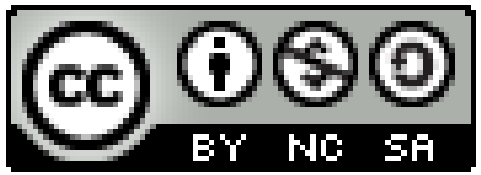
Aprendiendo SIG con The Lord Of The Rings

La comunidad de Leaflet

Jose Antonio Sánchez Martí

Enero 2020



Introducción

Objetivos del taller

- Leer y manejar archivos espaciales.
- Construir mapas estáticos y dinámicos.
- Aprendizaje básico sobre la librería Leaflet.

Librerías: leaflet, rgdal, foreign, htmlwidgets.

Materiales: CIUDADES *(csv)* y GEO *(archivos espaciales)*.

Reglas

Crear tu propio SIG basado en el señor de los anillos.

Frodo no pudo cumplir su misión y murio en Orodruin. Ahora los elfos necesitan tu ayuda para preparar la defensa de la tierra media.

Crea un mapa interactivo del señor de los anillos, guardalo en html y entregalo al rey elfo.

Regla 1: Cada habitante de cada raza tiene el mismo peso en la batalla decisiva.

Regla 2: Solamente los elfos tienen el doble de fuerza y cuentan por dos, si por lo menos una de cada tres áreas de superficie son bosques.

Preguntas

Lee los datos de las formas geométricas y de las ciudades del Señor de los Anillos.

- Haz un gráfico con la suma de población de las distintas razas. Debe aparecer visible la raza y el porcentaje de cada una, para cada raza un color. ¿Quién gana la batalla?
- Calcula el porcentaje de bosques. ¿Ganarán la batalla o los codiciosos hombres han talado los bosques?
- Todavía queda una última esperanza, elabora un mapa interactivo siguiendo las instrucciones de la valoración, y entregaselo al rey elfo, esto ayudará a refoestar y habrá un punto porcentual extra de superficie. ¿Hay futuro para la tierra media?

Puntos evaluación

- Leer los datos (shp y csv) (0.5 puntos).
- Gráfico de las razas y contestar pregunta batalla (1.5 puntos).
- Calcula el porcentaje de bosques sobre el total de la superficie, y contestar pregunta bosques (1 punto).
- Mapa interactivo (6 puntos).
  - Limita el zoom mínimo (0.5 puntos).
  - Añade un marco negro al mapa (0.5 puntos).
  - Añade las formas GEO, sin bordes, con contenido del nombre, y dales un color realista (1.5 puntos).
  - Añade las ciudades como círculos compactos, de radio 2, con color igual al gráfico de razas, y con contenido (1.5 puntos).
  - Añade una leyenda con título, razas y su color (1 punto).
  - Añade un control de tiles para GEO y ciudades, con overlayGroups y colapsado (1 punto).
- Entrega tu mapa interactivo en html (1 punto).

Trabajando datos espaciales

csv con coordenadas

Leyendo un csv

Leemos un csv con las ciudades. Examinamos su información.

```
CIUDADES <- read.csv("Ciudades.csv")
str(CIUDADES)
```

```
## 'data.frame':    32 obs. of  6 variables:
##  $ ID    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ NOMBRE: Factor w/ 32 levels "Abismo de Helm",..: 31 15 29 21 10 12 1 27 2
##  $ TIPO  : Factor w/ 5 levels "Elfo","Enano",..: 4 4 1 2 2 4 4 4 2 5 ...
##  $ POB   : int  10000 15000 30000 10000 20000 15000 40000 15000 25000 250001
##  $ X     : num  3.73 1.02 8.4 7.53 6.28 ...
##  $ Y     : num  50.2 53.7 53 50.6 47.7 ...
```

- ¿Cuál es la población total?
- ¿Cuántas razas hay?, ¿y cuál es la población de cada una?

Población total

Calculamos la población total.

```
sum(CIUDADES$POB)
```

```
## [1] 1000001
```

```
levels(CIUDADES$TIPO)
```

```
## [1] "Elfo"   "Enano"  "Hobbit" "Humano" "Mal"
```

Desglosamos la población por razas, y obtenemos un data frame con la información. ¿Qué función podemos usar?

