

# Visualización y geolocalización de datos

## Máster en Data Science & Business Analytics (con R Software)

Gema Fernández-Avilés      Diego Hernangómez

2022-01-20

## Índice general

<b>1 La revolución de los geodatos</b>	<b>2</b>
<b>2 Datos geográficos</b>	<b>2</b>
2.1 Contexto general . . . . .	2
2.2 Conceptos clave . . . . .	4
<b>3 Formatos de datos espaciales</b>	<b>6</b>
3.1 Tipos de ficheros . . . . .	6
3.2 Datos de vectores . . . . .	7
3.3 Datos raster . . . . .	9
3.4 Sistema de Referencia de Coordenadas (CRS) . . . . .	11
<b>4 Estadística espacial</b>	<b>28</b>
4.1 Antes de continuar... dependencia espacial. . . . .	29
4.2 Datos espaciales . . . . .	30
4.3 Clasificación de datos espaciales . . . . .	32
<b>5 Aplicaciones</b>	<b>33</b>
5.1 Temperaturas mínimas en España . . . . .	33
5.2 Renta media por municipios . . . . .	44
5.3 Análisis de patrones de puntos . . . . .	55
<b>6 Extensiones</b>	<b>62</b>
<b>Referencias</b>	<b>62</b>
<b>Objetivos de aprendizaje</b>	
<b>¿Por dónde empezamos? Recursos interesantes</b>	
Libros de referencia:	

- Spatial Data Science with applications in R
- Geocomputation with R
- Displaying time series, spatial and space-time data with R

Recursos de estadística espacial en R:

- r spatial
- R-spatial

Otros recursos web interesantes, esto es para nosotros:

- Libro de Ruben
- Rositter
- Libro Hengl
- Libro de Páez

## 1 La revolución de los geodatos

Que estamos en la la *era del dato*, que los *datos son el petróleo del siglo XXI* y que estamos rodeados de datos es una cuestión que ya hemos hecho inherente a nosotros. Vivimos en el momento del dato, donde la profesión de *Data Scientist* se ha convertido en la **profesión más sexy del siglo XXI** según vaticinó en 2012 Harvard Business Review. Cada segundo se producen 1,7 MB de datos/persona y cada año esta cifra se duplica se duplica.

Este incremento exponencial de los datos ha sido posible, sin duda, gracias al desarrollo de la tecnología, la informática, los ordenadores, los teléfonos móviles móviles, los satélites, internet, etc... y asociado a estas nuevas herramientas, se ha producido una lluvia sin precedentes hasta el momento de **datos espaciales o datos georreferenciados**. Cada teléfono inteligente tiene un **Receptor de posicionamiento Global** (en inglés, *Global Positioning System, GPS*) y una multitud de sensores en dispositivos que van desde satélites y vehículos semi-autónomos hasta científicos ciudadanos que miden incesantemente cada parte del mundo. La tasa de datos producidos es abrumadora. Un vehículo autónomo, por ejemplo, puede generar 100 GB de datos por día (The Economist, 2016).

Esta **revolución de los geodatos** y el **análisis de los datos espaciales** junto con los **Sistemas de Infomación Geográficos** (habitualmente expresados como **GIS** por las siglas de su nombre en inglés *Geographical Information System*) no sólo han impulsado la demanda de hardware informático de alto rendimiento y software escalable y eficiente para manejar y extraer la información, lo que se conoce como **Geocomputación**, sino que han dado lugar una nueva rama de conocimiento, **Ciencia de Datos Espaciales** comúnmente conocida como *Spatial Data Scicene* (SDS).

Como ejemplo, de este abrumador desarrollo de datos georreferenciados, el Ministerio de Transportes, Movilidad y Agenda Urbana llevó a cabo durante los años 2020 y 2021 el denominado Estudio de movilidad con Big Data, cuya fuente principal de datos fue el posicionamiento de los teléfonos móviles anonimizado. Estos datos permiten, por ejemplo, analizar la movilidad entre diversas zonas del territorio español de manera diaria (véase Fig 1) Este tipo de análisis era impensable hace tan solo unos años.

## 2 Datos geográficos

### 2.1 Contexto general

La palabra geográfico puede dividirse en **geo** (tierra) + **gráfico** (dibujo/mapa). Por tanto, los datos geográficos contienen información de cualquier variable referenciada en un punto/área de la superficie terrestre

## Flujo de viajeros entre aeropuertos nacionales

Últimos domingos de mes

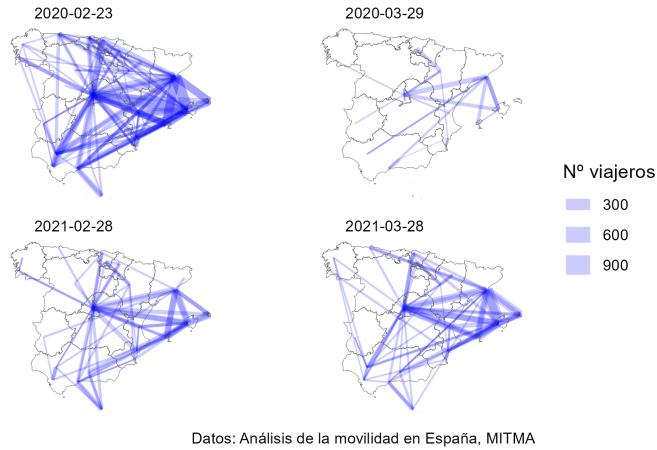


Figura 1: Análisis de movilidad COVID

y pueden representarse en mapas. El desarrollo de los datos geográficos ha producido grandes bases de datos espaciales y, a su vez, ha propiciado el desarrollo de herramientas para su tratamiento como los ya mencionados Sistemas de información geográficos y la Geocomputación.

### ¿Qué hace un Sistemas de información geográfico?

Un Sistema de información geográfica (SIG) es una herramienta que crea, administra, analiza y mapea todo tipo de datos. GIS conecta datos a un mapa, integrando datos de ubicación (dónde están las cosas) con todo tipo de información descriptiva (cómo son las cosas allí).

Esto proporciona una base para el mapeo y el análisis que se utiliza en la ciencia y en casi todas las industrias. GIS ayuda a los usuarios a comprender patrones, relaciones y contexto geográfico. Los beneficios incluyen una mejor comunicación y eficiencia, así como una mejor gestión y toma de decisiones.

La figura 2 muestra el flujo de trabajo de los Sistemas de Información Geográfica, que va desde (i) la elaboración de mapas, (ii) la obtención de geodatos o datos espaciales, (iii) el análisis de los datos geográficamente referenciados y (iv) la edición, mapeo y presentación de los resultados.

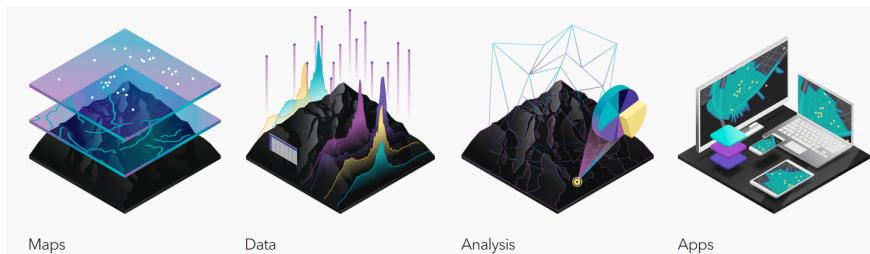


Figura 2: Flujo de trabajo de los GIS. Fuente: <https://www.esri.com/en-us/what-is-gis/overview>

Pero es más, el desarrollo de la **Inteligencia Artificial** y la **Inteligencia computacional**, han hecho que éstas se conviertan en herramientas creativas y complemenariás a los convencionales GIS, dando origen a la **Geocomputación**, que trata de utilizar el poder de los ordenadores para *hacer cosas* con los datos geográficos.

### ¿Y qué es la Geocomputación?

En primer lugar, señalar que, aunque la geocomputación es un término relativamente nuevo se encuentra influenciado por otros términos clásicos. De manera sencilla puede definirse como “*el proceso de aplicar tecnologías de computación a problemas geográficos*” (Rees, 1998). Abrahart, Openshaw, Abrahart, & See (2000) aporta más elementos formales a esta definición destacando que “*la geocomputación trata sobre los diferentes tipos de geodatos, y sobre el desarrollo de geo-herramientas relevantes en un contexto científico*”.

La geocomputación está muy relacionada con otros términos como los Sistema de información geográfica (GIS, del inglés, Geographic Information Systems), y con diversos tipos de campos científicos, como las Geociencias, las Ciencias atmosféricas y climáticas, la Geoinformática, la Topología, la Ecología y las Ciencia de datos geográficos (GDS, Geographic Data Science).

Cada término comparte un énfasis en un enfoque **científico** (que implica reproducible y falsable) influenciado por los GIS, aunque sus orígenes y principales campos de aplicación difieren. La geocomputación es ampliamente utilizada en ámbitos como la sociología, análisis político o el desarrollo de aplicaciones para móviles. Por tanto, usamos geocomputación como un sinónimo aproximado que encapsula a todas las ciencias que buscan usar datos geográficos para trabajos científicos aplicados.

En resumen, la geocomputación trata de aplicar herramientas y técnicas de análisis y estudio de datos a un tipo de datos específicos: los datos espaciales.

