Fundamentos de la Ciencia de Datos Práctica 6

Fernández Díaz, Daniel
Cano Díaz, Francisco
Fernández Hernández, Alberto

10 de diciembre del 2019

Índice

1. Mapas estáticos

Los mapas estáticos (*Static Maps*) es uno de los métodos de visualización más utilizados, el cual consiste en **visualizar una sola imagen**. Una de las herramientas más utilizadas para visualizar gráficos de mapas es *ggplot2*. Este paquete se basa en la **gramática de gráficos**, que consiste en una serie de componentes:

- Datos a representar
- Aspectos estéticos (*aesthetics*), como ejes de coordenadas, leyendas etc.
- Objetos geométricos (puntos, lineas, polígonos, áreas etc.)
- Escalas
- Coordenadas

Para este ejemplo visualizaremos los **porcentajes de asaltos producidos en Estados Unidos, argupados por Estados**. En primer lugar, instalamos el paquete *ggplot2*:

```
> # Cargamos el paquete ggplot2
> if(!require(ggplot2)){
+   install.packages("ggplot2")
+   require(ggplot2)
+ }
```

Una vez instalado el paquete, debemos obtener las coordenadas de **latitud** y **longitud** de cada uno de los estados. Para ello, nos descargamos un *dataframe* que contienen las coordenadas limítrofes de cada Estado, importado desde la librería *ggplot2*. Por otro lado, descargamos el *dataset* que contiene los **porcentajes de asaltos por cada Estado**:

```
> # Descargamos las coordenadas de cada estado
> if (!require(maps)) {
+   install.packages("maps")
+   require(maps)
+ }
> estados <- map_data("state")
> head(estados)

       long      lat group order  region subregion
1 -87.46201 30.38968     1     1 alabama      <NA>
2 -87.48493 30.37249     1     2 alabama      <NA>
3 -87.52503 30.37249     1     3 alabama      <NA>
4 -87.53076 30.33239     1     4 alabama      <NA>
5 -87.57087 30.32665     1     5 alabama      <NA>
6 -87.58806 30.32665     1     6 alabama      <NA>

> # Descargamos el dataset del numero de asaltos
> arrestos <- USArrests
> # Ponemos los noimbres de las columnas a minusculas
> names(arrestos) <- tolower(names(arrestos))
> arrestos$region <- tolower(rownames(USArrests))
> head(arrestos)
```

```
           murder assault urbanpop rape     region
Alabama      13.2     236       58 21.2    alabama
Alaska       10.0     263       48 44.5     alaska
Arizona       8.1     294       80 31.0    arizona
Arkansas      8.8     190       50 19.5   arkansas
California    9.0     276       91 40.6 california
Colorado      7.9     204       78 38.7   colorado
```

Una vez descargados, unimos ambos *dataframes* mediante la función *merge*

```
> df <- merge(estados, arrestos, sort = FALSE, by = "region")
> df <- df[order(df$order), ]
> head(df)
```

```
   region      long      lat group order subregion murder assault urbanpop rape
1 alabama -87.46201 30.38968     1     1      <NA>   13.2     236       58 21.2
2 alabama -87.48493 30.37249     1     2      <NA>   13.2     236       58 21.2
6 alabama -87.52503 30.37249     1     3      <NA>   13.2     236       58 21.2
7 alabama -87.53076 30.33239     1     4      <NA>   13.2     236       58 21.2
8 alabama -87.57087 30.32665     1     5      <NA>   13.2     236       58 21.2
9 alabama -87.58806 30.32665     1     6      <NA>   13.2     236       58 21.2
```

Por último, representaremos gráficamente los datos mediante la función *ggplot*. Para ello, especificamos los siguientes parámetros:

- *ggplot*: indicamos el *dataframe*. Por otro lado, establecemos las coordenadas de longitud para el eje X, así como las coordenadas de latitud para el eje Y.
- *geom_polygon*: para dibujar las diferentes fronteras.
- *coord_map*: para proyectar una parte del globo terráqueo (concretamente la región de Estados Unidos).

```
> if(!require(mapproj)){
+   install.packages("mapproj")
+   require(mapproj)
+ }
> ggplot(df, aes(long, lat)) +
+   # Con fill rellenamos cada estado en funcion del asalto
+   geom_polygon(aes(group = group, fill = assault)) +
+   # (29.5 , 45.5) Coordenadas de EEUU
+   coord_map("albers",  at0 = 45.5, lat1 = 29.5)
```

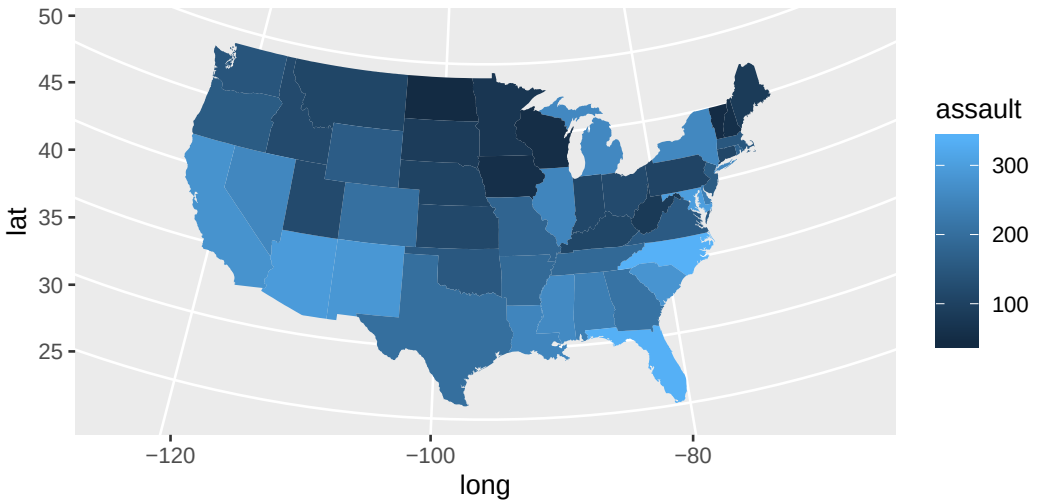


Figura 1: Total de asaltos producidos en Estados Unidos

2. *Bubble Maps*

Los mapas de burbujas (*Bubble Maps*) permiten representar **grandes volúmenes de datos, agrupados por intervalos**, donde cada intervalo se representa mediante un **círculo de diferente diámetro**. Para ello, utilizaremos el paquete *tmap*. Este paquete ofrece un conjunto de herramientas que proporcionan una sintaxis concisa para la visualización de mapas **utilizando la menor cantidad de código posible**. Se basa en el concepto de **gramática de gráficos** (al igual que *ggplot2*), separando los datos de entrada de la estética (**cómo visualizar los datos**). Para comenzar, **crearemos un *mapa-mundi* en el que visualizaremos el índice de felicidad de cada país**. Para ello, realizamos los siguientes pasos:

1. En primer lugar, debemos proyectar la plantilla del mapa. Para ello utilizaremos la función *tm_shape* para proyectar el *mapa-mundi*.

