Uso de módulos o paquetes

Los módulos o paquetes permiten agrupar un conjunto de funciones o elementos de código, de modo que, para poderlos utilizar, es necesario incorporar dicho paquete a nuestro programa. Gran parte de los paquetes de terceros disponibles para el lenguaje R se almacenan en un repositorio online denominado CRAN (*Comprehensive R Archive Network*), donde también se almacena la distribución del propio lenguaje R. Podéis consultar su web oficial.

1. Instalación y uso de paquetes

Los siguientes comandos permiten gestionar la instalación de paquetes:

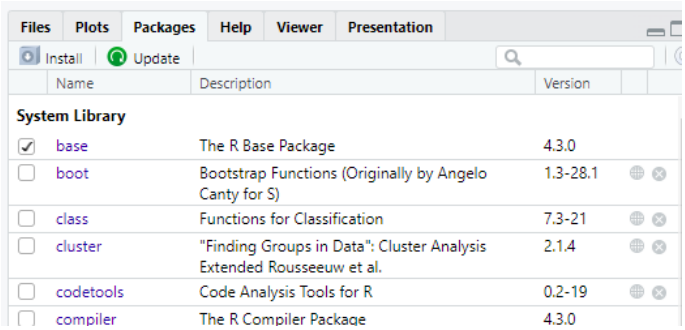
- `install.packages` instala el paquete o paquetes que le indiquemos en el sistema
- `installed.packages` muestra los paquetes que tenemos actualmente instalados
- `old.packages` muestra un listado de paquetes desfasados que necesitan actualización
- `remove.packages` elimina los paquetes que indiquemos del sistema
- `update.packages` actualiza los paquetes que lo necesiten. Para cada uno nos preguntará si lo queremos actualizar o no

Aquí vemos algunos ejemplos con los paquetes *ggplot2* y *tibble* que se utilizan para tareas de *data science*.

```
install.packages("ggplot2")              # Instala los paquetes en el sistema
install.packages("tibble")
install.packages(c("ggplot2", "tibble")) # Otra forma de hacer lo mismo en una línea
installed.packages()                     # Muestra los paquetes instalados
remove.packages("ggplot2")               # Elimina el paquete del sistema
```

Es **IMPORTANTE** recalcar que estas instrucciones no deberían incluirse en el código fuente de un programa, ya que sólo necesitan ejecutarse una vez. En su lugar, deben ejecutarse desde una consola R. La herramienta *R Studio* incorpora una consola para hacer esto directamente, pero si no disponemos de esta herramienta, podemos ir al terminal de nuestro sistema operativo y ejecutar el comando `r` para abrir la consola de R (luego ejecutaremos `quit()` para cerrar la consola).

Además, en R Studio tenemos una pestaña *Packages* para comprobar los paquetes que tengamos instalados en un momento determinado. Desde esa misma sección también podemos instalar/borrar/actualizar los paquetes.



1.1. Carga de paquetes

Para los paquetes que ya tengamos instalados, deberemos incorporarlos a nuestro código para poderlos usar. Esto se realiza a través de la instrucción `library`, indicando el nombre del paquete sin comillas:

```
library(tibble)
library(ggplot2)
```

2. Ejemplo: *lubridate*

El paquete `lubridate` permite gestionar de forma sencilla fechas y horas. Aquí podemos consultar su web oficial. Se instala en el sistema con el correspondiente comando:

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

> **NOTA:** este paquete también forma parte del ecosistema tidyverse, que engloba varios paquetes útiles para *data science* con lo que, si hemos instalado ya el paquete *tidyverse*, tendremos incorporado *lubridate*

Una vez instalado el paquete, podemos incorporarlo en nuestros programas:

```
library(lubridate)
```

2.1. Obtener y mostrar fechas y horas

Algunas de las funciones que incorpora *lubridate* para gestionar fechas y horas son las siguientes:

- La función `today` obtiene la fecha actual, y la función `now` la fecha y hora actuales:

```
hoy <- today()
ahora <- now()
```

- Las funciones `ymd`, `mdy` y `dmy` convierten en fecha un texto que esté en el formato especificado (año-mes-día en el caso de *ymd*). Podemos comprobar que es un sistema bastante flexible, y también admite variantes con otras funciones que añaden las horas:

```
fecha1 <- ymd("2023-01-24")          # 24 de enero de 2023
fecha2 <- mdy("01-24-2023")          # 24 de enero de 2023
fecha3 <- mdy("January 24st, 2023") # 24 de enero de 2023
fecha4 <- dmy("24/01/2023")          # 24 de enero de 2023
fecha5 <- dmy_hms ("24/01/2023 22:11:55") # Misma fecha a las 22:11:55
```

Para mostrar una fecha/hora por pantalla podemos emplear la instrucción `format` para darle formato de salida. Por ejemplo:

```
cat(format(fecha1, "%d/%m/%Y"))
```

2.2. Acceso a las partes de la fecha y hora

Una vez hemos construido una fecha/hora, las instrucciones `day`, `month`, `year`, y también `hour`, `minute` y `second`:

```
day(fecha1)     # 24
minute(fecha5)  # 11
```

Si queremos modificar alguno de estos componentes podemos emplear la instrucción `update`, a la que le pasamos la fecha a modificar y cada uno de los componentes a modificar:

```
fecha1bis <- update(fecha1, day=30, month=12)
```

2.3. Operaciones con fechas y horas

Podemos también realizar algunas operaciones básicas con fechas y horas, como por ejemplo sumar/restar días/horas/meses/etc a una fecha, para obtener otra, usando las funciones `days`, `hours`, `months`, `years` ...

```
fecha6 <- fecha1 + days(10)  # 3 de febrero de 2023
fecha7 <- fecha5 + hours(2)  # 25 de enero de 2023 a las 00:11:55
```

También podemos obtener la diferencia entre dos fechas, y expresar el resultado en distintas unidades (días, horas...). Es lo que se conoce como **intervalos** (*intervals*), y permiten hacer otras operaciones más avanzadas, como por ejemplo ver si dos intervalos se solapan.

```
fecha1 <- dmy("24/01/2023")
fecha2 <- dmy("25/02/2023")
fecha3 <- dmy("01/03/2023")
fecha4 <- dmy("05/04/2023")

intervalo1 <- interval(fecha1, fecha2)
intervalo2 <- interval(fecha3, fecha4)
intervalo3 <- interval(fecha2, fecha4)

int_overlaps(intervalo1, intervalo2)  # FALSE
int_overlaps(intervalo2, intervalo3)  # TRUE
intervalo1 / days(1)                  # 32 días
intervalo1 / hours(1)                 # 768 horas
```

Ejercicio 1:

Escribe un programa llamado `vacaciones.r` que le pida al usuario la fecha en que quiere salir de vacaciones y la fecha en la que quiere volver y, con esos datos, le muestre:

- Cuántos días va a estar de vacaciones
- Si su período de vacaciones se solapa con la temporada alta del año actual, que serán los días de Navidad (finales de diciembre y/o primeros de enero) y verano (julio y agosto).

AYUDA: vídeo con solución del ejercicio