Dates and Times

Tepi

9/22/2020

Objetivos

1. Familiarizar al estudiante con la librería de lubridate.
2. Aprender las operaciones básicas de manipulación entre fechas.

```
library(nycflights13)
```

```
## Warning: package 'nycflights13' was built under R version 4.0.2
```

```
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(lubridate)
```

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

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

Hoy y ahora

```
##el dia de hoy
today()
```

```
## [1] "2020-09-23"
```

```
## este momento
now()
```

```
## [1] "2020-09-23 11:23:55 CST"
```

Parseando fechas de texto.

Una de las grandes ventajas de esta libreria es que es muy intuitiva para parsear fechas, independientemente del formato en el que se encuentren. Lo único malo es que funciona únicamente con fechas en formato de

texto que estén en inglés.

```
# parsear x
x <- "1994 October 27th"
ymd(x)
```

```
## [1] "1994-10-27"
```

```
# parsear y
y <- "27.10.1994"
dmy(y)
```

```
## [1] "1994-10-27"
```

```
# parsear z
z <- "oct, 27th 1994 14:00"
mdy_hm(z)
```

```
## [1] "1994-10-27 14:00:00 UTC"
```

```
# parsear a
a <- 19942710
ydm(a)
```

```
## [1] "1994-10-27"
```

Diferencia de tiempos.

Más allá de agrupar por fechas al momento de hacer agregaciones utilizando `SQL` o `dplyr`, es necesario tomar en cuenta la temporalidad entre las distintas observaciones que podamos tener en nuestro set de datos. Por ejemplo, podemos estar interesados en cuánto tiempo transcurrió entre un evento A y un evento B como medida de desempeño o comportamiento.

Por ejemplo, ¿cuánto tiempo ha transcurrido desde que Neil Armstrong dio el primer paso en la luna?

```
# La fecha de aterrizaje y el primer paso en la luna
date_landing <- mdy("July 20, 1969")
moment_step <- mdy_hms("July 20, 1969, 02:56:15", tz = "UTC")

# Cuantos dias desde el primer hombre en la luna?
difftime(today(), date_landing, units = "days")
```

```
## Time difference of 18693 days
```

```
# Cuantos segundos desde el primer hombre en la luna?
difftime(now(), moment_step, units = "secs")
```

```
## Time difference of 1615133393 secs
```

Suma de tiempos

También es posible sumar intervalos de tiempo que comúnmente utilizamos en nuestro lenguaje. Por ejemplo, a la fecha de hoy podemos sumarle una semana, y obtener la fecha de esa semana.

Esto se puede utilizar para la generación de reportes automatizados y búsquedas históricas de información.

```
# Add a period of one week to mon_2pm
wed_1pm <- dmy_hm("23 Sep 2020 13:00")
wed_1pm + weeks(x=1)
```

```
## [1] "2020-09-30 13:00:00 UTC"
```

Consideraciones importantes de lubridate:

Lubridate maneja diferentes tipos de clases temporales.

- durations: miden la cantidad exacta de tiempo entre dos momentos, es decir, funciona como un cronómetro.
- periods: miden de forma precisa los tiempos del reloj sin considerar años bisiestos o day light savings. Son más una interpretación humana del tiempo.

El año 2020 fue un año bisiesto. Veamos la diferencia de sumar un año en las dos clases previamente mencionadas.

```
this_feb <- dmy("28 feb 2020")
this_feb + dyears(1)
```

```
## [1] "2021-02-27 06:00:00 UTC"
```

```
this_feb + years(1)
```

```
## [1] "2021-02-28"
```

Como podemos observar, dyears devuelve como resultado el 27 de febrero del 2021, debido a que es un cronómetro. Por su lado, years devuelve la misma fecha con un año más, es decir, como nosotros coloquialmente nos comunicamos al referirnos al tiempo.

Esto puede ocasionar problemas si no existen días en el calendario para las sumas que estamos realizando.

¿Qué deberia de devolver `ymd("2020-01-31") + months(1)`? ¿Debería de ser 30, 31, 28 o 29 días en el futuro?