2. Por otro lado, rellenamos cada uno de los países en función del índice de felicidad

Para proyectar el mapa, es necesario un **DataFrame espacial** (*Spatial Dataframe*), el cual consiste en un conjunto de arcos y flechas unidos por puntos, proyectando finalmente el mapa. Para visualizar el *mapa-mundi*, utilizaremos el *dataframe* espacial *World*, disponible en el paquete *tmap*:

```
> # Para concatenar diferentes funciones
> # descargamos el paquete dplyr
> # (para concatenar utilizamos %>%)
> if(!require(dplyr)){
+   install.packages("dplyr")
+   require(dplyr)
+ }
> # Cargamos el paquete tmap
> if(!require(tmap)){
+   install.packages("tmap")
+   require(tmap)
+ }
> # Cargamos el dataset y los convertimos a dataframe
> data("World")
> World %>% as.data.frame() %>% head()
  iso_a3                 name           sovereignt     continent
1    AFG          Afghanistan          Afghanistan          Asia
2    AGO               Angola               Angola        Africa
3    ALB              Albania              Albania        Europe
4    ARE United Arab Emirates United Arab Emirates          Asia
5    ARG            Argentina            Argentina South America
6    ARM              Armenia              Armenia          Asia
               area  pop_est pop_est_dens                   economy
1  652860.00 [km^2] 28400000     43.50090 7. Least developed region
2 1246700.00 [km^2] 12799293     10.26654 7. Least developed region
3   27400.00 [km^2]  3639453    132.82675      6. Developing region
4   71252.17 [km^2]  4798491     67.34519      6. Developing region
5 2736690.00 [km^2] 40913584     14.95003   5. Emerging region: G20
6   28470.00 [km^2]  2967004    104.21510      6. Developing region
              income_grp gdp_cap_est life_exp well_being footprint inequality
1          5. Low income    784.1549   59.668        3.8      0.79  0.4265574
2  3. Upper middle income   8617.6635       NA         NA        NA         NA
3  4. Lower middle income   5992.6588   77.347        5.5      2.21  0.1651337
4 2. High income: nonOECD  38407.9078       NA         NA        NA         NA
5  3. Upper middle income  14027.1261   75.927        6.5      3.14  0.1642383
6  4. Lower middle income   6326.2469   74.446        4.3      2.23  0.2166481
       HPI                       geometry
1 20.22535 MULTIPOLYGON (((5310471 451...
2       NA MULTIPOLYGON (((1531585 -77...
3 36.76687 MULTIPOLYGON (((1729835 521...
4       NA MULTIPOLYGON (((4675864 313...
5 35.19024 MULTIPOLYGON (((-5017766 -6...
6 25.66642 MULTIPOLYGON (((3677241 513...
```

Como podemos observar, lo que nos devuelve es un *dataframe espacial* cuya última columna (*geomtry*) contiene las dimensiones de cada polígonos que queremos representar (en este caso, la forma de cada país). Una vez cargado el *dataframe* espacial, visualizamos los niveles de felicidad por país, utilizando las siguientes funciones:

- *tm_shape*: para proyectar el mapa anterior.
- *tm_polygons*: para visualizar un dataset sobre una plantilla *tm_shape*

```
> # 1. tm_shape: Utilizamos como plantilla el dataframe espacial
> # del mapa-mundi
>
> # 2. tm_polygons: visualizamos cada uno de los poligonos
> # HPI: Happiness Index
> # palette: seleccionamos la paleta de colores
> tm_shape(World) + tm_polygons("HPI", palette = "-Blues")
>
```