### ¿Por qué R para datos geográficos?

R es una herramienta con capacidades avanzadas de análisis, modelado y visualización. Por ejemplo, los nuevos entornos de desarrollo integrado (IDE), como RStudio, han hecho que R sea más fácil de usar para muchos, facilitando la creación de mapas con un panel dedicado a la visualización interactiva (Lovelace, Nowosad, & Muenchow, 2019). Además, el uso del código R, permite la enseñanza de la geocomputación con referencia a ejemplos reproducibles en lugar de conceptos abstractos. Por ejemplo, de una forma relativamente sencilla, se puede geoposicionar de manera interactiva la localización de la Puerta del Sol en Madrid y, además, dejar la el código R para hacerlo reproducible, ver Figura 3.

```
library(leaflet)
leaflet(width = "100%", height = "500px") %>%
  addTiles() %>%
  setView(-3.703548, 40.417147, zoom = 60)
```

Por otra parte R dispone de cientos de librerías especializadas para datos espaciales. Una descripción detallada puede ver se en CRAN Task View: Analysis of Spatial Data

Para no abrumar al lector, a continuación se muestran, de manera esquemática, las librerías más usadas para el tratamiento de datos espaciales y que se emplearán a lo largo de la asignatura Estadística Espacial y Espacio-Temporal, no sólo en el tema que nos ocupa:

- **sp y sf**: para el tratamiento de clases y métodos de los datos vectoriales.
- **raster,terra y stars** para datos raster.
- **gstat y geoR**: para el análisis de datos geoestadísticos, ajuste y estimación de semivariogramas, interpretación, etc.
- **spdep** para el análisis de datos con econometría espacial, creación de matrices de contiguidad/distancia **W**, estimación de modelos econométricos espaciales.
- **spatstat** para el análisis de procesos de puntos espaciales.

## 2.2 Conceptos clave

Una vez visto el contexto actual de los datos georreferenciados y antes de entrar en detalle en su análisis, debemos tener en cuenta una serie de conceptos clave que se irán desarrollando a lo largo del tema.



Figura 3: Localización interactiva de la Puerta del Sol en Madrid

Hemos dicho que Geográfico = Geo (tierra) + gráfico (mapa). Por tanto, si tenemos varios datos geográficos, localizados en distintos puntos de la tierra, es porque tenemos las **coordenadas** que los posicionan en esos puntos concretos. Asociado a estas coordenadas debemos conocer el **Sistema de referencia de espacial** o Coordinate reference system (CRS) en el que están proyectadas dichas coordenadas.

Por otra parte, los formatos de estos datos pueden ser **vectores** o **raster** como se explicará en la siguiente sección.

Si damos un paso más e incorporamos el concepto de **distancia**, pues es lógico pensar que en un fenómeno de interés, por ejemplo, en la modelización de la cantidad y dirección de lava en La Palma tras la erupción del volcán “Cumbre Vieja” la distancia es un factor clave, pues aquellas zonas más cercanas al volcán tendrán niveles más parecidos entre sí y con valores más altos.

En este caso el nivel de contaminación en el aire en La Palma no puede ser modelado como si las observaciones fuesen independientes pues las más cercanas entre sí serán más parecidas que las más lejanas, dando lugar al concepto de **dependencia espacial**. Y depende del tipo de datos espaciales tendremos tres grandes formas de abordar el tratamiento de los datos espaciales: **geoestadística**, **procesos de punto** y **econometría espacial** (véase sección

*Sistema de Referencia de Coordenadas (CRS)*

).

**si nos gusta poner la imagen más pequeña**

nos gusta

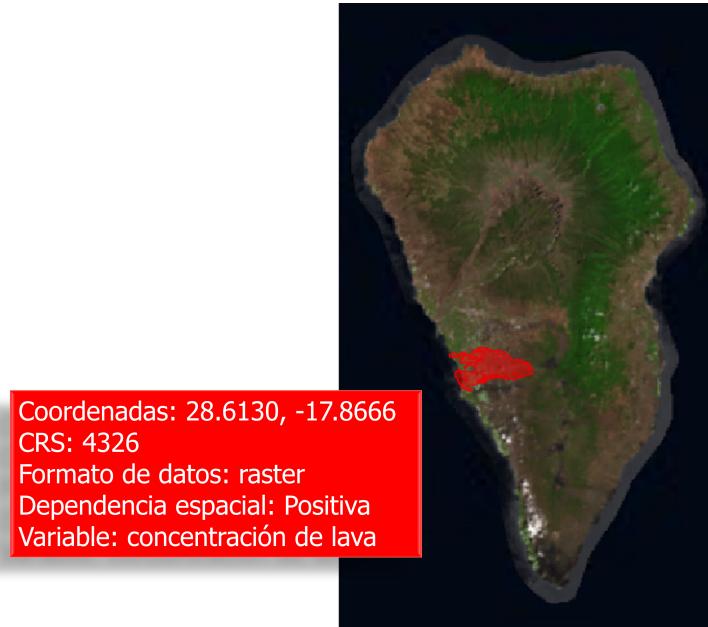


Figura 4: Información espacial de la concentración de lava en Cumbre Vieja

### 3 Formatos de datos espaciales

#### 3.1 Tipos de ficheros

En el ámbito del análisis espacial en **R**, se pueden clasificar los **datos** o **el formato?**, **pregunto sólo** espaciales en función del modelo de datos (Lovelace et al., 2019). Se pueden distinguir dos tipos de modelos de datos:

## 3.2 Datos de vectores

Este modelo está basado en puntos georeferenciados. Los **puntos** pueden representar localizaciones específicas, como la localización de edificios:

```
library(ggplot2)
library(sf)

# Hospitales en Toledo segun Eurostat
hosp_toledo <- st_read("data/hosp_toledo.geojson", quiet = TRUE)

# Plot
ggplot() +
  geom_sf(
    data = hosp_toledo, aes(fill = "Centros Sanitarios"),
    color = "blue"
  ) +
  labs(
    caption = "Datos: Eurostat",
    title = "Hospitales y Centros de Salud en Toledo",
    fill = ""
  ) +
  theme_minimal() +
  theme(legend.position = "bottom")
```

Hospitales y Centros de Salud en Toledo

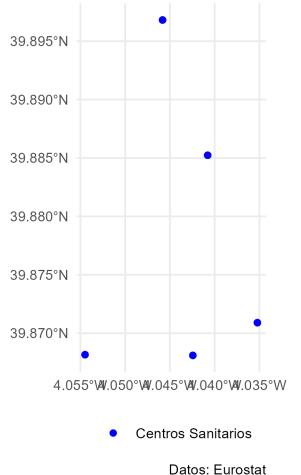


Figura 5: Datos vector: Puntos

Estos puntos también pueden estar conectados entre sí, de manera que formen geometrías más complejas, como **líneas** y **polígonos**:

```
tajo <- st_read("data/tajo_toledo.shp", quiet = TRUE)
toledo <- st_read("data/toledo_ciudad.gpkg", quiet = TRUE)
```

```

ggplot(toledo) +
  geom_sf(fill = "cornsilk2") +
  geom_sf(data = tajo, col = "lightblue2", lwd = 2, alpha = 0.7) +
  geom_sf(data = hosp_toledo, col = "blue") +
  coord_sf(
    xlim = c(-4.2, -3.8),
    ylim = c(39.8, 39.95)
  ) +
  theme_minimal()

```

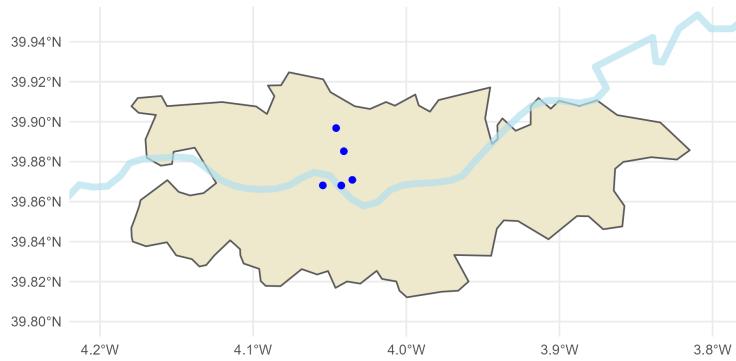


Figura 6: Datos vector: Puntos, líneas y polígonos

En la Fig. 6 , el río Tajo está representado como una línea (sucesión de puntos unidos entre sí) y la ciudad de Toledo como un polígono (línea de puntos cerrada formando un continuo). A modo ilustrativo, la Fig. 7 representa la descomposición en puntos de todos los datos espaciales representados en la Fig. 6.

