Introducción al tidyverse

David García Callejas

Lo que llamamos `tidyverse` es un conjunto de paquetes de R que están fuertemente integrados entre sí, a través de una sintaxis similar y de funciones compatibles entre ellos. La filosofía básica de los paquetes que forman el `tidyverse` es facilitar el análisis de datos, a través de operaciones "encadenadas" sobre conjuntos de datos. Muchas de las funciones del `tidyverse` son útiles por si mismas, como hemos ido viendo en las anteriores clases. Sin embargo, estas funciones demuestran todo su potencial cuando se usan encadenando operaciones, de manera que el resultado de una función es la entrada de otra, y así sucesivamente.

```
# cargamos el paquete
library(tidyverse)

# trabajaremos con los datos de terremotos
eq <- read.csv2(file = "../data/Earthquake_data.csv",
                dec = ".",
                stringsAsFactors = FALSE)

eq.clean <- eq[,c("En","Year","Mo","Da","Ho",
                  "Mi","Se","Area","Lat","Lon",
                  "Dep","M")]

# el encadenamiento se codifica con la orden %>%
# estas dos sintaxis son equivalentes
summary(eq.clean)
eq.clean %>% summary()
```

Una de las grandes fortalezas de estos paquetes es que se pueden concatenar operaciones

```
eq.clean %>%
  drop_na() %>%
  nrow()

eq.clean %>%
  filter(!is.na(Dep)) %>%
  summary()
```

Este tipo de sintaxis es muy utilizada para unir operaciones de filtrado, ordenación, agrupamiento, etc. Como primer ejemplo, usamos el dataset `iris` para obtener algunos datos a nivel de especie.

```
valores.medios <- iris %>%
  group_by(Species) %>%
  summarise(mean.sepal.length = mean(Sepal.Length),
            mean.petal.length = mean(Petal.Length))
```

También se pueden crear columnas nuevas en el dataframe original

```
iris2 <- iris %>%
  mutate(columna.nueva = Sepal.Length * Sepal.Width)
```

Muchas de estas operaciones (todas, en realidad) tienen formas alternativas en R "base". Al codificarlas en `tidyverse` ganamos flexibilidad y hacemos código más compacto y legible.

Veamos un ejemplo completo, en el que obtenemos la magnitud y profundidad medias de los terremotos en diferentes paises. Para ello, primero necesitamos asignar a nuestros datos de terremotos el país en el que se registraron. Definimos una función que devuelve el nombre del país a partir en el que se encuentra un punto de latitud y longitud determinadas. https://stackoverflow.com/questions/14334970/convert-latitude-and-longitude-coordinates-to-country-name-in-r

```
# necesitamos estos dos paquetes
library(sp)
library(rworldmap)

# points es un dataframe con valores de longitud (columna 1) y latitud (c2)
coords2country <- function(points){
  countriesSP <- rworldmap::getMap(resolution='low')

  #setting CRS directly to that from rworldmap
  pointsSP = sp::SpatialPoints(points,
                               proj4string=sp::CRS(sp::proj4string(countriesSP)))

  # use 'over' to get indices of the Polygons object containing each point
  indices = sp::over(pointsSP, countriesSP)

  # return the ADMIN names of each country
  indices$ADMIN
  #indices$ISO3 # returns the ISO3 code
  #indices$continent   # returns the continent (6 continent model)
  #indices$REGION   # returns the continent (7 continent model)
}
```

Creamos una función equivalente para devolver el continente de cada punto

```
coords2continent <- function(points){
  countriesSP <- rworldmap::getMap(resolution='low')

  #setting CRS directly to that from rworldmap
  pointsSP = sp::SpatialPoints(points,
                               proj4string=sp::CRS(sp::proj4string(countriesSP)))

  # use 'over' to get indices of the Polygons object containing each point
  indices = sp::over(pointsSP, countriesSP)

  # return the ADMIN names of each country
  # indices$ADMIN
  #indices$ISO3 # returns the ISO3 code
  indices$continent   # returns the continent (6 continent model)
  #indices$REGION   # returns the continent (7 continent model)
}
```

Ahora podemos obtener el país donde sucedió cada terremoto

```
eq.paises <- eq.clean %>%
  mutate(country = coords2country(data.frame(Lon,Lat)),
         continent = coords2continent(data.frame(Lon,Lat)))

table(eq.paises$continent)
```

```r
table(eq.paises$country)
```

y generar un dataframe con los valores que nos interesen por país/continente

```r
magnitud.eurasia <- eq.paises %>%
  filter(continent == "Eurasia") %>%
  summarise(mean.magnitude = mean(M),
            sd.magnitude = sd(M),
            min.year = min(Year))

# podemos repetir el mismo análisis para todos los continentes
magnitud.continentes <- eq.paises %>%
  group_by(continent )%>%
  summarise(mean.magnitude = mean(M),
            sd.magnitude = sd(M),
            num.eq = n(),
            min.year = min(Year))
```