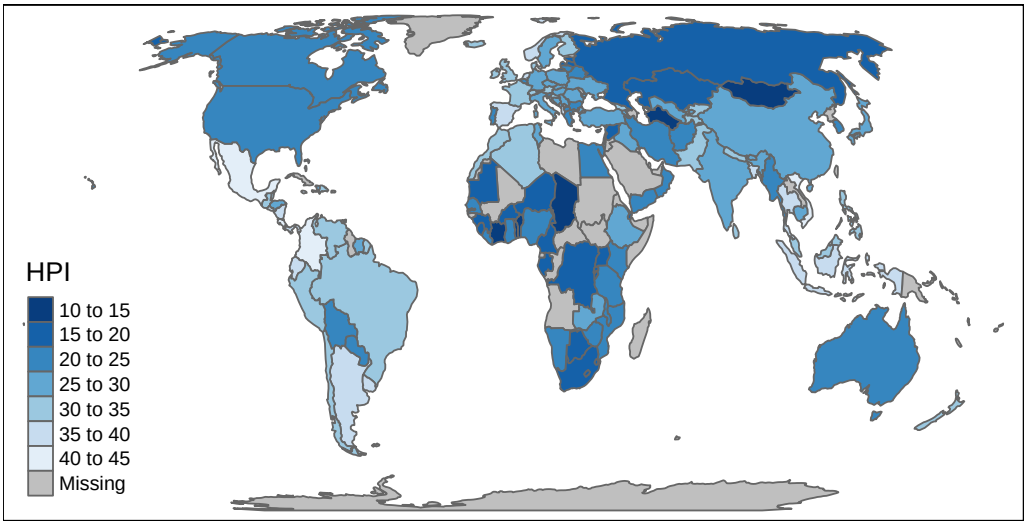


Figura 2: Indice de felicidad por país

tmap nos permite añadir más capas sobre la plantilla original, por lo que vamos a visualizar el mapa anterior junto con el nuevo mapa de burbujas. Para ello, sobre la plantilla original creada con la función *tm_shape*, representaremos mediante la función *tm_polygons* los índices de felicidad por país y la población por ciudad (empleando la función *tm_bubbles* para representar los datos de población por puntos)

```
> # 1, En primer lugar, cargamos los datos del total de
> # poblacion de las grandes metropolis
> data(metro)
> # A continuacion, visualizamos los indices de felicidad
> # mediante poligonos, asi como el total de poblacion con
> # burbujas
> tm_shape(World) + tm_polygons("HPI") + tm_shape(metro) + tm_bubbles("pop2010")
```