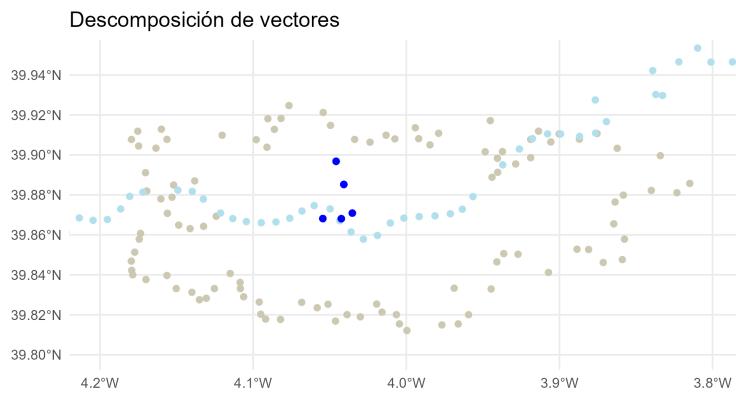


Figura 7: Datos vector: Descomposición en puntos

### 3.3 Datos raster

Los datos ráster son datos representados en una rejilla rectangular de píxeles (denominada **matriz**) que se puede visualizar en diversos dispositivo de representación. El caso más cotidiano de un ráster es una fotografía, donde la imagen se representa como una serie de celdas, determinadas por la resolución de la imagen (número total de píxeles, determinados como número de píxeles en cada fila por número de píxeles en cada columna) y el color que presenta cada uno de estos píxeles.

En el ámbito de los datos espaciales, la definición es muy similar. Un archivo ráster está formado por una malla regular de píxeles georreferenciada, tal y como muestra la Fig. 8:

```
library(raster)

elev <- raster("data/Toledo_DEM.tif")
plot(elev, main = "Elevación de la provincia de Toledo")

# Mostramos el grid
polys <- rasterToPolygons(elev)
plot(polys, add = TRUE, border = "grey90")

# Añadimos la provincia
Tol_prov <- st_read("data/Toledo_prov.gpkg", quiet = TRUE)

# Si queremos solamente la forma en sf, usamos st_geometry
plot(st_geometry(Tol_prov), add = TRUE)
```

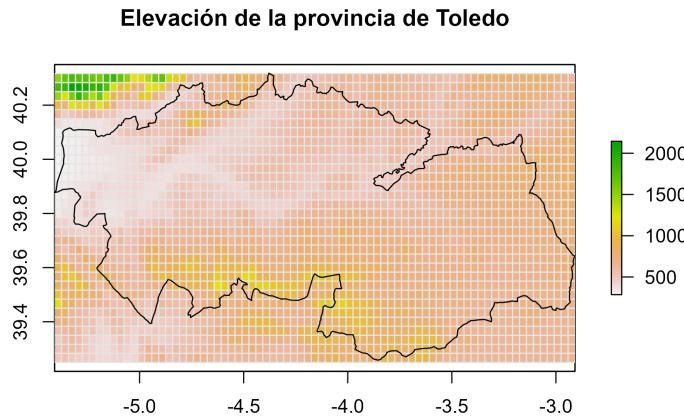


Figura 8: Datos ráster

En la Fig. 8, el objeto ráster `elev` tiene únicamente una capa (denominada `ESP_alt`). Eso implica que cada píxel tiene asociado un único valor, en este caso, en este caso la altitud media del terreno observada:

Los rásters pueden contener varias capas (o layers), de manera que cada píxel puede tener asociados varios valores. Volviendo al ejemplo de la fotografía, en un modelo simple de color RGB cada píxel lleva asociado 3 valores (rojo, verde o azul), de manera que al combinar las tres capas se puede definir un color distinto en cada píxel.

En la Fig. 10 vamos a usar una imagen de mapa georreferenciada, como las proporcionadas por servicios de mapas online, para analizar su composición.

Cuadro 1: Datos de un ráster (detalle)

x	y	Toledo_DEM
-5.391667	40.3	1498.312
-5.358333	40.3	1701.125
-5.325000	40.3	1825.312
-5.291667	40.3	1739.062
-5.258333	40.3	1756.062
-5.225000	40.3	1659.688
-5.191667	40.3	1607.375
-5.158333	40.3	1809.562
-5.125000	40.3	1874.625
-5.091667	40.3	1691.312
-5.058333	40.3	1511.500
-5.025000	40.3	1207.000
-4.991667	40.3	1160.125
-4.958333	40.3	1396.125
-4.925000	40.3	1624.125

Detalle de los primeros 15 pixels

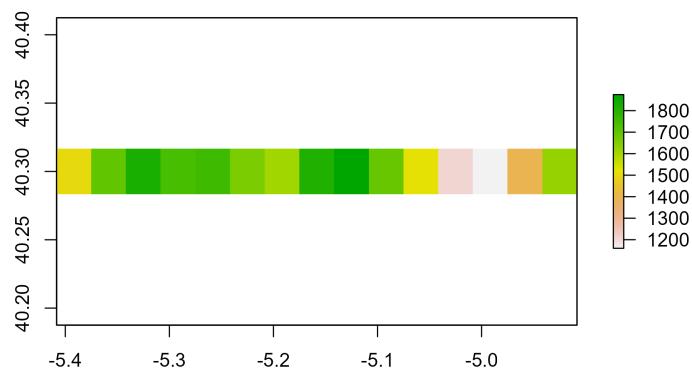


Figura 9: Datos ráster: Detalle

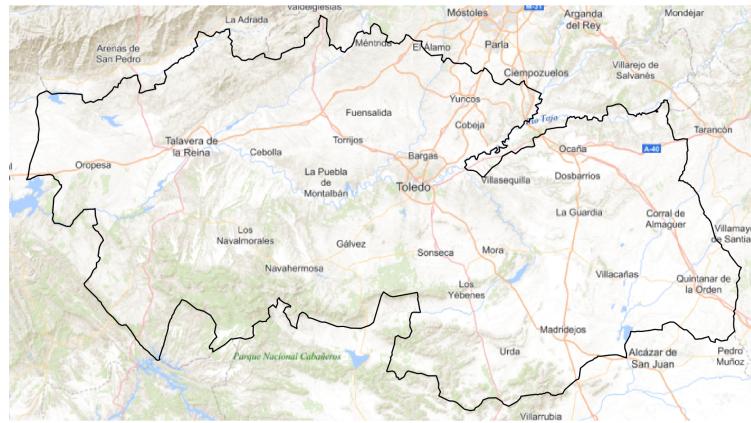


Figura 10: Datos ráster con varias bandas

El ráster se puede descomponer en las tres capas RGB mencionadas anteriormente:



Figura 11: Datos ráster multicapa: Descomposición

### 3.4 Sistema de Referencia de Coordenadas (CRS)

Un sistema de referencia de coordenadas (o CRS por sus siglas en inglés, **Coordinate Reference System**) permite relacionar datos espaciales con su localización en la superficie terrestre. Los CRS constituyen por tanto un aspecto fundamental en el análisis y representación de datos espaciales, ya que nos permiten identificar con exactitud la posición de los datos sobre el globo terráqueo.

Así mismo, cuando se trabaja con datos espaciales provenientes de distintas fuentes de información, es necesario comprobar que dichos datos se encuentran definidos en el mismo CRS:

En la Fig. 13, ambos puntos (verde y rojo) tienen los mismos valores de coordenadas en los ejes X e Y, en este caso las correspondientes a la ciudad de Toledo.

Cuadro 2: Datos de un ráster multicapa (detalle)

x	y	lyr.1	lyr.2	lyr.3
-5.466412	40.34418	215.2128	208.1061	190.5410
-5.463875	40.34418	228.0369	223.1854	211.2115
-5.461338	40.34418	229.3495	224.3414	213.4325
-5.458800	40.34418	215.8592	208.8660	191.2922
-5.456263	40.34418	219.2696	212.8231	196.6812
-5.453725	40.34418	235.0954	231.4222	222.4115
-5.451188	40.34418	240.3514	237.9094	231.4736
-5.448651	40.34418	237.2358	233.7561	226.2005
-5.446113	40.34418	229.9570	225.3262	214.6201
-5.443576	40.34418	226.7812	221.6796	209.2929
-5.441038	40.34418	222.3593	216.5022	202.0188
-5.438501	40.34418	220.9312	214.9060	200.0306
-5.435964	40.34418	224.7755	219.2661	206.2156
-5.433426	40.34418	222.0479	216.0124	201.6103
-5.430889	40.34418	225.0516	219.8074	207.0263

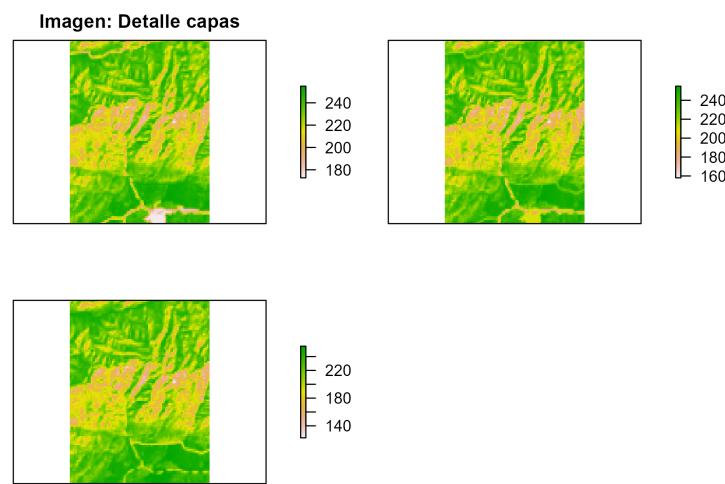


Figura 12: Datos ráster multicapa: Descomposición

### Ejemplo: Mismas coordenadas en distintos CRS

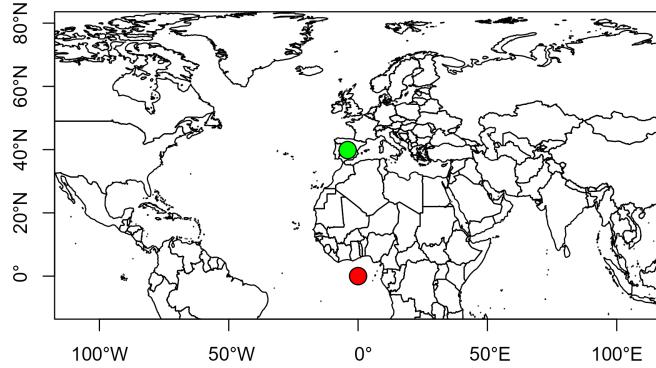


Figura 13: Representación de mismos valores de coordenadas en distintos CRS

Sin embargo, presentan distintos CRS. Por este motivo, al representar ambos puntos en un mapa, se observa que no se están refiriendo a la misma localización geográfica. Esto es así porque el CRS define la referencia (punto  $x=0$  e  $y =0$ ) y las unidades de los ejes (grados, metros, millas).

