Análisis de datos en R

En este documento veremos cómo podemos emplear algunas funcionalidades nativas de R (y alguna externa) para realizar un pre-procesado y análisis de los datos con los que vayamos a trabajar.

1. *Data frames*

Un *data frame* es un conjunto de columnas agrupadas, representando así una tabla de datos. Cada columna puede tener un *nombre* asociado, y ser de un tipo de dato concreto. El concepto es similar al concepto de *data frame* de la librería Pandas de Python. Pero, a diferencia de Python, en R no necesitamos instalar ninguna librería externa para poder utilizarlos.

1.1. Crear *data frames*

Podemos crear *data frames* en R de distintas formas. Una forma sencilla es emplear la instrucción `data.frame` e indicar el nombre de cada columna junto con su secuencia de valores (todas las columnas deben tener el mismo número de valores):

```
datos <- data.frame(nombre=c("Juan", "Pepe"),
    edad=c(70, 60), socio=c(TRUE, FALSE))
```

Esto generará un contenido como el siguiente:

	nombre	edad	socio
1	Juan	70	TRUE
2	Pepe	60	FALSE

Habitualmente lo que haremos será cargar un archivo con datos, normalmente en formato CSV. Para ello utilizaremos la función `read.csv`:

```
datos <- read.csv("fichero.csv", sep=";")
```

Por defecto la función asume que el separador de datos en el fichero es la coma, y asume que la primera fila contiene los nombres de columnas. Para elegir otro separador diferente podemos emplear el parámetro `sep`. Además, podemos emplear otras alternativas. Por ejemplo, es habitual que en algunos archivos CSV los números decimales se separen con una coma en lugar de un punto. En este caso podemos indicar con el

parámetro `dec` cuál es el separador decimal, o bien podemos usar alternativamente la instrucción `read.csv2`, que ya asume como separador decimal la coma:

```
datos <- read.csv("fichero.csv", sep=";", dec=",")
# Alternativa
datos <- read.csv2("fichero.csv")
```

1.2. Gestión de filas y columnas

Podemos acceder a una columna en concreto poniendo su nombre tras el *data frame*, enlazado con un `$`, igual que se hace con las listas:

```
datos$edad # Vector con las edades
```

Podemos seleccionar un rango de columnas indicando sus índices, como rango o como índices independientes en un vector. También podemos indicar los nombres de columna:

```
datos[1:2]                # nombre y edad
datos[c(1, 3)]            # nombre y socio
datos[c('edad', 'socio')] # edad y socio
```

Para acceder a una fila, indicamos su número, separado por una coma de la columna (o columnas) que nos interesen:

```
datos[1, c(1, 2)]  # nombre y edad de la 1ª fila
datos[2,]          # todos los datos de la 2ª fila
datos[1:3,c(1, 3)] # nombre y socio de 3 primeras filas
```

1.3. Algunas operaciones importantes

Veremos a continuación algunas operaciones habituales e importantes que suelen hacerse con los *data frames*.

- Las funciones `head` y `tail` permiten ver las primeras/últimas filas del *data frame*, respectivamente. Podemos indicar un parámetro `n` con la cantidad de filas que queremos mostrar. La función `summary` muestra un resumen del *data frame*, indicando valores máximos, mínimos, medias, etc de cada columna.

```
head(datos, n=3)
summary(datos)
```

- La función `dim` indica las dimensiones del *data frame*, en forma de vector con el número de filas y columnas. Alternativamente, `nrow` y `ncol` indican el número de filas y columnas por separado.

```
dimensiones <- dim(datos)
filas <- nrow(datos)
```

- Las funciones `rownames` y `colnames` permiten consultar/establecer los nombres de las filas y columnas, respectivamente, igual que ocurre con las matrices.

```
v1 <- 1:3
v2 <- 4:6
matriz <- cbind(v1, v2)
rownames(matriz) <- c("a", "b", "c")

# Tendremos un resultado como este:
#    v1  v2
# a   1   4
# b   2   5
# c   3   6

matriz['a', 'v2'] # 4
```

- La función `sapply` aplica una función a todos los elementos (columnas) del *data frame*

```
# Obtenemos el tipo de dato de cada columna
resultado <- sapply(datos, class)
```

- Podemos calcular la media de un conjunto de valores con la función `mean`. Podemos indicar un parámetro `na.rm` que, si está a *TRUE*, descartará de la colección los valores nulos para calcular la media. Ocurre algo similar con las funciones `sum`, `min` o `max`. Aquí vemos unos ejemplos:

```
v <- c(1, 2, NA, 3)
mean(v)             # NA
mean(v, na.rm=TRUE) # 2
sum(v, na.rm=TRUE)  # 6
max(v, na.rm=TRUE)  # 3
```