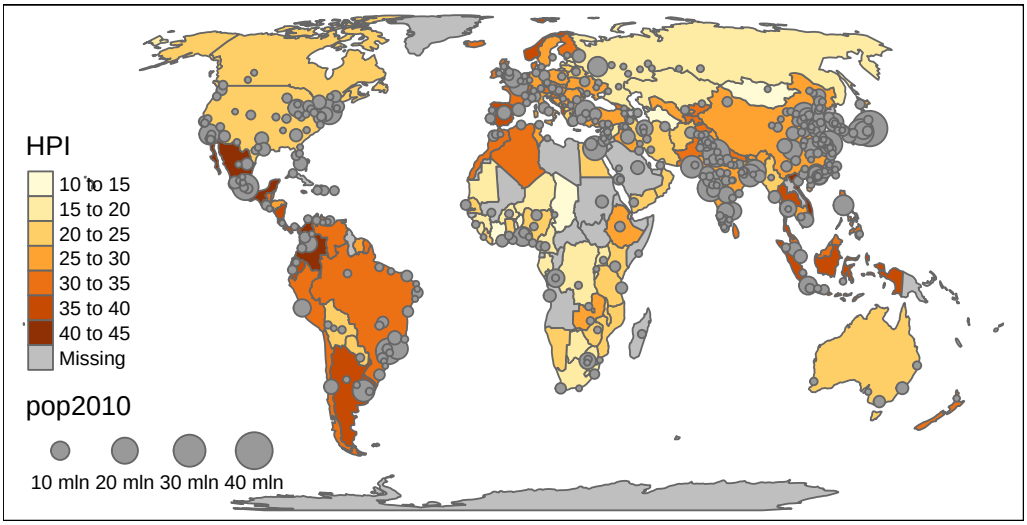


Figura 3: Indice de felicidad por país-Poblacion mundial del año 2010

3. Representar un *dataset* sobre un mapa

Una vez visto las diferentes herramientas para visualizar mapas en R, vamos a **representar datos sobre un mapa**. Por ejemplo, vamos a visualizar **los diferentes aeropuertos dispersos por Europa**. Para ello, nos descargamos un *dataset* que contiene las coordenadas de **latitud** y **longitud** de los diferentes aeropuertos: [1]

```
> # Descargamos el dataset
> aeropuertos <- read.csv("https://raw.githubusercontent.com/jpatokal/openflights/master/data/airports.c
> # Cambiamos el nombre de las columnas
> colnames(aeropuertos) <- c("ID", "nombre", "ciudad", "pais",
+ "IATA_FAA", "ICAO", "lat", "lon", "altitud", "zona_horaria", "DST")
> # Mostramos los 10 primeros datos
> head(aeropuertos,10)
```

```
   ID                                      nombre       ciudad             pais
1   1                           Goroka Airport       Goroka Papua New Guinea
2   2                           Madang Airport       Madang Papua New Guinea
3   3                Mount Hagen Kagamuga Airport  Mount Hagen Papua New Guinea
4   4                           Nadzab Airport       Nadzab Papua New Guinea
5   5 Port Moresby Jacksons International Airport Port Moresby Papua New Guinea
6   6                 Wewak International Airport        Wewak Papua New Guinea
7   7                         Narsarsuaq Airport Narssarssuaq        Greenland
8   8                     Godthaab / Nuuk Airport     Godthaab        Greenland
9   9                       Kangerlussuaq Airport  Sondrestrom        Greenland
10 10                           Thule Air Base        Thule        Greenland
   IATA_FAA ICAO       lat      lon altitud zona_horaria DST
1       GKA AYGA -6.081690 145.3920    5282           10   U
2       MAG AYMD -5.207080 145.7890      20           10   U
3       HGU AYMH -5.826790 144.2960    5388           10   U
4       LAE AYNZ -6.569803 146.7260     239           10   U
5       POM AYPY -9.443380 147.2200     146           10   U
6       WWK AYWK -3.583830 143.6690      19           10   U
7       UAK BGBW 61.160500 -45.4260     112           -3   E
8       GOH BGGH 64.190903 -51.6781     283           -3   E
9       SFJ BGSF 67.012222 -50.7116     165           -3   E
10      THU BGTL 76.531197 -68.7032     251           -4   E
                     NA      NA          NA
1  Pacific/Port_Moresby airport OurAirports
2  Pacific/Port_Moresby airport OurAirports
3  Pacific/Port_Moresby airport OurAirports
4  Pacific/Port_Moresby airport OurAirports
5  Pacific/Port_Moresby airport OurAirports
6  Pacific/Port_Moresby airport OurAirports
7       America/Godthab airport OurAirports
8       America/Godthab airport OurAirports
9       America/Godthab airport OurAirports
10        America/Thule airport OurAirports
```

[1] https://openflights.org/data.html

Este *dataframe* contiene las siguientes columnas:

- **ID del aeropuerto**
- **Nombre del aeropuerto**
- **Ciudad**
- **Pais**
- **Código Internacional del aeropuerto** (IATA_FAA)
- **Código de la Organización de Aviación Civil Internacional** (ICAO)
- **Coordenadas latitud, lontigud y altitud**
- **Zona franja horaria**

Una vez descargado el *dataset*, representamos gráficamente el *mapa-mundi*. Para ello, utilizaremos el paquete *rworldmap*:

```
> if(!require(rworldmap)){
+   install.packages("rworldmap")
+   require(rworldmap)
+ }
> newmap <- getMap(resolution = "low")
> plot(newmap, xlim = c(-20, 59), ylim = c(35, 71), asp = 1)
> points(aeropuertos$lon, aeropuertos$lat, col = "red", cex = .6)
```

