Trabajar con fechas y horas

Muchas veces la variable temporal es una variable de interés en nuestro trabajo. Trabajar con fechas y horas no es sencillo y presenta grandes molestias en todos los lenguajes de programación. Es por eso que en este curso le vamos a destinar unas cuantas páginas para abordar con tiempo y dedicación las distintas clases de objetos que existen en R para trabajar con fechas y las funciones más utilizadas.

R dispone en su paquete base con dos clases de objetos específicamente diseñadas para tratar con datos de tipo fecha/hora. La clase Date únicamente pensada para trabajar con fechas, y las del tipo POSIX que son las clases POSIXct y POSIXlt que almacenan información de fecha, hora y huso horario.

Excepto para la clase POSIXlt, las fechas se almacenan internamente como el número de días o segundos a partir de alguna fecha de referencia. Por lo tanto, las fechas en R tendrán generalmente un modo numérico, y la función de ***class()*** puede usarse para encontrar la forma en que están siendo almacenadas.

Las siguientes subsecciones describirán los diferentes tipos de valores de fecha con más detalle.

1 La clase Date

La clase más sencilla que encontramos en R para trabajar con fechas en la clase Date.

Esta clase almacena los valores de la fechas internamente como el número de días transcurridos desde el 1 de enero de 1970. Para las fechas anteriores, el número de días se cuenta en sentido negativo.

1.1 / Conversión: character >> clase Date

Para crear una fecha en R el modo más sencillo es utilizar la función ***as.Date()*** que convierte un dato de fecha en modo carácter y a la clase Date.

En esta clase, por *default* se utiliza el formato *"año-mes-día"* (yyyy-mm-dd).

Veamos un ejemplo donde introduciremos una fecha como character e indicaremos que se convierta a formato Date:

```
> navidad <-  as.Date("2018-12-25")
> navidad
[1] "2018-12-25"
```

Tras esto,

```
> class(navidad)
[1] "Date"
```

Si deseamos introducir la fecha en otro formato, es preciso indicarlo en la función ***as.Date()*** para que R pueda interpretar la fecha correctamente. Para ello utilizaremos el argumento ***format*** en el cual podemos especificar de qué forma ha sido introducida la fecha.

Prueba en tu consolas de R el resultado de las siguientes sentencias:

```
> as.Date("25/12/2013", format ="%d/%m/%Y" )
> as.Date("25-dic-13", format ="%d-%b-%y" )
> as.Date("25 Diciembre 2013", format = "%d %B %Y" )
```

El argumento **format** utiliza los distintos símbolos para codificar las fechas:

Símbolo	Significado
%d	día numérico, de 0 a 31
%a	día de la semana abreviado en tres letras
%A	día de la semana por nombre completo
%m	mes en numérico de 0 a 12
%b	mes abreviado a tres letras
%B	mes nombre completo
%y	año con dos dígitos. Ej: 98
%Y	año con cuatro dígitos. Ej: 1998

Como es de esperar, esta conversión también opera sobre vectores siempre que estén codificados del mismo modo:

```
> dias <- c("1/10/2005", "2/2/2006", "3/4/2006", "8/11/2014" )
> as.Date( dias, format = "%d/%m/%Y" )
[1] "2005-10-01" "2006-02-02" "2006-04-03" "2014-11-08"
```

Desafío: Reconocer fecha en otro idioma

Si bien no es algo muy frecuente, puede sucedernos que nuestro sistema no reconozca el español o que queramos trabajar en otro idioma.

En este desafío buscaremos que R reconozca una fecha en italiano. Para ello, llamaremos la librería **readr** y utilizaremos la función **parse_date()**

```
> library(readr)
> parse_date( "31 gennaio 2011",
              format = "%d %B %Y", locale = locale("it") )
```

¿Funcionó?

1.2 / Conversión: fecha >> character

Las funciones ***as.character()*** o ***format()*** pueden utilizarse indistintamente para convertir variables de clase Date en variables de clase character.