2. Los factores (*factors*)

Un *factor* permite categorizar datos en niveles. Se puede utilizar para definir las diferentes categorías dentro de un conjunto de datos, de forma que los valores que toman esos datos pertenecen a una de esas categorías. Imaginemos, por ejemplo, que tenemos un vector con calificaciones nominales de alumnos:

```
notas <- c("Notable", "Sobresaliente", "Notable",
    "Aprobado", "Suspenso", "Aprobado")
```

Si hacemos pasar este vector por la función `factor` obtendremos un vector *factorizado*, del que podremos obtener los niveles *levels* que contiene:

```
notas_factor <- factor(notas)
print(notas_factor)
# Mostrará las notas almacenadas y los niveles detectados:
# Levels: Aprobado Notable Sobresaliente Suspenso
```

La instrucción `levels` permite obtener las diferentes etiquetas o categorías de un factor:

```
categorias <- levels(notas_factor)
```

Además, podemos codificar numéricamente los valores de una columna categórica, a través de la función `factor`. Le tenemos que indicar los datos a convertir, los niveles (*levels*) que tiene y cómo queremos codificar cada nivel. Por ejemplo:

```
# Convertimos las notas en valores de 0 a 3
notas_codificadas <- factor(notas,
    levels=c("Suspenso", "Aprobado", "Notable", "Sobresaliente"),
    labels=c(0, 1, 2, 3))
# Resultado: 2 3 2 1 0 1
```

3.1. Factores en *data frames*

Los factores pueden resultar particularmente útiles en los *data frames* para categorizar columnas. Por ejemplo, dado el siguiente *data frame*:

```
datos <- data.frame(
    nombre=c("Juan", "Pepe", "Ana", "Ricardo"),
    estado=c("Casado", "Soltero", "Casado", "Viudo"),
    edad=c(70, 60, 40, 50)
)
```

Podemos factorizar las columnas que nos interesen

```
datos$estado <- factor(datos$estado)
print(datos$estado)
# Levels: Casado Soltero Viudo
```

También podemos factorizar indicando las categorías (*levels*) que queremos incluir, aunque algunos no estén presentes en la colección original:

```
datos$estado <- factor(datos$estado,
    levels=c("Casado", "Soltero", "Viudo", "Divorciado"))
```

La instrucción `table` nos permite contar cuántas ocurrencias hay de cada categoría en un conjunto de datos:

```
table(datos$estado)
# Casado  Soltero  Viudo  Divorciado
#      2        1      1           0
```

La librería forcats dispone de funciones para ordenar datos por frecuencia, recategorizar, etc.

3. *Tidyverse*

Tidyverse es un conjunto de librerías vinculadas en mayor menor medida con el procesamiento y análisis de datos. Dispone de módulos para pre-procesamiento de datos, manipulación de cadenas de texto, fechas... así como carga y análisis de datos. Aquí podemos acceder a su web oficial.

3.1. Instalación

Podemos instalar *tidyverse* en el sistema con la instrucción correspondiente:

```
install.packages("tidyverse")
```

Alternativamente, también podemos instalar sólo los módulos que vayamos a utilizar. Algunos de los más relevantes son:

- `dplyr` : proporciona funciones para manipulación de datos en general
- `ggplot2` : para representaciones gráficas de los datos
- `tidyr` : ofrece mecanismos para ordenar y arreglar los datos
- `stringr` : dispone de funciones para manejo de cadenas de texto
- `tibble` : ofrece la posibilidad de trabajar con *tibbles*, una especide *data frames* con operaciones algo más cómodas y avanzadas
- `DBI` : para manejo de diferentes sistemas gestores de bases de datos, tales como SQLite, MariaDB/MySQL o Postgres.
- `magrittr` : para uso de tuberías o *pipes* para enlazar procesamientos de datos
- `readr` : para leer/escribir información desde/hasta diferentes formatos de archivo (CSV, hojas de cálculo...)
- `rvest` : para obtener información de páginas web
- Puedes consultar otros muchos paquetes aquí.

Veremos a continuación cómo utilizar algunos de estos paquetes.