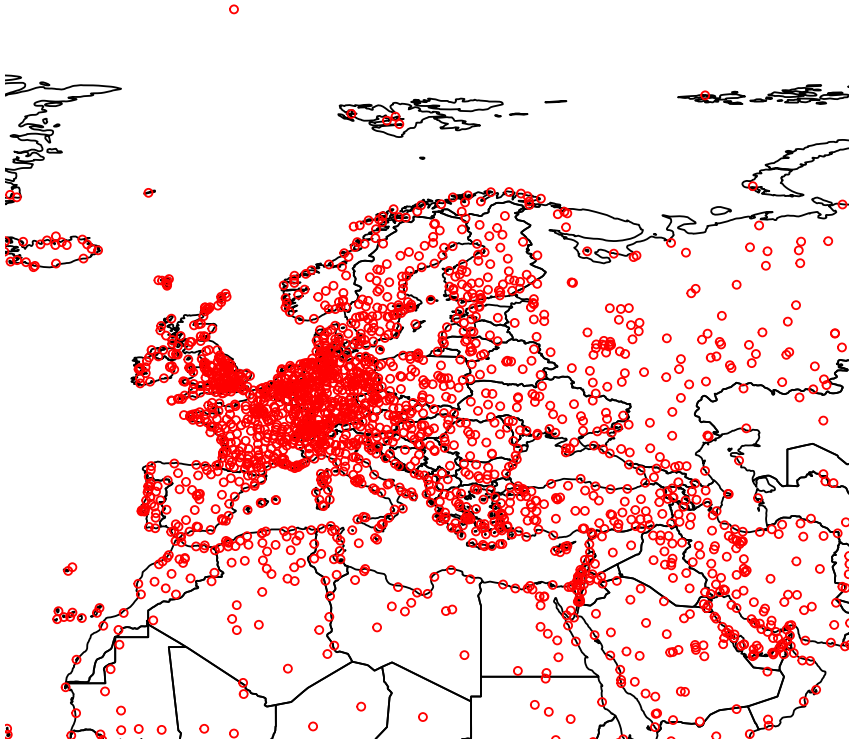


Figura 4: Localización de aeropuertos

4. Mapas hexagonales

Los mapas hexagonales (*Hexbin maps*), permiten visualizar los datos mediante hexágonos de diferentes tonalidades. Valores de tonalidad altos indican una **mayor concentración**, mientras que valores bajos indican una **menor concentración**. A modo de ejemplo, **visualizaremos la cantidad de *tweets* publicados con la etiqueta *surf* alrededor de Europa**. Junto con el paquete *ggplot2* utlizaremos el paquete *viridis* que contiene la función *scale_fill_viridis* con el que visualizaremos una leyenda sobre el mapa. En primer lugar, descargamos el paquete *viridis*:

```
> if(!require(viridis)){
+   install.packages("viridis")
+   require(viridis)
+ }
>
```

A continuación, descargamos el *dataset*:

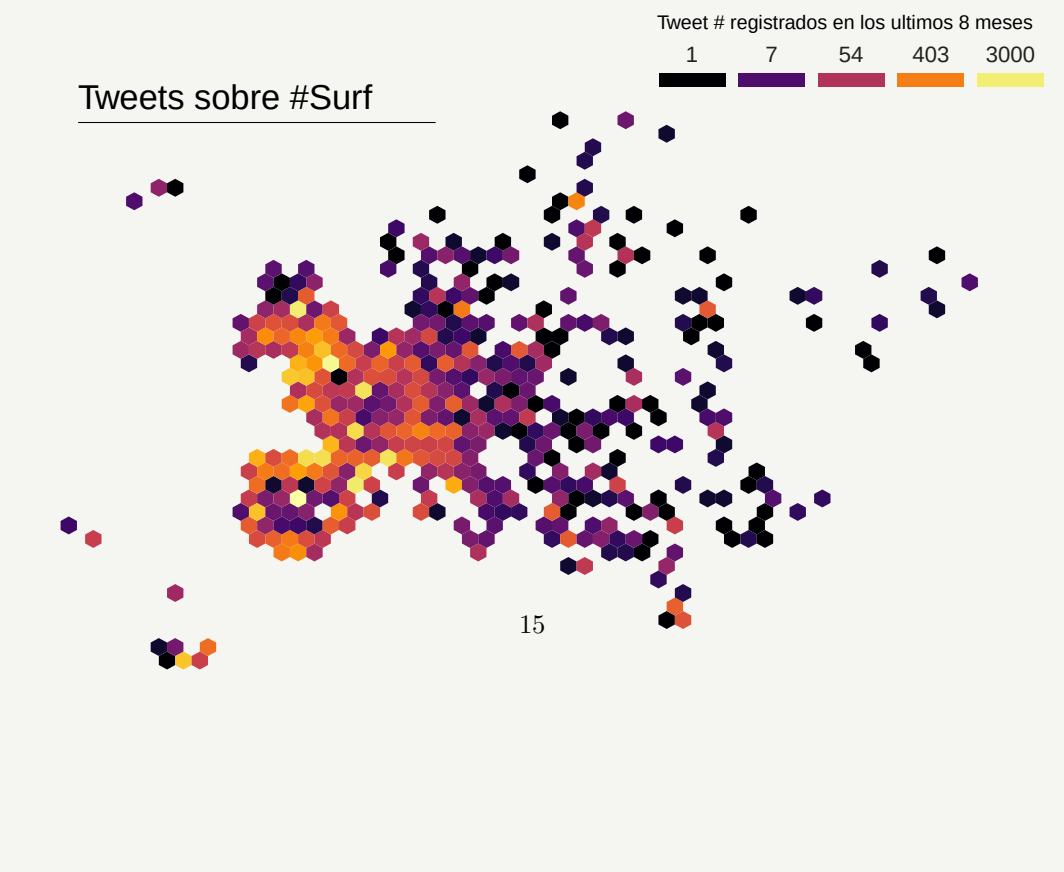
```
> # Load dataset from github
> data <- read.table("https://raw.githubusercontent.com/holtzy/data_to_viz/master/Example_dataset/17_Lis
> head(data)

    homelat    homelon homecontinent
1  18.28548  -70.33012 South America
2  39.10312  -84.51202 North America
3  19.41095  -99.27186 South America
4 -22.90685  -43.17290          <NA>
5 -22.90685  -43.17290          <NA>
6  33.93003 -118.28099 North America
```