Como conclusión, **además de disponer de las coordenadas de los datos espaciales, es necesario conocer el CRS en el que están definidos para conocer de manera exacta su localización geográfica**. Además, nótese que para cualquier **análisis de datos espaciales** es necesario que todos los geodatos **se encuentren referenciados en el mismo CRS**. Esto se consigue transformando (o proyectando) los datos a un CRS común, nunca sobreescribiendo el CRS de los mismos.

#### 3.4.1 Tipos de CRS

A continuación se definen los dos grandes tipos de CRS, los CRS geográficos y los CRS proyectados.

**3.4.1.1 CRS geográficos** Los CRS geográficos son aquellos en los que los parámetros empleados para localizar una posición espacial son la latitud y la longitud:

- **Latitud:** Es la distancia angular expresada en grados sobre el plano definido por el ecuador terrestre. Determina la posición sobre de una localización en el eje Norte-Sur de la Tierra y toma valores en el rango  $[-90, 90]$ . Las líneas imaginarias determinadas por una sucesión de puntos con la misma latitud a lo largo del eje Este-Oeste se denominan **paralelos**. (Ver Fig. 14)
- **Longitud:** Es la distancia angular expresada en grados sobre el plano definido por el meridiano de Greenwich. Determina la posición sobre de una localización en el eje Este-Oeste de la Tierra y toma valores en el rango  $[-180, 180]$ . Las líneas imaginarias determinadas por una sucesión de puntos con la misma longitud a lo largo del eje Este-Oeste se denominan **meridianos** (Ver Fig. 14)

Es muy importante destacar que en un sistema de coordenadas geográfico, es decir, basado en latitudes y longitudes, las **distancias** entre dos puntos representan **distancias angulares**. Por ejemplo, la distancia entre el meridiano de Greenwich y el meridiano correspondiente a la longitud  $20^\circ$  siempre es de  $+20^\circ$ . Sin embargo, debido a la forma esférica de la Tierra, la longitud en metros entre ambos meridianos no es constante:

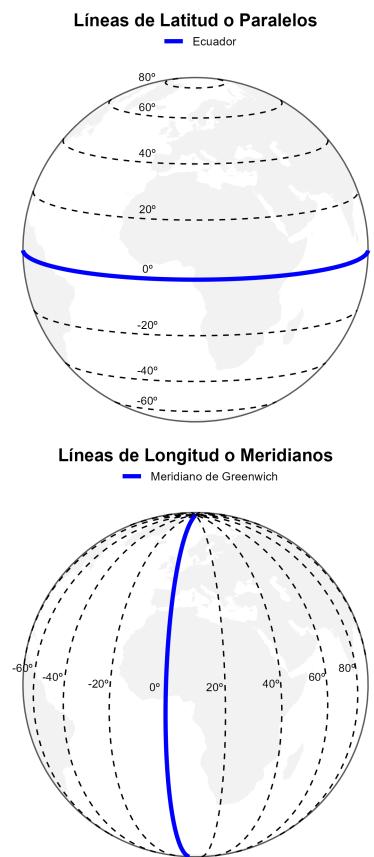


Figura 14: Paralelos y Meridianos terrestres

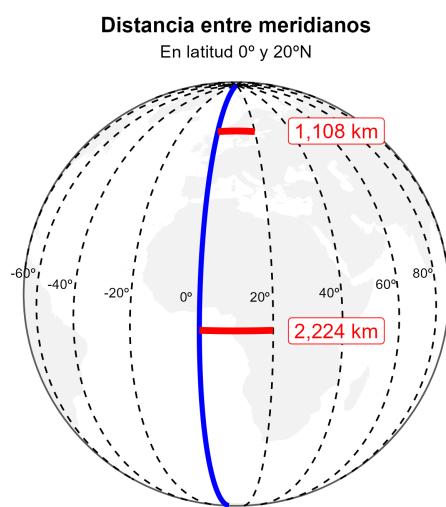


Figura 15: Distancia entre meridianos en distintas latitudes

**3.4.1.2 CRS proyectados** La representación de formas tridimensionales en un soporte plano (dos dimensiones) presenta algunos retos. Por ello, es habitual trabajar con proyecciones de mapas.

Una **proyección geográfica** es un método para reducir la superficie de la esfera terrestre a un sistema cartesiano de dos dimensiones. Para ello, es necesario transformar las coordenadas longitud y latitud en coordenadas cartesianas x e y.

Es importante destacar que las proyecciones pueden incluir un punto de origen ( $X=0, Y=0$ ) y unas unidades de distancia (habitualmente metros) específicas. Por ejemplo, la **proyección cónica equiáreas de Albers** (específica para Estados Unidos) define su punto de referencia (0,0) en la latitud  $40^{\circ}$  N y longitud  $96^{\circ}$ , y la unidad de variación están definida en metros. De ahí la importancia de conocer el CRS de los datos geográficos, como se expuso al principio de este tema.

Existen varias familias de proyecciones, que se pueden clasificar de diversas maneras: por tipo de superficie de proyección y por métrica a preservar.

#### a) Por tipo de superficie de proyección

El proceso de trasladar puntos de una esfera a un plano puede plantearse de manera práctica como el ejercicio de envolver una esfera con una superficie plana (como una hoja de papel) y trasladar los puntos de la esfera de manera lineal al punto de la superficie plana más cercano a ella. La Fig. 16 muestra estos tres tipos de proyección.

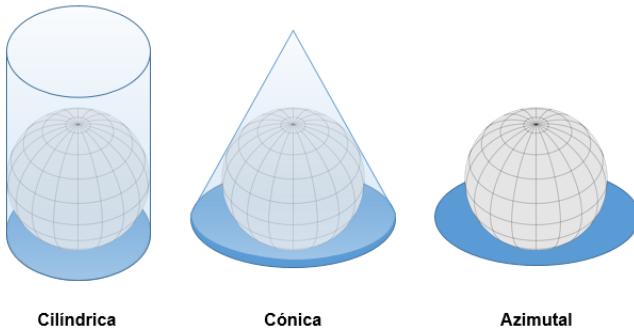


Figura 16: Tipos de proyección por superficie de proyección

A partir de este ejercicio, se plantean tres posibles soluciones, dependiendo del tipo de superficie que se use para proyectar.

- **Proyecciones cilíndricas:** Son aquellas proyecciones donde la superficie de proyección conforma un cilindro alrededor de la Tierra. Una de las proyecciones cilíndricas más conocidas es la **proyección de Mercator** (Ver Fig. 17).
- **Proyecciones cónicas:** En este tipo de proyecciones, se plantea la superficie de proyección como una forma cónica. Como ejemplo, la **proyección cónica equiáreas de Albers** es una de las proyecciones que más suele usarse en la representación de mapas de América del Norte (Ver Fig. 18).
- **Proyecciones acimutales o planares:** En este tipo de proyección se proyecta una porción de la Tierra sobre un plano que es tangente a la misma en el punto de referencia. Como ejemplos de proyecciones acimutales podemos destacar la **proyección ortográfica** (Ver Fig. 19).

#### b) Por métrica a preservar



Figura 17: Proyección Mercator



Figura 18: Proyección cónica equiáreas de Albers



Figura 19: Proyección ortogonal

Es importante tener en cuenta que cualquier proyección de la superficie de la Tierra produce distorsiones en una o varias características geográficas. Como ejemplos clásicos, la proyección de Mercator produce distorsiones del **tamaño** especialmente en aquellas regiones más cercanas a los polos (Groenlandia, que la proyección de Mercator presenta una área similar a la de África, tiene menor superficie real que Argelia). Otras de las métricas que suele verse distorsionada son la **distancia** entre dos puntos geográficos, la **dirección** o la **forma** de regiones de la Tierra.

A lo largo de la Historia se han desarrollado diversas proyecciones cuyo objetivo es preservar alguna o varias de las propiedades mencionadas anteriormente, sin embargo es importante destacar que **no existe una proyección que sea capaz de preservar todas las métricas a la vez**.

Según la métrica a presevar, las proyecciones se pueden clasificar en:

- **Proyecciones conformales:** Intentan preservar los **ángulos** que se forman en la superficie terrestre. Por ejemplo, la proyección de Mercator representa ángulos rectos en las intersecciones de los paralelos y los meridianos (Ver Fig. 20).