3.2. El paquete *magrittr* y el operador *pipe*

El paquete `magrittr` ofrece ciertos operadores útiles en el tratamiento de datos. Podemos instalarlo a través del paquete general *tidyverse* o instalando el propio paquete en concreto *magrittr*. Después lo incorporamos a nuestro código:

```
library(magrittr)
```

Uno de los operadores más habituales de este paquete es el operador de tubería o *pipe*, `%>%` . Permite proporcionar argumentos a funciones; por ejemplo, en lugar de calcular la media de un conjunto de valores de esta forma:

```
mean(1:10)  # 5.5
```

podemos hacerlo de esta otra forma:

```
1:10 %>% mean()
```

Aparentemente no hemos ganado mucho; la instrucción resultante es más larga que la original. Sin embargo, este operador tiene su utilidad cuando queremos enlazar varias de estas operaciones. Por ejemplo, podemos redondear la media de los números calculados:

```
1:10 %>%
mean() %>%
floor()
```

Es **IMPORTANTE** recalcar que el operador *pipe* sólo rellena el primer argumento de la función. El resto de argumentos deben proporcionarse en el orden indicado en cada caso.

```
"Hola" %>%
paste("mundo")  # Hola mundo
```

En términos más avanzados, nos resultará extremadamente útil para enlazar operaciones con *data frames*:

```
datos <-
  read.csv('fichero.csv') %>%
  subset(variable_a > x) %>%
  transform(variable_c = variable_a/variable_b) %>%
  head(100)
```

3.3. Los *tibbles*

Un *tibble* es una estructura similar al *data frame* de R, que proporciona unas características más avanzadas o cómodas para el análisis de datos. Una vez hemos instalado la librería `tidyverse` (o, de forma particular, el paquete `tibble`) incorporamos este módulo a nuestro programa:

```
library(tibble)
```

Los *data frames* de R se pueden utilizar como *tibbles* y viceversa. Para convertir un *data frame* en un *tibble* usamos la función `as_tibble`:

```
datos <- data.frame(nombre=c("Juan", "Pepe"),
    edad=c(70, 60), socio=c(TRUE, FALSE))

t1 <- as_tibble(datos)
```

> **NOTA:** observad cómo las funciones en *tidyverse* separan sus palabras con subrayados `_`, mientras que en R es habitual separar las palabras con puntos (`as.character`, por ejemplo).

Alternativamente, podemos crear un *tibble* desde cero con la función `tibble`. Ésta tiene algunas particularidades, como por ejemplo el hecho de rellenar una columna con un valor fijo, o poder usar valores de columnas previas en las que estamos creando:

```
t2 <- tibble(
    x = 1:5,     # 5 filas del 1 al 5
    y = 1,       # 5 filas rellenas con 1
    z = x + y    # 5 filas con la suma x + y
)
```

Podemos mostrar por pantalla los datos de un *tibble* con la instrucción `print`. En el parámetro `n` indicamos el número de líneas (filas) que queremos mostrar y en el parámetro `width` la anchura (columnas) que queremos que ocupe la información mostrada:

```
# Muestra las 5 primeras filas ocupando 40 columnas
print(t2, n=5, width=40)
```

3.4. Gestión de cadenas de texto con *stringr*

La librería `stringr` dispone de funciones adicionales a las que ya proporciona R para manipulación de cadenas de texto. Muchas de ellas aceptan como entrada un vector de cadenas de texto, para procesarlas todas en conjunto. Veamos algunos ejemplos:

```
library(stringr)
textos <- c("Uno", "Dos", "Tres")

# Longitud de cada texto
str_length(textos)  # 3  3  4
str_length("Hola")  # 4

# Subcadenas
str_sub("Hola", 1, 2)  # Ho
str_sub(textos, 1, 2)  # Un  Do  Tr

# Detectar subcadena en texto
str_detect("Hola", "a") # TRUE
str_detect(textos, "o") # TRUE  TRUE  FALSE

# Reemplazar texto
str_replace_all("Hola", "a", "A") # HolA
str_replace_all(textos, "o", "O") # UnO  DOs  Tres
```

Puedes encontrar muchas más funciones y ejemplos en la documentación oficial.

3.5. Procesamiento de datos con *dplyr*

Como hemos comentado antes, `dplyr` es uno de los paquetes más relevantes de *tidyverse*, destinado a la manipulación de conjuntos de datos. Sería equivalente a la librería Pandas para Python, y dispone de operaciones para mecanismos habituales como el filtrado, ordenación, reemplazos o cambios, etc. Deberemos instalarlo como paquete (si no hemos instalado *tidyverse* en su conjunto) y luego incorporarlo a nuestros programas:

```
library(dplyr)
```

Veremos a continuación algunas operaciones habituales con este paquete. Supondremos que tenemos un *data frame* en un fichero *personas.csv* con esta información:

	Nombre	Edad	Email	Telefono	...
1	Juan Pérez	70	jperez@gmail.com	611223344	...
2	Ana Sánchez	41	asg@hotmail.com	655443322	...
3	Pedro Carrillo	56	p.carrillo@gmail.com	677889900	...
...	...	...	...	...	...