En general, `lubridate` devuelve el mismo día del mes en el mes siguiente. Sin embargo, como el 31 de enero no existe, devuelve NA.

Existen otros operadores alternativos para suma y resta de fecha que se comportan de manera distinta. En lugar de devolver NA, hacen *rollback* a la última fecha existente:

```
this_jan <- dmy("31 jan 2020")
this_jan + months(1) ## devuelve NA
```

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

```
this_jan %m+% months(1) ##devuelve el ultimo dia del mes
```

```
## [1] "2020-02-29"
```

```
add_with_rollback(this_jan,months(1), roll_to_first  = FALSE)
```

```
## [1] "2020-02-29"
```

```
add_with_rollback(this_jan,months(1), roll_to_first  = TRUE)
```

```
## [1] "2020-03-01"
```

Generando secuencias de fechas

```
jan_31 <- ymd("2020-01-31")
# Secuencia de 1 a 12 periods de 1 mes
month_seq <- 1:12 * months(1)
# Add 1 to 12 months to jan_31
month_seq + jan_31
```

```
##  [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"
```

```
jan_31 <- ymd("2020-01-31")
oct_31 <- ymd("2020-10-31")
seq(jan_31,oct_31, "weeks")
```

```
##  [1] "2020-01-31" "2020-02-07" "2020-02-14" "2020-02-21" "2020-02-28"
##  [6] "2020-03-06" "2020-03-13" "2020-03-20" "2020-03-27" "2020-04-03"
## [11] "2020-04-10" "2020-04-17" "2020-04-24" "2020-05-01" "2020-05-08"
## [16] "2020-05-15" "2020-05-22" "2020-05-29" "2020-06-05" "2020-06-12"
## [21] "2020-06-19" "2020-06-26" "2020-07-03" "2020-07-10" "2020-07-17"
## [26] "2020-07-24" "2020-07-31" "2020-08-07" "2020-08-14" "2020-08-21"
## [31] "2020-08-28" "2020-09-04" "2020-09-11" "2020-09-18" "2020-09-25"
## [36] "2020-10-02" "2020-10-09" "2020-10-16" "2020-10-23" "2020-10-30"
```

flights

Es un dataset que contiene todos los vuelos realizados en Nueva York para el año 2013.

```
flights %>%
select (year, month,day, hour, minute, arr_time)
```

```
## Warning: `...` is not empty.
##
## We detected these problematic arguments:
## * `needs_dots`
##
## These dots only exist to allow future extensions and should be empty.
## Did you misspecify an argument?
```

```
## # A tibble: 336,776 x 6
##     year month   day  hour minute arr_time
##    <int> <int> <int> <dbl>  <dbl>    <int>
##  1  2013     1     1     5     15      830
##  2  2013     1     1     5     29      850
##  3  2013     1     1     5     40      923
##  4  2013     1     1     5     45     1004
##  5  2013     1     1     6      0      812
##  6  2013     1     1     5     58      740
##  7  2013     1     1     6      0      913
##  8  2013     1     1     6      0      709
##  9  2013     1     1     6      0      838
## 10  2013     1     1     6      0      753
## # ... with 336,766 more rows
```

make_date

Esta función es una forma muy fácil y eficiente de crear fechas en R. Tiene como parámetros el año, mes y día para generar una fecha en específico.

```
make_date(year = 1995, month = 11, day = 21)
```

```
## [1] "1995-11-21"
```

También la podemos utilizar para crear una nueva variable:

```
flights <- flights %>%
mutate(departure = make_date(year, month,day) )
```

`make_datetime()` hace lo mismo, pero incluye tambien la hora, minutos y segundos:

```
flights <- flights %>%
mutate(departure = make_datetime(year, month,day, hour, minute))
flights$departure %>% head()
```

```
## [1] "2013-01-01 05:15:00 UTC" "2013-01-01 05:29:00 UTC"
## [3] "2013-01-01 05:40:00 UTC" "2013-01-01 05:45:00 UTC"
## [5] "2013-01-01 06:00:00 UTC" "2013-01-01 05:58:00 UTC"
```