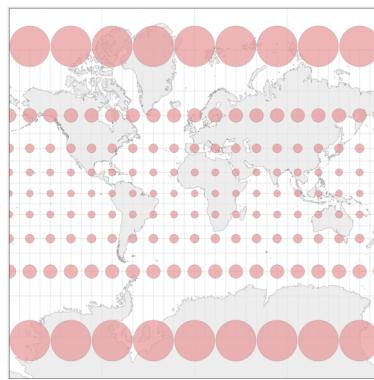


Figura 20: Ejemplo de proyección conformal: Mercator

- **Proyecciones equivalentes:** Preservan las proporciones de las **áreas**, provocando a su vez deformaciones en el resto de características, como la forma o los ángulos. La proyección acimutal equivalente de Lambers es un tipo de proyección equivalente (Ver Fig. 21).

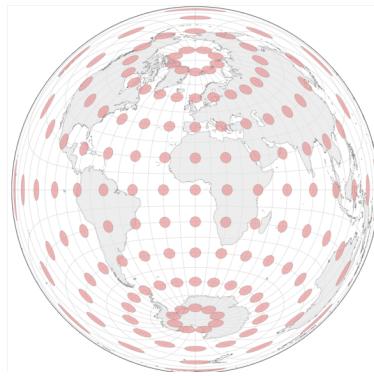


Figura 21: Ejemplo de proyección equivalente: Proyección acimutal equivalente de Lambers

- **Proyecciones equidistantes:** Preservan la **distancia** entre dos puntos geográficos específicos. Por ejemplo, la proyección Plate carré preserva la distancia entre el Polo Norte y el Polo Sur (Ver Fig. 22).

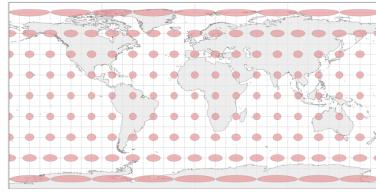


Figura 22: Ejemplo de proyección equidistante: Platé carre

- **Proyecciones de compromiso:** No intentan preservar ninguna métrica en concreto. En su lugar, se centran en intentar encontrar un **equilibrio** entre las diversas distorsiones que provocan para intentar dar una representación más o menos representativa de la superficie terrestre. La proyección de Winkel Tripel, usada en los mapas de National Geographic, es un ejemplo de proyección de compromiso (Ver Fig. 23).

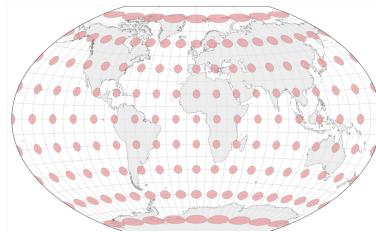


Figura 23: Ejemplo de proyección de compromiso: Winkel Tripel

En los anteriores ejemplos se ha añadido a cada proyección la **indicatriz de Tissot**. Ésta consiste en una serie de círculos imaginarios de igual área distribuidos sobre la superficie esférica de la Tierra en determinados puntos. De este manera, al presentar la indicatriz de Tissot en una proyección específica, se puede entender de una manera intuitiva la distorsión provocada por dicha proyección, ya que los círculos se ven distorsionados o preservados según los parámetros y la naturaleza de la proyección en cuestión.

### 3.4.2 Trabajando con proyecciones en R

Existe toda una serie de proyecciones predefinidas, identificadas mediante los **códigos EPSG, ESRI, WKT** o proj4 (en desuso en R, pero todavía admitidos). Existen varios recursos web donde se pueden consultar y seleccionar los códigos correspondientes:

- <https://epsg.io/>
- <https://spatialreference.org/>
- <https://proj.org/operations/projections/index.html>

Algunos de los códigos de proyecciones que es fundamental conocer son:

- **EPSG: 4326:** Proyección correspondiente a WGS 84, que es el sistema usado por los sistemas GPS. Cuando trabajemos con coordenadas geográficas longitud/latitud, este es habitualmente el CRS de referencia.
- **EPSG: 3857:** Código correspondiente a la proyección de Mercator, usada habitualmente por servicios como Google Maps, etc.

En la sección 3.4.3 veremos cómo encontrar un CRS usando el paquete `crssuggest`.

El paquete `sf` permite obtener los parámetros de cualquier proyección mediante la función `st_crs()`:

```
library(sf)

# Ejemplo: EPSG WGS 84 (Sistema Global GPS): EPSG 4326

st_crs(4326)
#> Coordinate Reference System:
#>   User input: EPSG:4326
#>   wkt:
#> GEOGCRS["WGS 84",
#>   DATUM["World Geodetic System 1984",
#>       ELLIPSOID["WGS 84",6378137,298.257223563,
#>           LENGTHUNIT["metre",1]],
#>       PRIMEM["Greenwich",0,
#>           ANGLEUNIT["degree",0.0174532925199433]],
#>       CS[ellipsoidal,2],
#>           AXIS["geodetic latitude (Lat)",north,
#>               ORDER[1],
#>                   ANGLEUNIT["degree",0.0174532925199433]],
#>           AXIS["geodetic longitude (Lon)",east,
#>               ORDER[2],
#>                   ANGLEUNIT["degree",0.0174532925199433]],
#>       USAGE[
#>           SCOPE["Horizontal component of 3D system."],
#>           AREA["World."],
#>           BBOX[-90,-180,90,180]],
#>           ID["EPSG",4326]

# Usando código ESRI North America Albers Equal Area Conic

st_crs("ESRI:102008")
#> Coordinate Reference System:
#>   User input: ESRI:102008
#>   wkt:
#> PROJCRS["North_America_Albers_Equal_Area_Conic",
#>   BASEGEOGCRS["NAD83",
#>       DATUM["North American Datum 1983",
#>           ELLIPSOID["GRS 1980",6378137,298.257222101,
#>               LENGTHUNIT["metre",1]],
#>               PRIMEM["Greenwich",0,
#>                   ANGLEUNIT["Degree",0.0174532925199433]]],
#>   CONVERSION["North_America_Albers_Equal_Area_Conic",
#>       METHOD["Albers Equal Area",
#>           ID["EPSG",9822]],
#>       PARAMETER["Latitude of false origin",40,
#>           ANGLEUNIT["Degree",0.0174532925199433],
#>           ID["EPSG",8821]],
#>       PARAMETER["Longitude of false origin",-96,
#>           ANGLEUNIT["Degree",0.0174532925199433],
#>           ID["EPSG",8822]],
#>       PARAMETER["Latitude of 1st standard parallel",20,
```

```

#>           ANGLEUNIT["Degree",0.0174532925199433],
#>           ID["EPSG",8823]],
#>           PARAMETER["Latitude of 2nd standard parallel",60,
#>           ANGLEUNIT["Degree",0.0174532925199433],
#>           ID["EPSG",8824]],
#>           PARAMETER["Easting at false origin",0,
#>           LENGTHUNIT["metre",1],
#>           ID["EPSG",8826]],
#>           PARAMETER["Northing at false origin",0,
#>           LENGTHUNIT["metre",1],
#>           ID["EPSG",8827]]],
#>           CS[Cartesian,2],
#>           AXIS["(E)",east,
#>           ORDER[1],
#>           LENGTHUNIT["metre",1]],
#>           AXIS["(N)",north,
#>           ORDER[2],
#>           LENGTHUNIT["metre",1]],
#>           USAGE[
#>               SCOPE["Not known."],
#>               AREA["North America - onshore and offshore: Canada - Alberta; British Columbia; Manitoba; Ne
#>               BBOX[23.81,-172.54,86.46,-47.74]],
#>               ID["ESRI",102008]]]

# Usando proj4string: Robinson

st_crs("+proj=robin")
#> Coordinate Reference System:
#>   User input: +proj=robin
#>   wkt:
#>   PROJCRS["unknown",
#>     BASEGEOGCRS["unknown",
#>       DATUM["World Geodetic System 1984",
#>         ELLIPSOID["WGS 84",6378137,298.257223563,
#>           LENGTHUNIT["metre",1]],
#>           ID["EPSG",6326]],
#>           PRIMEM["Greenwich",0,
#>             ANGLEUNIT["degree",0.0174532925199433],
#>             ID["EPSG",8901]]],
#>           CONVERSION["unknown",
#>             METHOD["Robinson"],
#>             PARAMETER["Longitude of natural origin",0,
#>               ANGLEUNIT["degree",0.0174532925199433],
#>               ID["EPSG",8802]],
#>             PARAMETER["False easting",0,
#>               LENGTHUNIT["metre",1],
#>               ID["EPSG",8806]],
#>             PARAMETER["False northing",0,
#>               LENGTHUNIT["metre",1],
#>               ID["EPSG",8807]]],
#>             CS[Cartesian,2],
#>               AXIS["(E)",east,
#>                 ORDER[1],

```

```

#>           LENGTHUNIT["metre",1,
#>             ID["EPSG",9001]],
#>           AXIS["(N)",north,
#>             ORDER[2],
#>           LENGTHUNIT["metre",1,
#>             ID["EPSG",9001]]]

```

La mayoría de los objetos espaciales serán de la clase `sf`, por tanto, resulta interesante conocer cómo se proyectan estos objetos.