Por defecto, ambas funciones devuelven la fecha con el formato estándar de R (año-mes-dia):

```
> f1_nav <-  as.character(navidad)
> f2_nav <-  format(navidad)
> c( f1_nav, f2_nav)
[1] "2018-12-25"   "2018-12-25"
```

Si corroboramos las clases, efectivamente ambas funciones realizan la conversión:

```
> c( class(f1_nav), class(f2_nav) )
[1] "character"    "character"
```

Si se desea presentar la fecha como carácter en formatos distintos del estándar podemos indicarlo con el argumento **format**.

Prueba los siguientes formatos de fecha en tu consola pero ¡no te olvides de crear la variable navidad previo a ejecutar las sentencias!

```
> as.character( navidad, format = "%d %b. %Y")
> format(navidad, format = "%A, %d de %B de %Y")
```

Incluso podemos presentar sólo partes de la fecha, como pueden ser el día o el mes:

```
> as.character( navidad, format="%A")
> format( navidad, format = "%B")
```

Si introducimos una variable de clase *character* en las funciones ***as.character()*** *y* ***format()*** todo parece funcionar correctamente. Sin embargo, cuando queramos cambiar el formato en la función ***as.character()*** no funcionará la sentencia:

```
> as.character("2018-12-25", format="%d/%m/%Y") # No funciona
[1] "2018-12-25"
```

y en el caso de ***format()***, nos devolverá un mensaje de error:

```
> format("2018-12-25", "%d/%m/%Y")  #arroja error

Error in format.default("2018-12-25", "%d/%m/%Y"): argumento
'trim' inválido
format("2018-12-25","%A")
Error in format.default("2018-12-25", "%A"): argumento 'trim'
inválido
```

Por tanto, es preciso que estemos atentos y recordemos convertir la variable a la clase Date:

```
> format(as.Date("2018-12-25"),"%A")

[1] "miércoles"
```

1.3 / Fecha actual

La función **Sys.Date()** lee la fecha actual del reloj de nuestro ordenador y la codifica como un objeto de la clase Date.

¿Qué día es hoy? Introduce la siguiente sentencia en tu consola:

```
> Sys.Date()
```

Desafío: Jugando con las fechas

1. ¿Qué día de la semana era el 20 de julio de 1969? (llegada del hombre a la luna)
2. ¿Qué día de la semana llegó Colón a América? (12 de octubre de 1492)
3. ¿En qué día de la semana naciste?

1.4 / Codificación interna de objetos de clase Date

La función ***unclass()*** nos muestra la representación interna de un objeto en R.

Para un objeto de la clase Date, como hemos mencionado anteriormente, dicha representación consiste en el número de días transcurridos desde el 1/1/1970.

Prueba las siguientes líneas de código y observa su resultado:

```
> unclass(as.Date("1970-1-1"))
> unclass(as.Date("1970-1-10"))
> unclass(as.Date("1969-12-20"))
```

1.5 / Operaciones con variables de clase Date

Una ventaja de trabajar con la clase Date es que podemos realizar sumas y restas como si se tratasen de números.

1.6 / Diferencia entre fechas

Si se restan dos variables de clase Date, R nos devuelve la diferencia en días:

```
> dia1 <- as.Date("25/12/2017", format = "%d/%m/%Y")
> dia2 <- as.Date("20/1/2018", format = "%d/%m/%Y")
> dia2 - dia1
Time difference of 26 days
```

También, podemos calcular la diferencia en otras unidades, como semanas, horas con la función ***difftime()***:

```
> difftime(dia2, dia1, units = "weeks")
> difftime(dia2, dia1, units = "days")
> difftime(dia2, dia1, units = "hours")
> difftime(dia2, dia1, units = "mins")
> difftime(dia2, dia1, units = "secs")
```

El argumento ***unit*** admite cualquiera de las siguientes unidades: “secs”, “mins”, “hours”, “days”, “weeks”.