Población por razas

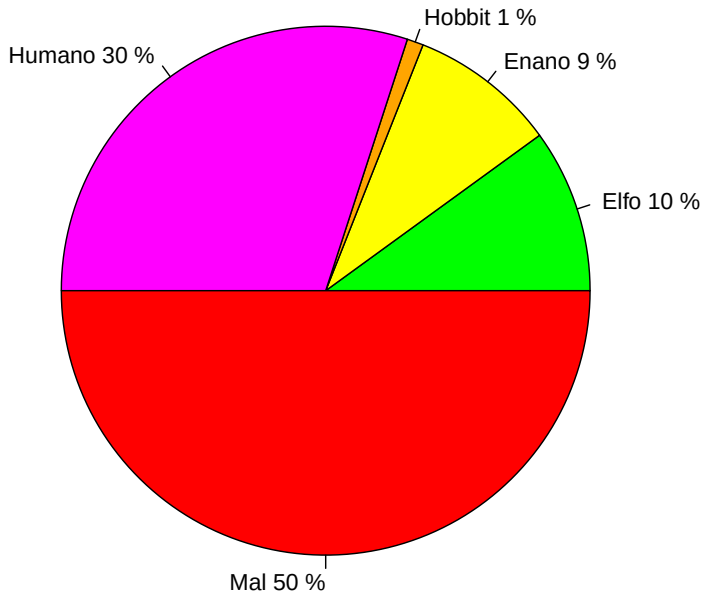
Para realizar la suma de cada uno de los niveles de raza, usamos la función **aggregate**. Nota: los niveles deben ir en lista.

```
CIUDADES.POB<- aggregate (
  CIUDADES$POB, list(CIUDADES$TIPO), sum)
names(CIUDADES.POB) <- c("Raza", "Población")
CIUDADES.POB
```

```
##     Raza Población
## 1   Elfo    100000
## 2  Enano     90000
## 3 Hobbit     10000
## 4 Humano    300000
## 5    Mal    500001
```

Gráfico de población

Realizamos un gráfico en forma de porciones, con la intención de visualizar la información. Usamos la función **pie**, especificando los pesos, una etiqueta de cada poción, y un color.



Solución gráfico de población

```
CIUDADES.POB$peso <- round(
  CIUDADES.POB$Población/sum(CIUDADES.POB$Población),2)
pie.POB <- pie(CIUDADES.POB$peso,
               paste(levels(CIUDADES.POB$Raza),(CIUDADES.POB$peso)*100, "%" ),
               col= c("green", "yellow", "orange", "magenta", "red" ))
```

Tipos de archivos espaciales

Extensiones

- **shp** Archivo con las formas geométricas.
- **shx** Índice de las formas geométricas.
- **dbf** Base de datos.
- **prj** Proyección de los datos espaciales. Nos identifica el Sistema de Referencia de Coordenadas (CRS), nos sirve para posicionarnos sobre el globo terraqueo.

Leyendo y guardando una base de datos

Para manejar un fichero **dbf**, necesitamos la libería **foreign**, también podemos abrir el archivo con LibreOffice. Leemos con **read.dbf**, y guardamos con **write.dbf**.

```
library(foreign)
# Leemos un archivo dbf
GEO_DBF <- read.dbf("GEO.dbf")

# Guardamos un data frame en dbf
write.dbf(CIUDADES, "CIUDADES.dbf")
```

Estructura de un dbf

En un dbf, en la primera fila se especifican las cabeceras. Nombre variable, tipo de dato, límite.

- **Texto (C)** Despúes de coma se establece el máximo de caracteres de la variable. Ejemplo: Texto, C, 16.
- **Número (N)** Se ponen dos comas, una para el número entero, y otra para los decimales. Ejemplo: Numero, N, 6, 2.

Leyendo archivos espaciales

libreria rgdal

Para leer los datos espaciales, necesitamos la librería **rgdal**, y utilizamos la función **readOGR** con los argumentos: **dsn** el archivo shp, **layer** idem sin extensión, y **encoding** trabajamos con extensión que es ESRI shapefile.

```
library(rgdal)
GEO <- readOGR(dsn="GEO.shp", layer="GEO",
               encoding = "ESRI Shapefile")
```

Examinando datos espaciales

Se pueden acceder a los datos del shapefile o del dbf. Se recomienda usar los espaciales, que van precedidos por @. Dentro de @ tenemos acceso a los datos, coordenadas, proyección, poligonos...