Es posible proyectar un objeto `sf` mediante la función `st_transform()`. En el siguiente ejemplo vemos cómo partimos de un objeto con **EPSG:4326** y cambiamos su proyección a otras proyecciones, como **Mercator** o **Robinson**:

```

# Usa datos del paquete mapSpain

library(giscoR)

paises <- gisco_get_countries()

# Comprobamos el CRS de estos datos
# Se puede almacenar en un objeto y usar posteriormente
# Vemos que es EPSG:4326, por tanto son coordenadas geográficas longitud/latitud
st_crs(paises)
#> Coordinate Reference System:
#>   User input: EPSG:4326
#>   wkt:
#>     GEOGCS["WGS 84",
#>       DATUM["WGS_1984",
#>         SPHEROID["WGS 84",6378137,298.257223563,
#>           AUTHORITY["EPSG", "7030"]],
#>         AUTHORITY["EPSG", "6326"]],
#>       PRIMEM["Greenwich",0,
#>         AUTHORITY["EPSG", "8901"]],
#>       UNIT["degree",0.0174532925199433,
#>         AUTHORITY["EPSG", "9122"]],
#>       AUTHORITY["EPSG", "4326"]]

# Plot
plot(st_geometry(paises), axes = TRUE)

# Proyectamos a Mercator
# El eje cambia porque Mercator usa metros
paises_merc <- st_transform(paises, st_crs(3857))
plot(st_geometry(paises_merc), axes = TRUE)

# Proyectamos a Robinson
paises_robin <- st_transform(paises, st_crs("+proj=robin"))
plot(st_geometry(paises_robin), axes = TRUE)

```

Como se comentó anteriormente, cuando se usan geodatos de diversas fuentes, es necesario que todos presenten el mismo CRS. En la Fig ?? se muestra lo que ocurre si esto no se cumple:

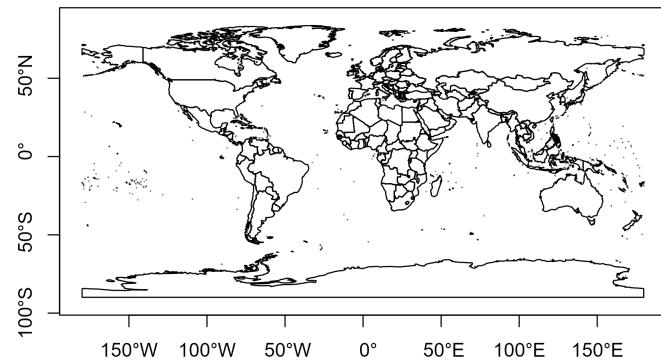


Figura 24: Proyección del mundo en coordenadas geográficas (EPSG 4326)

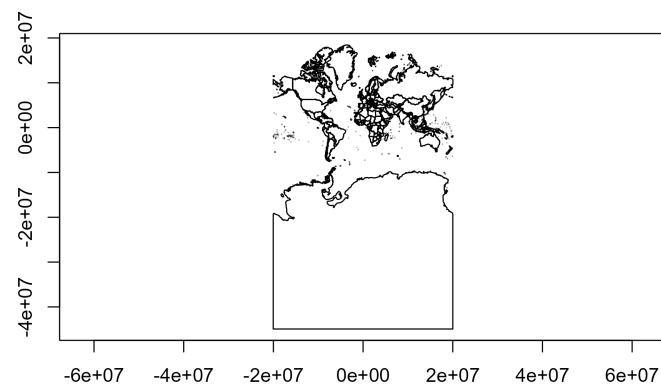


Figura 25: Proyección del mundo en Mercator (EPSG 3857)

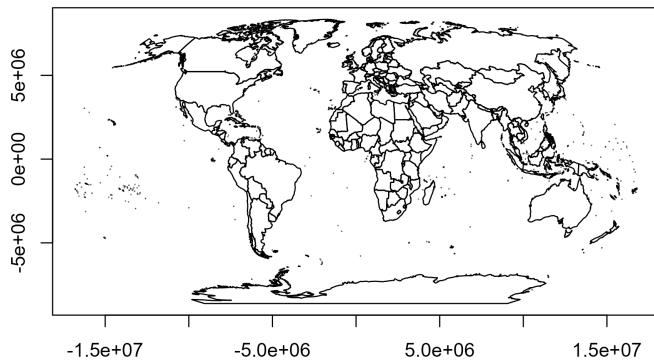


Figura 26: Proyección del mundo en Robinson (+proj=robin)

```
# Añadimos a este mapa puertos mundiales de giscoR
puertos <- gisco_get_ports()
plot(st_geometry(paises_robin), main = "Puertos en el mundo")
plot(st_geometry(puertos), add = TRUE, col = "red", pch = 20)
```



Figura 27: Ejemplo: Puertos del mundo

Vemos que ha habido algún tipo de error, ¿a que puede deberse?

```
# Comprueba CRS
st_crs(puertos) == st_crs(paises_robin)
#> [1] FALSE
# Los puertos no están en Robinson! Proyectamos al mismo CRS
```

```

puertos_robin <- st_transform(puertos, st_crs(paises_robin))
plot(st_geometry(paises_robin), main = "Puertos en el mundo")
plot(st_geometry(puertos_robin), add = TRUE, col = "blue", pch = 20)

```

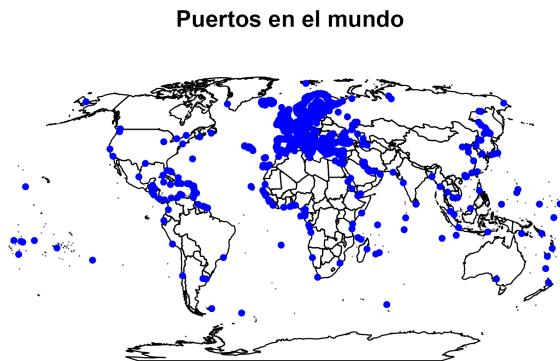


Figura 28: Ejemplo: Puertos del mundo, CRS alineados

Como vemos, en el primer mapa (Fig. 27) los puertos se concentran en un único punto, dado que no están referenciados en el mismo CRS. Tras proyectarlos al mismo CRS, el mapa se representa adecuadamente (Fig. 28).

En otros paquetes, como **sp** o **raster**, existen funciones parecidas que nos van a permitir obtener los parámetros de un CRS y proyectar los objetos al CRS deseado. Cuando empleemos el paquete **sp** podemos usar las funciones **CRS()** y **spTransform()**:

```

library(sp)

# Convertimos sf a sp
paises_sp <- as(paises, "Spatial")

# En sp podemos usar:
# CRS("+proj=robin")
#
# O también desde sf
# CRS(st_crs(paises_robin)$proj4string)

paises_sp_robin <- spTransform(paises_sp, CRS("+proj=robin"))
plot(paises_sp_robin)

```

En el caso de un objeto **raster**, podemos usar **crs()** y **projectRaster()**:

```

library(raster)

# Extrae información de altitud para España

```

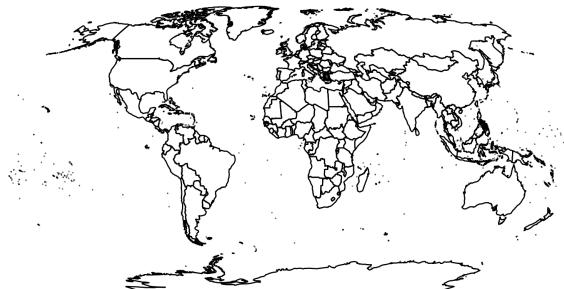


Figura 29: Transformaciones en sp

```
elev <- raster("data/ESP_msk_alt.grd")

# Transforma
elev_robinson <- projectRaster(elev, crs = crs("+proj=robin"))
plot(elev_robinson)
```

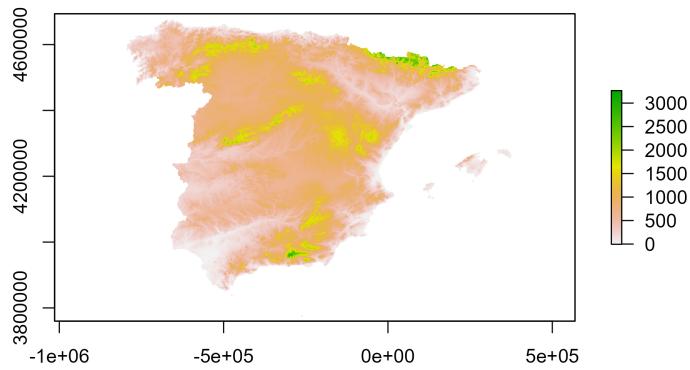


Figura 30: Transformaciones en raster

Por último, en el paquete `terra` las funciones correspondientes son `crs()` y `project()`:

```
library(terra)

# Convierte de raster a terra
elev_terra <- rast(elev)
```

```
# Transforma
elev_terra_robinson <- terra::project(elev_terra, terra::crs(elev_terra))
plot(elev_terra_robinson)
```