1.7 / Suma o resta de una fecha con un entero

Al sumar o restar un número entero a una fecha, la fecha se adelanta o retrasa en ese número de días:

```
> dia2 - 10
> dia2 + 42
```

1.8 / Diferencias entre fechas sucesivas en un vector

La función ***diff()*** calcula la diferencia en días, entre los términos sucesivos de un vector de fechas declaradas con la clase Date:

```
> dias <- as.Date( c("1/12/2015",
                     "24/8/2016","4/5/2018"), format = "%d/%m/%Y")

> diff(dias)

Time differences in days
[1] 267 618
```

1.9 / Secuencia de fechas

En el encuentro anterior vimos la función ***seq()*** que nos permitía generar secuencia de números. Esta función también puede aplicarse a objetos de clase Date.

De igual manera, es posible especificar la longitud de la secuencia que queremos realizar mediante el parámetro ***length.***

```
> seq (dia1, dia2, length = 10)
[1] "2017-12-25" "2017-12-27" "2017-12-30" "2018-01-02"
[5] "2018-01-05" "2018-01-08" "2018-01-11" "2018-01-14"
[9] "2018-01-17" "2018-01-20"
```

o el tiempo de separación entre las fechas de la secuencia con el parámetro ***by***:

```
> seq (dia1, dia2, by  =  "week")
[1] "2017-12-25" "2018-01-01" "2018-01-08" "2018-01-15"
```

Probar las otras expresiones para el argumento ***by***:

```
> seq( dia1, dia2, by = 15)
> seq( dia1, dia2, by = "month")
> seq( dia1, dia2, by = "2 weeks")
> seq( dia1, dia2, by = "3 months")
```

2 Posixt

La clase Date nos permitía trabajar con fechas, ahora veremos la clase Posix que **nos permite trabajar con fechas y horas**.

El término POSIX engloba a una colección de estándares de acceso a funciones del sistema operativo. Es un acrónimo de Portable Operating System Interface. La X procede del sistema operativo UNIX, si bien estos estándares son también válidos para otros sistemas.

Esta última clase contiene dos subclases, **POSIXct y POSIXlt** que se diferencian simplemente en la forma en que almacenan internamente la fecha y la hora. En ambos casos, se utiliza como fecha y hora de referencia el número de segundos transcurridos desde el 1 de enero de 1970. La clase **POSIXct almacena internamente esta cifra como un número entero, mientras que la clase POSIXlt la descompone en una lista con elementos para los segundos, minutos, horas, día, mes y año**. El sufijo ***ct*** son las iniciales de ***"calendar time"*** y ***lt*** de **"local time".**

Como esta nueva clase permite trabajar con fecha y hora, a continuación se presentan los distintos códigos de formato que podremos necesitar:

Code	Meaning	Code	Meaning
%a	Día de la semana abreviado a tres letras	%A	Día de la semana ,nombre completo
%b	Mes, abreviado a tres letras	%B	Mes, nombre completo
%c	Fecha y Hora local	%d	Día , numérico de 0 a 31
%H	Hora, numérico de 00 a 23	%I	Hora, numérico de 00 a 12
%j	Día del año, número 001 a 365	%m	Mes del año, número de 0 a 12
%M	Minuto, numérico de 00 a 59	%p	Indica AM/PM , configuración regional
%S	Segundos, numérico de 00 a 59	%U	Semana del año, numérico 00 a 53 (1º domingo como día 1 de la semana 1)
%w	Día de la semana (0= Domingo)	%W	Semana del año, numérico 00 a 53 (1º lunes como día 1 de la semana 1
%x	Fecha específica loca	%X	Hora específica local
%y	Año, con dos dígitos	%Y	Año, con cuatro dígitos
%z	Desplazamiento desde GMT	%Z	Zona horaria como cadena de caracteres

Clase POSIXct

Como dijimos anteriormente, esta clase permite manejar la hora además de la fecha. El almacenamiento interno de una fecha en formato POSIXct es el número de segundos transcurridos desde el 1 de enero de 1970, y para fechas anteriores se usan números negativos.