Entramos a los datos.

```
GEO$NOMBRE
GEO@data$NOMBRE
```

```
levels(GEO@data$TIPO)
```

```
## [1] "Agua"   "Bosque" "Isla"   "Monte"
```

Examinamos las coordenadas

Entramos a las coordenadas máximas y mínimas.

```
GEO@bbox
```

```
##          min     max
## x   18631.71 2768029
## y 4726547.99 6364000
```

Hacemos plot

Si hacemos **plot** a nuestra variable, nos dibuja un mapa. Le añadimos un título, y le damos color.

```
plot(GEO, border="black",
     col=rainbow(length(GEO@data$NOMBRE)),
     main="Geografía de la Tierra Media")
```

Un primer mapa estático

Calculamos áreas de bosque

Calculamos el área de los bosques en la Tierra Media, una forma es filtrando los datos por su tipología.

Solución pregunta área de bosque

```
BOSQUES <- GEO[GEO@data$TIPO=="Bosque",]
sum (BOSQUES@data$AREA) / sum(GEO@data$AREA)
```

```
## [1] 0.3280735
```

Guardamos el archivo espacial de los bosques

Para guardar los polígonos espaciales de los bosques, usamos la función **writeOGR**, y le especificamos nuestro objeto que queremos guardar, el nombre de archivo shp, del resto de archivos, y el tipo.

```
writeOGR(BOSQUES, dsn="BOSQUES.shp", layer="BOSQUES",
         driver = "ESRI Shapefile" )
```

Conociendo leaflet

Cargando leaflet

Para crear mapas interactivos necesitamos la librería de **leaflet**. Podemos llamar a la función **leaflet()** para generar un mapa interactivo vacío.

```
library(leaflet)
leaflet()
```

Pipe

Una vez que tenemos leido el archivo espacial, podemos añadirle cosicas a nuestro leaflet vacio, para ello necesitamos añadir **%>%** antes de la función **addTiles**, por ejemplo. Con ello, le indicamos a nuestro leaflet que después viene esa acción. Por defecto, nos cargara Open Street Map.

```
library(leaflet)
leaflet() %>% addTiles()
```

Añadiendo polígonos

Para añadir los polígonos, es simple, le indicamos a leaflet que queremos añadir cosas con **%>%**, y en este caso usamos la función **addPolygons**.

```
library(leaflet)
leaflet() %>% addPolygons(data=GEO)
```

¿Algo va mal?...

Transformando SRC

Necesitamos trabajar con el Sistema de Coordenadas de Referencia (CRS). Y tenemos que transformar el CRS con el especifico de leaflet, que usa **+init=epsg:4326**. Para dicha tarea, usamos la función **spTransform**.

```
GEO <- spTransform(GEO, CRS("+init=epsg:4326"))
leaflet() %>% addPolygons(data=GEO)
```

Etiquetando

Con la opción **label** añadimos etiquetas, en este caso el nombre de cada accidente geográfico. Dentro de label, tenemos otra opción interesante con **noHide**, podemos hacer que se oculten o no.

```
leaflet() %>% addPolygons(data = GEO,
                          label = GEO@data$NOMBRE,
                          labelOptions = labelOptions(noHide = F))
```

Una pequeña prueba...

¿Dónde está mordor en el mundo real?

Añadiendo cosicas

Coloreando

Crear una paleta

Vamos a crear unas paletas tanto para las distintas razas, como para cada tipología geográfica. Para especificar los colores, primero creamos una variable con los colores que queremos, y la ponemos como argumento a **colorFactor**.

```
color.CIUDADES <- c("green", "yellow", "orange", "magenta", "red" )
paleta.CIUDADES <- colorFactor(color.CIUDADES, CIUDADES$TIPO)

color.GEO <- c("blue", "darkgreen", "black", "tan")
paleta.GEO <- colorFactor(color.GEO, GEO@data$TIPO)
```