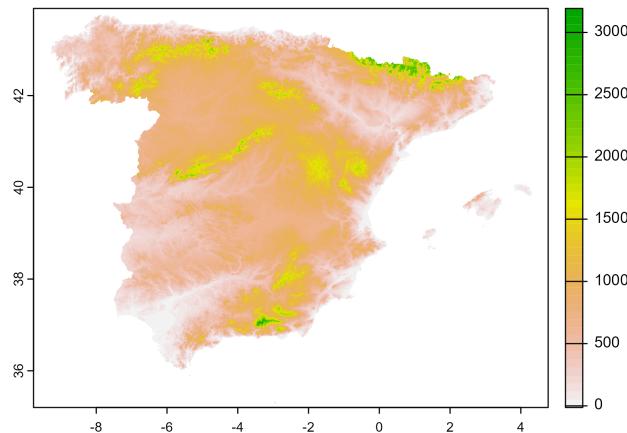


Figura 31: Transformaciones en terra

### 3.4.3 ¿Qué proyección uso?

El CRS adecuado para cada análisis depende de la localización y el rango espacial de los datos. Un CRS adecuado para representar un mapa del mundo puede no serlo para representar datos de zonas específicas de la Tierra. Los recursos web mencionados anteriormente permiten la búsqueda de CRS por zona geográfica, y adicionalmente en **R** existe el paquete **crssuggest** (Walker, 2021) que nos facilita la labor, sugiriendo el CRS más adecuado para cada zona:

```
library(crsuggest)

# Usando raster
sugerencias <- suggest_crs(elev)

# Probamos sugerencia
crs_suggest <- suggest_crs(elev, limit = 1)

elev_suggest <- projectRaster(elev, crs = raster::crs(crs_suggest$crs_proj4))

plot(elev_suggest)
```

```
# Ejemplo con sf: China
china <- gisco_get_countries(country = "China")
china_crs <- suggest_crs(china, limit = 1)
```

Cuadro 3: Tabla sugerencias, detalle

crs_code	crs_name	crs_type	crs_gcs	crs_units	crs_proj4
2062	Madrid 1870 (Madrid) / Spain LCC	projected	4903	m	+proj=lcc +lat_1=40 +lat_0=40
2154	RGF93 / Lambert-93	projected	4171	m	+proj=lcc +lat_0=46.5 +lon_0=3
26191	Merchich / Nord Maroc	projected	4261	m	+proj=lcc +lat_1=33.3 +lat_0=3
3944	RGF93 / CC44	projected	4171	m	+proj=lcc +lat_0=44 +lon_0=3
3943	RGF93 / CC43	projected	4171	m	+proj=lcc +lat_0=43 +lon_0=3
27573	NTF (Paris) / Lambert zone III	projected	4807	m	+proj=lcc +lat_1=44.1 +lat_0=4
27572	NTF (Paris) / Lambert zone II	projected	4807	m	+proj=lcc +lat_1=46.8 +lat_0=4
27563	NTF (Paris) / Lambert Sud France	projected	4807	m	+proj=lcc +lat_1=44.1 +lat_0=4
30791	Nord Sahara 1959 / Nord Algerie	projected	4307	m	+proj=lcc +lat_1=36 +lat_0=36
30493	Voirol 1879 / Nord Algerie (ancienne)	projected	4671	m	+proj=lcc +lat_1=36 +lat_0=36

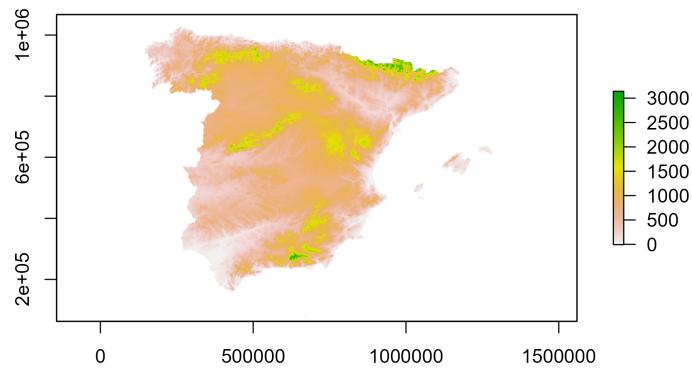


Figura 32: raster: Ejemplo de transformación usando crsuggest

```

china_crs
#> # A tibble: 1 x 6
#>   crs_code crs_name          crs_type crs_gcs crs_units crs_proj4
#>   <chr>     <chr>           <chr>      <dbl>   <chr>       <chr>
#> 1 4584     New Beijing / Gauss-Kruger CM 105E projected    4555 m      +proj=tmerc +lat_0=0 +lon_
#>                                         _0=105 +x_0=4500000 +y_0=0 +ellps=GRS80 +towgs84=0,0,0 +units=m +no_defs

china_suggest <- st_transform(
  china,
  st_crs(as.integer(china_crs$crs_code))
)

plot(st_geometry(china_suggest), axes = TRUE)

```

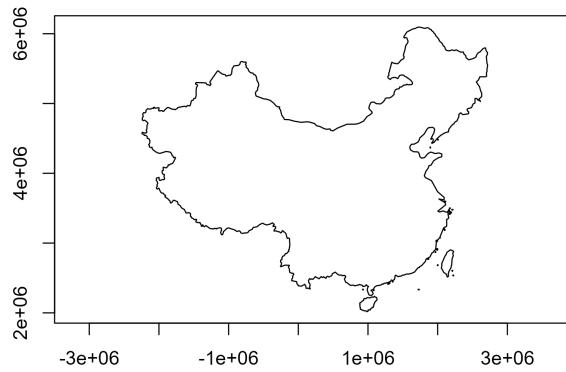


Figura 33: sf: Ejemplo de transformación usando crs\_suggest

## 4 Estadística espacial

La estadística espacial reconoce y aprovecha la ubicación espacial de los datos a la hora de diseñar, recopilar, gestionar, analizar y mostrar las observaciones. Éstas son generalmente **dependientes**, si bien existen modelos espaciales a disposición del investigador que permiten tratar con dicha dependencia espacial a la hora de llevar a cabo labores de predicción. La estadística espacio-temporal incorpora, además, el tiempo y su interacción con el espacio como argumento de ayuda en tales labores predictivas.

Las mediciones y modelos espaciales están presentes, sorprendentemente, en una amplia variedad de disciplinas científicas. Los orígenes de la vida humana vinculan los estudios de la evolución de las galaxias, la estructura de las células biológicas y los patrones de asentamiento arqueológicos. Los ecólogos estudian las interacciones entre plantas y animales. Silvicultores y agricultores necesitan investigar las variaciones que se producen en el terreno para sus experimentos. La estimación de las precipitaciones y de las reservas de oro y petróleo es de vital importancia económica. Estos son, entre otros, buenos ejemplos de la importancia del espacio (espacio-tiempo en su caso) en el mundo de la Ciencia.

En todo caso, la geología, la edafología, el tratamiento de imágenes, la epidemiología, la agronomía, la ecología, la silvicultura, la astronomía, el estudio de la atmósfera, la economía, o simplemente, cualquier disciplina

que trabaje con datos espaciales recopilados de diferentes lugares y en distintos instantes temporales, necesita del desarrollo de modelos geoestadísticos que indiquen la estructura e intensidad de la dependencia espacial y/o espacio-temporal presente en los fenómenos que comprenden.

Sin embargo, el estudio de la **variabilidad espacial**, y sobre todo espacio-temporal, es una disciplina relativamente nueva en el marco de la Estadística, lo que explica la escasez de instrumentos de estadística espacial 30 años atrás. En los últimos 10 años ha habido una creciente toma de conciencia de esta necesidad, habiéndose realizado un gran esfuerzo por buscar herramientas adecuadas y útiles a tales efectos. Y todo ello porque utilizar modelos espaciales o espacio-temporales para caracterizar y explotar la dependencia espacial (o espacio-temporal) de un conjunto de observaciones tiene importantes ventajas:

1. Modelos más generales, ya que, en la mayoría de los casos, los modelos clásicos que no tienen en consideración la dimensión espacial o la interacción de las dimensiones espacial y temporal son un caso particular de un modelo espacial o espacio-temporal.
2. Estimaciones más eficientes: de la tendencia, de los efectos de las variables explicativas, de promedios regionales,...
3. Mejora de las predicciones: más eficientes, con propiedades de extrapolación más estables,...
4. La variación espacial no explicada en la estructura de la media debe ser absorbida por la estructura del error, por lo que un modelo que incorpore la dependencia espacial puede decirse que está protegido frente a una mala especificación de este tipo. Esto, en muchos casos, tiene como resultado una simplificación en la especificación de la tendencia; en general, los modelos con dependencia espacial suelen tener una descripción más parsimoniosa (en ocasiones con muchos menos parámetros) que los clásicos modelos de superficie de tendencia.

Estas mejoras de la estadística espacial y espacio-temporal, junto con el fuerte y reciente desarrollo de los Sistemas de Información Geográfica o GIS (Geographic Information System), han propiciado que en la actualidad exista una importante motivación por la búsqueda de herramientas espaciales o espacio-temporales.

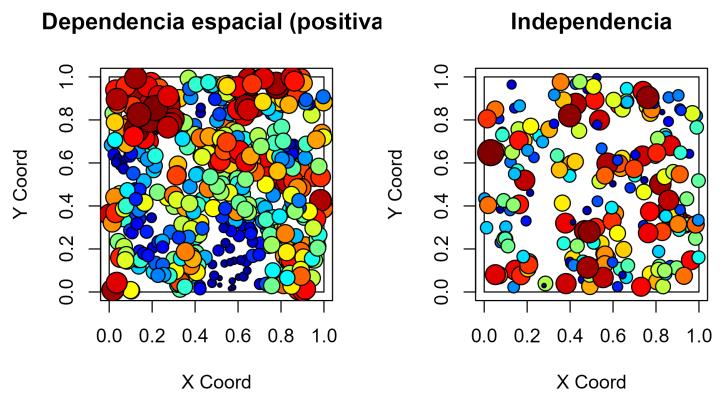
#### 4.1 Antes de continuar... dependencia espacial.