2.1 / Conversión: character -> clase POSIXct

Para esta conversión se utiliza la función ***as.POSIXct()*** y podemos especificar una fecha o una fecha y una hora:

```
> fh1 <- as.POSIXct( "01/10/1983", format = "%d/%m/%Y" )
> class(fh1)
[1] "POSIXct" "POSIXt"
```

Nótese que en la conversión, R incluye también la zona temporal que figura en la configuración de nuestro ordenador.
Las zonas temporales son los nombres que reciben los husos horarios (ver Huso Horario, Lista de zonas horarias o Database Tz).

R muestra los nombres de todos los husos horarios mediante la función ***OlsonNames()***. En concreto, Buenos Aires se encuentra en la zona UTC -3 (*Coordinated Universal Time*). Si queremos especificar una fecha y hora correspondiente a uso huso horario podemos especificarlo mediante el parámetro **tz**.

Por ejemplo, si la hora que queremos introducir corresponde a la ciudad de Buenos Aires usamos:

```
> fh1 <- as.POSIXct("01/10/1983 22:10:00",
             format = "%d/%m/%Y %H:%M:%S",
             tz = "America/Argentina/Buenos_Aires" )
[1] "1983-10-01 22:10:00 -03"
```

2.2 / Operaciones con la clase POSIXct

Dado que esta clase almacena la fecha/hora en segundos, las operaciones con objetos de esta clase se realizan en segundos:

```
> fh1 + 200
[1] "1983-10-01 22:13:20 -03"
```

Al igual que como veíamos con la clase Date, si se utiliza ***difftime()*** se pueden especificar las unidades:

```
> fh1 <- as.POSIXct("01/10/1983 22:10:00",
             format = "%d/%m/%Y %H:%M:%S",
             tz = "America/Argentina/Buenos_Aires" )
> fh2 <- as.POSIXct("04/12/1985 10:30:00",
             format = "%d/%m/%Y %H:%M:%S",
             tz = "America/Argentina/Buenos_Aires" )
> difftime(fh2, fh1, units="weeks")
1] Time difference of 113.502 weeks
```

2.3 / Clase POSIXlt

La clase POSIXlt tiene como particularidad que almacena los valores de fecha y hora como una

lista de componentes que facilitan la extracción de estas partes. Para crear una fecha en esta clase podemos utilizar la función ***as.Posixlt()***:

```
> fh3 <- as.POSIXlt("2018-03-08 16:31:04")
> class(fh3)
[1] "POSIXlt" "POSIXt"
```

o la función ***strptime()*** especificando el formato con el argumento ***format***:

```
> fh3 <- strptime("2018-03-08 16:31:04",
                  format = "%Y-%m-%d %H:%M:%S")
> class(fh3)
[1] "POSIXlt" "POSIXt"
```

La lista de componentes contiene el año ($year), mes ($mon), día ($mday), hora ($hour), minutos ($min) segundos ($sec), el día de la semana ($wday), el día del año ($yday), el parámetro isdst (que vale 0 en horario de verano y 1 en horario de invierno), el huso horario ($zone) y el parámetro gmtoff que cuenta el número de segundos de desfase respecto a GMT, Greenwich Meridian Time u hora del meridiano de Greenwich).

Utilizando la función **unclass()** podemos ver la forma de almacenamiento en esta clase:

```
> unclass(fh3)
$sec
[1] 4
$min
[1] 31
$hour
[1] 16
$mday
[1] 8
$mon
[1] 2
$year
[1] 118
$wday
[1] 4
$yday
[1] 66
$isdst
[1] 0
$zone
[1] "GMT"
$gmtoff
[1] NA
```

Esto nos permite acceder fácilmente a los componentes de esta fecha, mediante incorporar el símbolo $:

```
> fh3$sec
[1] 4
```

