguments (for example, in the call to `ymd_hms()` above).
- Time zones are specified by "content/city." For example, `"America/New_York"` and `"Europe_Paris"`
- You can see a complete list of time zones with `OlsonNames()`.
- The default time zone is `UTC` (which has no daylight savings).
- You usually don't have to worry about timezones unless you loaded them in incorrectly. For example, R might think it's `UTC` even though it should be `America/New_York` and then forget daylight savings.
- If a date-time is labelled with the incorrect time zone, use `force_tz()`.

```
d1 <- ymd_hms("20140101 10:01:11")
d1
```

```
[1] "2014-01-01 10:01:11 UTC"
```

```
force_tz(d1, tzone = "America/New_York")
```

```
[1] "2014-01-01 10:01:11 EST"
```

- If the timezone is correct, but you want to change it, use `with_tz()`.

```
with_tz(d1, tzone = "America/New_York")
```

```
[1] "2014-01-01 05:01:11 EST"
```