Finalmente, mediante la función *ggplot* visualizamos el mapa hexagonal. Para ello, utilizaremos las siguientes funciones:

- *filter*: para filtrar por continente europeo.
- *ggplot*:
    - *aes*: definimos las coordenadas de longitud (*homelon*) como eje X y las coordenadas de latitud (*homelat*) como eje Y.
    - *geom_hex*: para visualizar cada dato con forma de hexágonos.
    - *annotate*: permite realizar añadir elementos concretos sobre la gráfica, como fragmentos de texto (especificado con el parámetro *text*) o segmentos (*segment*), indicando diferentes parámetros como la posición del objeto (x,y); el color, tamaño o el ajuste de texto, entre otros.
    - *theme_void*: para eliminar la cuadrícula y los ejes.
    - *xlim*, *ylim*: para especificar los límites del gráfico.
    - *scale_fill_veridis*: para añadir una leyenda al gráfico
        - *option*: para especificar la paleta de colores (*inferno*, por ejemplo).
        - *trans*: la escala a utilizar para dividir la paleta de colores (por ejemplo, una escala logarítmica o *log*).
        - *breaks*: intervalo de valores en la leyenda.
        - *name*: título de la leyenda
        - *guide_leyend*: especifica parámetros para la leyenda como la posición o la anchura.

- *ggtitle*: para añadir un título al gráfico.
- *theme*: parámetros del gráfico, como el color de fondo, la fuente, el color de la leyenda, el tamaño de letra o el tamaño del título.

```
> data %>%
+   filter(homecontinent=='Europe') %>%
+   ggplot( aes(x=homelon, y=homelat)) +
+     geom_hex(bins=59) +
+     ggplot2::annotate("text", x = -27, y = 72,
+             label="Tweets sobre #Surf", colour = "black", size=5, alpha=1, hjust=0) +
+     ggplot2::annotate("segment", x = -27, xend = 10,
+             y = 70, yend = 70, colour = "black", size=0.2, alpha=1) +
+     theme_void() +
+     xlim(-30, 70) +
+     ylim(0, 72) +
+     scale_fill_viridis(
+       option="inferno",
+       trans = "log",
+       breaks = c(1,7,54,403,3000),
+       name="Tweet # registrados en los ultimos 8 meses",
+       guide = guide_legend( keyheight = unit(2.5, units = "mm"),
+               keywidth=unit(10, units = "mm"), label.position = "top",
+               title.position = 'top', nrow=1)
+     )  +
+     ggtitle( "" ) +
+     theme(
+       legend.position = c(0.8, 1),
+       legend.title=element_text(color="black", size=8),
+       text = element_text(color = "#22211d"),
+       plot.background = element_rect(fill = "#f5f5f2", color = NA),
+       panel.background = element_rect(fill = "#f5f5f2", color = NA),
+       legend.background = element_rect(fill = "#f5f5f2", color = NA),
+       plot.title = element_text(size= 13, hjust=0.1, color = "#4e4d47",
+                   margin = margin(b = -0.1, t = 0.4, l = 2, unit = "cm")),
+     )
>
```



5. Mapa de conexiones

Un mapa de conexiones (*Connection Map*) **muestra las conexiones entre diferentes puntos a lo largo de un mapa**. Para ello, utilizaremos los siguientes paquetes:

- *geosphere*: para la creación de conexiones entre diferentes puntos a lo largo del mapa.
- *maps*: para la visualización del *mapa-mundi*.