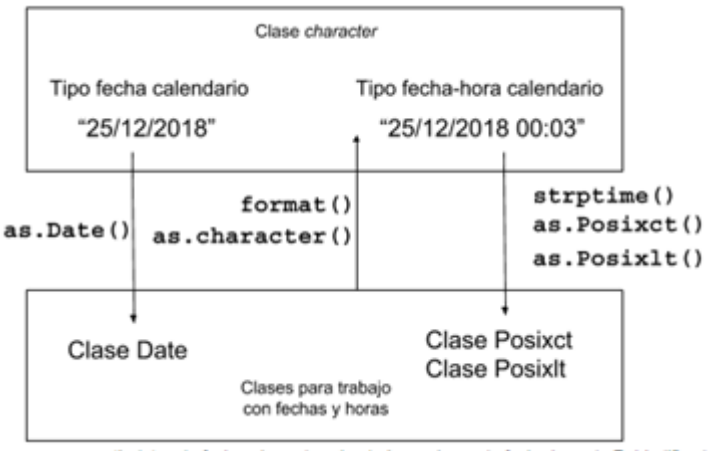
Aunque el almacenamiento interno sea en forma de lista, al igual que el caso de la clase POSIXct las operaciones con objetos de la clase POSIXlt se realizan en segundos:

```
> fh3 + 20
[1] "2018-03-08 16:31:24 GMT"
```

Resumiendo

Es preferible que las fechas las trabajemos en una clase que nos facilite su manipulación. Si son fechas tenemos la clase Date, y si son fechas con horas las clases Posixct o Posixlt. Dependiendo de cuál sea el trabajo que queremos realizar conviene utilizar Posixct o Posixlt, pero esto no es un problema puesto que podemos convertir las clases sin dificultad.

A continuación un breve ayuda memoria de las funciones disponibles para la conversión de formatos:



Funciones para convertir datos de fecha y hora de calendario en clases de fecha-hora de R. Modificado de: Applied Epidemiology Using R (An Open Access Book)

3 El paquete lubridate

El paquete **lubridate** dispone de diversas funciones que facilitan la extracción de componentes de un objeto fecha y hora de clase POSIXct.

Antes que nada recordemos cómo instalar y llamar un paquete:

```
> install.packages("lubridate")
> library(lubridate)
```

Lubridate añade funciones que facilitan la manipulación de fechas expresadas en la clase POSIXct.
Un ejemplo de esto es la facilidad que presenta para convertir cadenas de caracteres a la clase Posixct. La función ***dmy_hms()*** espera que incorporemos la fecha en el orden día-mes-año-horas-minutos-segundos sin importar los separadores que usemos.

```
> fh4 <- ymd_hms("2013-03-10 08:32:07")
> class(fh4)
[1] "POSIXct" "POSIXt"
```

y si usamos ***ymd_h()*** espera la fecha en el orden año-mes-día-hora.

```
> fh5 <- dmy_hm("1/10/1963 08:32")
> class(fh5)
[1] "POSIXct" "POSIXt"
```

Este paquete también trae las funciones ***month()***, ***week()***, ***year()***, ***wday()***, ***hour()*** y ***tz()*** para acceder a la información de la fecha guardada:

```
> week(fh4)
[1] 10
```

Otras funciones útiles para la manipulación de fechas en **lubridate** son las funciones ***round_date()*** que redondea la fecha en la unidad que especifiquemos:

```
> round_date(fh4, unit = "month")

[1] "2013-03-01 UTC"
```

La ventaja de redondear una fecha frente a extraer sus elementos es que mantiene el contexto de la unidad. Ej: al extraer la hora de un objeto fecha-tiempo perderemos el día en que ocurrió, sin embargo, si redondeamos a la hora más cercana, se mantiene la fecha y la hora.

Mientras que **round_date()** redondea a la unidad o múltiplo de unidad más cercana, **ceiling_date()** redondea por exceso en la unidad especificada y **floor_date()** redondea por defecto en la unidad especificada.

```
> ceiling_date(fh4, unit = "month")
[1] "2013-04-01 UTC"
```

El argumento ***unit*** admite cualquiera de las siguientes unidades: “second”, “minute”, “hour”, “day”, “week”, “month”, “bimonth”, “quarter”, “halfyear” o “year”.

```
> round_date(fh4, unit = "month")
[1] "2013-03-01 UTC"
```

También podemos encontrar la función ***now()*** *y* ***today()*** que nos que nos brindan, respectivamente, la hora y el día actual.
Para conocer más sobre esta librería podés consultar el manual oficial.