Para realizar análisis sobre cadenas de caracteres (e.g. sentiment analysis), necesitamos instalar y cargar el paquete `tidytext`. Un ejemplo combinando tidyverse y tidytext:

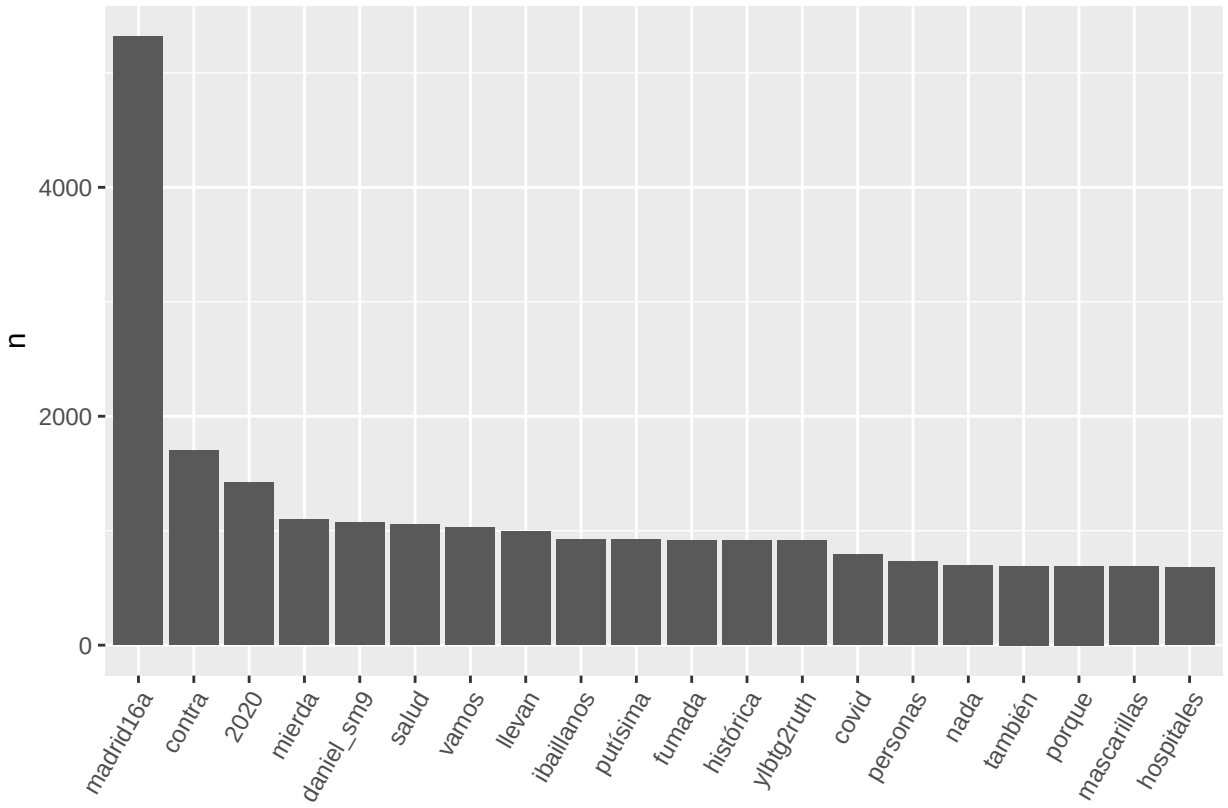
```r
library(tidytext)

tuits <- read.delim("../data/tuits_madrid_16a.csv",stringsAsFactors = FALSE)

# extraemos cada palabra de los tuits de cada usuario
tweet_words <- tuits %>%
  select(id.tweet, text) %>%
  unnest_tokens(word,text)

# seleccionamos palabras de más de 3 caracteres y eliminamos algunas irrelevantes
invalid_words <- c("none","https","t.co","twitter.com","status","false",
                   "images","twitter","pbs.twimg.com","profile_images",
                   "están","esto","para","android")
tweet_words_2 <- tweet_words %>%
  filter(nchar(word) > 3) %>%
  filter(!word %in% invalid_words)

# qué palabras son las 20 más repetidas
tweet_words_2 %>% count(word,sort=T) %>%
  slice(1:20) %>%
  ggplot(aes(x = reorder(word,
    n, function(n) -n), y = n)) +
  geom_bar(stat = "identity") +
  theme(axis.text.x = element_text(angle = 60, hjust = 1)) +
  xlab("")
```



Y un pequeño sentiment analysis

```
# necesita el paquete "textdata"
# bing_lex <- get_sentiments("nrc")

# fn_sentiment <- tweet_words_2 %>% left_join(bing_lex)

# fn_sentiment %>% filter(!is.na(sentiment)) %>% group_by(sentiment) %>% summarise(n=n())
```