3.5.1. Filtrados

La operación de filtrado nos va a permitir quedarnos con las filas de un *data frame* que cumplan un determinado criterio, mediante la función `filter`. Por ejemplo, si queremos filtrar las personas mayores de edad del ejemplo anterior podríamos hacer algo así:

```
personas <- read.csv("personas.csv")
adultos <- filter(personas, Edad >= 18)
```

Alternativamente podemos emplear el operador *pipe*:

```
adultos <-
   read.csv("personas.csv") %>%
   filter(Edad >= 18)
```

3.5.2. Selección de columnas

La operación `select` permite seleccionar las columnas que consideremos relevantes de un *data frame*. Podemos elegir un rango determinado...

```
# Columnas 1 a 3
seleccion <-
   read.csv("personas.csv") %>%
   select(1:3)

# Otra forma de hacer lo mismo
seleccion <-
   read.csv("personas.csv") %>%
   select(Nombre:Email)
```

... o también podemos especificar los nombres o números de columna por separado:

```
# Columnas Nombre y Email
seleccion <-
   read.csv("personas.csv") %>%
   select(c(1,3))

# Otra forma de hacer lo mismo
seleccion <-
   read.csv("personas.csv") %>%
   select(Nombre, Email)
```

También podemos hacer otras selecciones alternativas como, por ejemplo, todas las columnas menos las que indiquemos:

```
# Todo menos Nombre y Email
seleccion <-
   read.csv("personas.csv") %>%
   select(!c(Nombre,Email))
```

3.5.3. Modificar columnas

La operación `mutate` nos permitirá añadir o modificar columnas en un *data frame*. Por ejemplo, podemos añadir una columna más a la tabla anterior para indicar si las personas son socios de un determinado club, inicializando todos los valores a *FALSE*:

```
datos <-
   read.csv("personas.csv") %>%
   mutate(socio = FALSE)
```

También podemos modificar el contenido de una columna, o crear otra como combinación de varias. Por ejemplo, le podemos sumar uno a la edad de cada persona:

```
datos <-
   read.csv("personas.csv") %>%
   mutate(edad = edad + 1)
```

Esta operación puede resultar también útil para factorizar (convertir en factores) algunas columnas:

```
datos <-
   read.csv("personas.csv") %>%
   mutate(EstadoCivil = as.factor(EstadoCivil))
```

3.5.4. Ordenación

La operación `arrange` permite ordenar las filas de un *data frame* de acuerdo a un criterio determinado. Por ejemplo, podemos ordenar las personas por edad de menor a mayor:

```
datos <-
   read.csv("personas.csv") %>%
   arrange(Edad)
```

o de mayor a menor:

```
datos <-
   read.csv("personas.csv") %>%
   arrange(desc(Edad))
```

También podemos especificar varios criterios, separados por comas, para que se ordenen por uno y, en caso de igualdad, por el siguiente. Así ordenamos a las personas por edad y nombre:

```
datos <-
   read.csv("personas.csv") %>%
   arrange(Edad, Nombre)
```

3.5.5. Otras operaciones

Existen otras operaciones disponibles desde *dplyr* para manipulación de datos. Por ejemplo, la instrucción `group_by` permite agrupar filas por un determinado criterio, y la instrucción `summarize` permite calcular algún dato resumen. Este código muestra la media de edad agrupados por estado civil, y contando cuántos individuos hay en cada categoría:

```
datos <-
   read.csv("personas.csv") %>%
   mutate(EstadoCivil = as.factor(EstadoCivil)) %>%
   group_by(EstadoCivil) %>%
   summarize(media=mean(Edad), n=n())
```

> **Ejercicio 1:**
>
> Descarga este archivo CSV con cotizaciones de acciones de la bolsa española un día determinado, y utiliza los elementos vistos en este documento para crear un programa `cotizaciones.r` que muestre el nombre y el valor máximo de cotización de las 5 acciones con mayor valor (en orden descendente).

> **Ejercicio 2:** Descarga este CSV sobre datos de salud de distintos pacientes, y crea un programa `salud.r` que filtre todas las mujeres (columna *gender=F*) de 50 años o más que tengan problemas cardíacos. Calcula también el porcentaje sobre el total de mujeres de esa franja de edad.

> **AYUDA:** vídeo con solución del ejercicio