Contenido

Contenido en html

Creamos el contenido que aparece cuando pinchamos un polígono. Le damos un formato con html.

```
#<b> ... </b> ponemos negrita.

#Con <br/> señalamos un salto de línea.
```

```
contenido.GEO <-GEO$NOMBRE
```

```
contenido.CIUDADES <- paste ( "<b>DATOS</b>", "<br/>",
                          "<b> Nombre:</b>", CIUDADES$NOMBRE, "<br/>",
                          "<b> Tipo:</b>", CIUDADES$TIPO, "<br/>",
                          "<b> Población:</b>", CIUDADES$POB, "<br/>")
```

Polígonos y puntos

Marco a nuestro mapa

Con la función **addRectangles** podemos añadir un rectángulo a nuestro mapa, que nos hace de marco. ¿Cómo representar los límites de nuestro mapa? Necesitamos dichas coordenadas es decir, especificar **lng** y **lat**, además de que el rectángulo sea transparente con bordes de color negro.

```
leaflet()%>% addRectangles(lng1=GEO@bbox[1,1], lat1=GEO@bbox[2,2],
                                        lng2=GEO@bbox[1,2], lat2=GEO@bbox[2,1],
                                        fillColor = "transparent", color="black")
```

Polígonos

Para añadir los polígonos, usamos la función **addPolygons**. Dentro de la función le indicamos los datos, con **stroke** manipulamos los bordes. Mientras que **color** es usado para las fronteras del polígono, **fillColor** es usado para rellenar el polígono. El contenido lo añadimos con **popup**.

```
%>% addPolygons(data = GEO, popup=contenido.GEO, stroke=F,
                fillColor = paleta.GEO(GEO@data$TIPO))
```

Añadiendo puntos

Añadimos las coordenadas y las representamos con la función **addCircleMarkers**, especificando que **lng** es el eje X y **lat** el eje Y. Le ponemos con **radius** 2, los queremos todo ópacos con **opacity** y **fillOpacity**. Por último, le añadimos el contenido y le damos color con la paleta correspondiente, en este caso usamos la opción **color**.

```
%>% addCircleMarkers(lng=CIUDADES$X, lat=CIUDADES$Y, popup=contenido.CIUDADES,
                     radius=2, opacity = 1, fillOpacity = 1,
                     color =paleta.CIUDADES(CIUDADES$TIPO))
```

Cambiando el radio según la población

Podemos cambiar el radio (se mide en pixel, con circles en metros) para que cambie en función de su tamaño poblacional. Con **weight** indicamos que el borde de cada círculo es de un pixel. Para ello, estudiamos la distribución de la población. Una idea es que la ciudad menos poblada sea la unidad, y que como máximo no llegue a más bits que una Sega Mega Drive.

```
%>% addCircleMarkers(lng=CIUDADES$X, lat=CIUDADES$Y, popup=contenido.CIUDADES,
                     radius= sqrt(CIUDADES$POB)/sqrt(min(CIUDADES$POB)),
                     opacity = 1, fillOpacity = 1, weight = 1,
                     color =paleta.CIUDADES(CIUDADES$TIPO))
```

Puntos con Makers

Leaflet nos permite cambiar círculos por iconos, en este caso añadimos la función **addMakers**, pero nos abre las puertas a poder cambiar los iconos.

```
%>% addMarkers(lng=CIUDADES$X, lat=CIUDADES$Y,
               popup=contenido.CIUDADES)
```

Cambiando los iconos

Podemos cambiar a iconos ya hechos, y que se encuentran en las librerías de Font Awesome (**fa**). Como vamos a cambiar de icono entre los buenos y los malos, creamos una lista con dos iconos con **awesomeIconList**. Para cada icono usamos la función **makeAwesomeIcon**, y dentro seleccionamos el nombre del icono, los colores y la librería a la que pertenece.

```
Iconos <- awesomeIconList(
  Bien = makeAwesomeIcon(icon = "fab fa-linux",
  iconColor = "black", markerColor = "blue",
  library = "fa"),
  Mal = makeAwesomeIcon(icon = "fab fa-windows",
  iconColor = "black", markerColor = "red",
  library = "fa"))
```