En primer lugar, descargamos y añadimos los paquetes:

```
> if(!require(geosphere)){
+   install.packages("geosphere")
+   require(geosphere)
+ }
```

Para visualizar las conexiones entre los diferentes puntos, crearemos una función a la que llamaremos *plot_my_connection*, que tendrá los siguientes parámetros:

- textitdep_lon y *dep_lat*: coordenadas de latitud y longitud del nodo 1.
- textitarr_lon y *arr_lat*: coordenadas de latitud y longitud del nodo 2.

En primer lugar, utilizaremos la función *gcIntermediate*, la cual nos devuelve las **coordenadas de los puntos que conforman la línea que conecta ambos nodos, en forma de matriz**. Con los puntos obtenidos, los almacenamos en un *dataframe*. Finalmente, mediante la función *lines* creamos una línea con los puntos obtenidos:

```
> # Funcion para crear una conexion entre dos nodos
> plot_my_connection=function(dep_lon, dep_lat, arr_lon, arr_lat, ...){
+     inter <- gcIntermediate(c(dep_lon, dep_lat), c(arr_lon, arr_lat), n=50,
+     addStartEnd=TRUE, breakAtDateLine=F)
+     inter=data.frame(inter)
+     diff_of_lon=abs(dep_lon) + abs(arr_lon)
+     if(diff_of_lon > 180){
+         lines(subset(inter, lon>=0), ...)
+         lines(subset(inter, lon<0), ...)
+     }else{
+         lines(inter, ...)
+         }
+     }
```

Una vez creada la función, vamos a realizar una conexión (a modo de prueba) entre las siguientes ciudades:

- **Buenos Aires**
- **París**
- **Melbourne**
- **San Petersburgo**
- **Abidjan**
- **Montreal**
- **Nairobi**
- **Salvador**

```
> # En primer lugar, creamos un dataset con las coordenadas
> # de latitud y longitud de cada una de las ciudades.
> data <- rbind(
+     Buenos_aires=c(-58,-34),
+     Paris=c(2,49),
+     Melbourne=c(145,-38),
+     Saint.Petersburg=c(30.32, 59.93),
+     Abidjan=c(-4.03, 5.33),
+     Montreal=c(-73.57, 45.52),
+     Nairobi=c(36.82, -1.29),
+     Salvador=c(-38.5, -12.97)
+     )  %>% as.data.frame()
> colnames(data)=c("long","lat")
> # A continuacion, generamos parejas de nodos
> all_pairs <- cbind(t(combn(data$long, 2)), t(combn(data$lat, 2))) %>% as.data.frame()
> colnames(all_pairs) <- c("long1","long2","lat1","lat2")
> # Mediante la funcion map representamos la plantilla
> # con el mapa-mundi
> par(mar=c(0,0,0,0))
> # world = mapa-mundi
> maps::map('world',col="#d1d1d1", fill=TRUE, bg="white",
+ lwd=0.05,mar=rep(0,4),border=0, ylim=c(-80,80))
> # Por cada pareja de nodos, representamos graficamente
> # las conexiones entre los nodos mediante lineas
> for(i in 1:nrow(all_pairs)){
+     plot_my_connection(all_pairs$long1[i], all_pairs$lat1[i], all_pairs$long2[i],
+     all_pairs$lat2[i], col="skyblue", lwd=1)
+     }
> # Finalmente, representamos cada nodo mediante la funcion points
> points(x=data$long, y=data$lat, col="slateblue", cex=2, pch=20)
> text(rownames(data), x=data$long, y=data$lat,  col="slateblue", cex=1, pos=4)
```