Aplicando los iconos personalizados

Una vez tenemos los iconos hechos, falta aplicarlos a una variable. Por ello, creamos la variable CAT, que nos indica la bondad o maldad de cada raza. Para añadir los marcadores creados, los añadimos con la función **addAwesomeMarkers**, tal como lo habiamos hecho anteriormente, salvo que incorporamos la opción **icon**, indicando la variable a la que hacemos referencia.

```
# Creamos la categoría
CIUDADES$CAT <- CIUDADES$TIPO
levels(CIUDADES$CAT) <- c("Bien", "Bien", "Bien", "Bien", "Mal")

# Añadimos a leaflet las marcas con los iconos
%>% addAwesomeMarkers(lng=CIUDADES$X, lat=CIUDADES$Y,
                      popup=contenido.CIUDADES, icon=Iconos[CIUDADES$CAT])
```

Leyenda

Añadiendo una leyenda

Para añadir la leyenda usamos la función **addLegend**. En primer lugar seleccionamos su posición, podemos usar coordenadas o un comando como **"topright"**. Después indicamos con **pal** la paleta, la variable a usar **values**, y el título con **title**.

```
%>% addLegend("topright", pal = paleta.CIUDADES,
              values =CIUDADES$TIPO, title = "Razas")
```

Algo más avanzado

Jugando con el zoom

Cambiando el zoom

Dentro de las opciones de leaflet, podemos controlar el número de zoom, para este caso establecemos un mínimo de 5.

```
leaflet(options = leafletOptions(minZoom = 5))
```

Cambiando la vista a los hobbits

Para cambiar la vista, le añadimos una nueva configuración de la vista con **setView**, especificamos las coordenadas y el nivel de zoom inicial. Creamos un data frame con las coordenadas de Hobbytown.

```
H_coor <- CIUDADES[CIUDADES$NOMBRE=="Hobbytown",5:6]
leaflet() %>%  setView(H_coor$X, H_coor$Y , zoom = 8)
```

Controles de grupos

Estableciendo controles

Para establecer los controles, primero identificamos los grupos con **group**, que son los polígonos y los puntos.

```
%>% addPolygons(data = GEO, popup=contenido.GEO,
stroke=F, fillColor = paleta.GEO(GEO@data$TIPO), group="GEO")
```

Después, con la función **addLayersControl**, especificamos en **overlayGroups** los grupos, también puede usarse **baseGroups**. Por último, escondemos los controles hasta que los usemos, así no molestan.

```
%>% addLayersControl(overlayGroups = c("GEO", "CIUDADES"),
                     options = layersControlOptions(collapsed = TRUE))
```

Para finalizar

Guardando mapas interactivos

Código completo

```
MAPA <- leaflet(options = leafletOptions(minZoom = 5)) %>%
  addRectangles(lng1=GEO@bbox[1,1], lat1=GEO@bbox[2,2], lng2=GEO@bbox[1,2],
                lat2=GEO@bbox[2,1], fillColor = "transparent",
                color="black") %>%
  addPolygons(data = GEO, popup=contenido.GEO,stroke=F,
              fillColor = paleta.GEO(GEO@data$TIPO), group="GEO") %>%
  addCircleMarkers(lng=CIUDADES$X, lat=CIUDADES$Y, popup=contenido.CIUDADES,
                   radius=2, opacity = 1, fillOpacity = 1,
                   color =paleta.CIUDADES(CIUDADES$TIPO), group="CIUDADES") %>%
  addLayersControl(overlayGroups = c("GEO", "CIUDADES"),
                   options = layersControlOptions(collapsed = TRUE)) %>%
  addLegend("topright", pal = paleta.CIUDADES,
            values =CIUDADES$TIPO, title = "Razas")
```

Guardando el mapa en html

Para guardar nuestro mapa creado, necesitamos la librería **htmlwidgets**. Usamos la función **saveWidget** para guardar nuestro mapa, le escribimos el nombre de nuestro archivo con extensión html.

```
library (htmlwidgets)
saveWidget( MAPA, file= "MAPA_ANILLOS.html")
```

Más información

Seguir aprendiendo

- **Tutorial leaflet R** https://rstudio.github.io/leaflet/
- **Tutorial leaflet js** https://leafletjs.com/
- **Iconos fa** https://fontawesome.com/
- **Web personal** https://sanchezmarti.wordpress.com/

MUCHAS GRACIAS

¡MUCHAS GRACIAS POR SU ATENCIÓN!