Frecuentemente los datos tienen una componente espacial y/o temporal asociada a ellos y es de esperar que datos cercanos en el espacio o en el tiempo sean más semejantes que aquellos que están más alejados; en cuyo caso **no** deben ser modelados como estadísticamente independiente, sino que habrá que tomar en cuenta esa dependencia espacial o espacio-temporal.

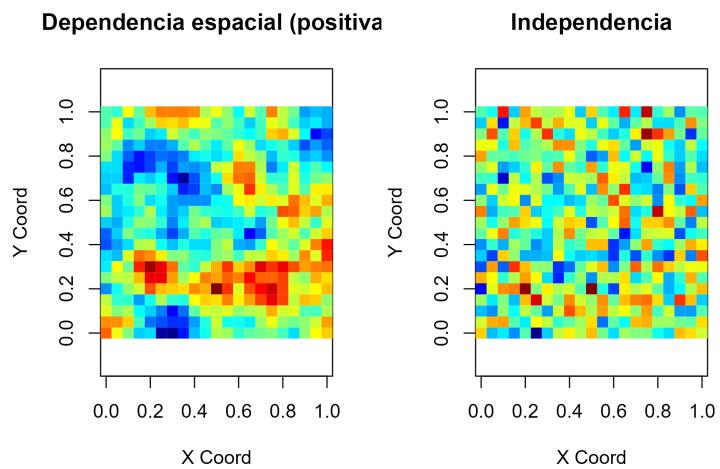
De forma natural y de acuerdo a la Ley Tobler (1973) surge la idea de que los datos cercanos en el espacio o en el tiempo serán más similares y estarán más correlacionados entre sí que aquellos que están más lejos. Además, esta correlación disminuye al aumentar la separación entre ellos, por lo que se puede pensar en la presencia de una dependencia espacial o espacio-temporal. Esto da lugar al concepto de proceso espacial o espacio-temporal.

Si los datos no exhiben dependencia espacial no tiene sentido aplicar las herramientas de estadística espacial. Veamos un ejemplo simulado de unos datos que muestran dependencia espacial y otros puramente aleatorios.

```
#> 'RandomFields' will use OMP
```



Comentar



## 4.2 Datos espaciales

Los **datos espaciales**, también conocidos como datos **geoespaciales**, son aquellos datos relacionados o que contienen información de una localización o área geográfica de la superficie de la Tierra.

La forma más intuitiva de representar los datos espaciales es a través de un mapa.

```
<!-- Propuesta: mapa temático cualquiera con poco código para empezar como el libro de SDS https://keen-->
<!-- Y comentar algunas características de los datos espaciales que luego se explican. -->
```

```
# Mapa de porcentaje de mujeres en Castilla-La Mancha
library(mapSpain)

# Datos de población
pob <- mapSpain:::pobmun19
```

```

# Datos en forma de tabla, sin información en formato espacial
head(pob)
#>   cpro provincia cmun      name  pob19    men women
#> 1  02 Albacete  001 Abengibre  790  379  411
#> 2  02 Albacete  002 Alatoz    519  291  228
#> 3  02 Albacete  003 Albacete 173329 84687 88642
#> 4  02 Albacete  004 Albatana   692  356  336
#> 5  02 Albacete  005 Alborea    658  337  321
#> 6  02 Albacete  006 Alcadozo   654  363  291

# Porcentaje
pob$porc_mujeres <- pob$women / pob$pob19

# Datos espaciales
geo <- esp_get_munic(region = "Castilla-La Mancha")

# Estos datos tienen una columna (geometry) con coordenadas.
head(geo)
#>   codaauto      ine.ccaa.name cpro ine.prov.name cmun      name LAU_CODE
#> 76      08 Castilla - La Mancha 02     Albacete  001 Abengibre  02001 POLYGON ((-1.58316 39.20
#> 69      08 Castilla - La Mancha 02     Albacete  002 Alatoz    02002 POLYGON ((-1.40607 39.12
#> 119     08 Castilla - La Mancha 02     Albacete  003 Albacete  02003 POLYGON ((-2.0562 38.886
#> 106     08 Castilla - La Mancha 02     Albacete  004 Albatana   02004 POLYGON ((-1.54055 38.61
#> 83      08 Castilla - La Mancha 02     Albacete  005 Alborea   02005 POLYGON ((-1.38514 39.35
#> 81      08 Castilla - La Mancha 02     Albacete  006 Alcadozo  02006 POLYGON ((-2.15635 38.71

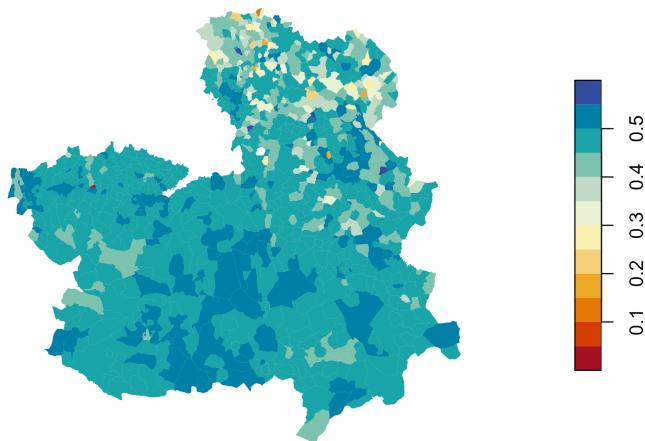
# Une ambos datos
geo_pob <- merge(geo,
  pob,
  by = c("cpro", "cmun"),
  all.x = TRUE
)

# Mapa básico
plot(geo_pob["porc_mujeres"],
  # Cambiamos título
  main = "Castilla-La Mancha: % mujeres (2019)",

  # Cambiamos la paleta de colores para hacerlo mas atractivo
  border = NA,
  pal = hcl.colors(12, "RdYlBu")
)

```

Castilla-La Mancha: % mujeres (2019)



### 4.3 Clasificación de datos espaciales

Tal y como acabamos de señalar y de acuerdo con Schabenberger y Gotway (2005, p. 6), debido a que los datos espaciales surgen en una gran variedad de campos y aplicaciones, también hay una gran variedad de tipos de datos espaciales, estructuras y escenarios. Por tanto, una clasificación exhaustiva de los datos espaciales sería un reto muy difícil y hemos apostado por una clasificación general, simple y útil de datos espaciales proporcionada por Cressie (1993).

La **clasificación** de Cressie de datos espaciales se basa en la naturaleza del dominio espacial en estudio. Dependiendo de esto, podemos tener: datos geoestadísticos, datos de patrones de puntos y datos latice.

Siguiendo a Cressie (1993), sea  $s \in \mathbb{R}^d$  una localización en un espacio Euclídeo  $d$ -dimensional y  $Z(s) : s \in \mathbb{R}^d$  una función aleatoria espacial, donde  $Z$  representa el atributo en el cual estamos interesados:

1. **Datos geoestadísticos:** Surgen cuando el dominio en estudio es **conjunto y fijo**  $D$ . Es decir: (i)  $Z(s)$  se puede observar en cualquier punto del dominio (continuo); y (ii) los puntos en  $D$  no son estocásticos (son fijos,  $D$  es el mismo para todas las realizaciones de la función aleatoria espacial ).

Algunos ejemplos de datos geoestadísticos son el nivel de un contaminante en una ciudad, los valores de precipitación o temperatura del aire en un país, las concentraciones de metales pesados en la capa superior del suelo de una región, etc.

Es obvio que, al menos en teoría, el nivel de un contaminante específico podría medirse en cualquier lugar de la ciudad; Lo mismo puede decirse de las mediciones de precipitaciones o temperaturas del aire en un país o concentraciones de un metal pesado en una región.

Sin embargo, en la práctica, no es posible una observación exhaustiva del proceso espacial. Por lo general, el proceso espacial se observa en un conjunto de ubicaciones (por ejemplo, el nivel de un contaminante específico en una ciudad se observa en los puntos donde están ubicadas las estaciones de monitoreo) y, basado en tales valores observados, el análisis geoestadístico reproduce el comportamiento de el proceso espacial en todo el dominio de interés.

En el análisis geoestadístico lo más importante es cuantificar la correlación espacial entre observaciones (a través de la herramienta básica en geoestadística, el semivariograma) y utilizar esta información para lograr los objetivos anteriores.

# ejemplo

**2. Datos reticulares:** Surgen cuando: (i) el dominio bajo estudio  $D$  es **discreto**, es decir,  $Z(s)$  puede observarse en una serie de ubicaciones fijas que pueden enumerarse. Estas ubicaciones pueden ser puntos o regiones, pero generalmente son códigos postales, pistas censales, vecindarios, provincias, países, etc., y los datos en la mayoría de los casos son datos agregados espacialmente sobre estas áreas. Aunque estas regiones pueden tener una forma regular, normalmente la forma que tienen es irregular, y esto, junto con el carácter espacialmente agregado de los datos, es por lo que los datos lattice tambien se denominan datos regionales. Y (ii) las ubicaciones en  $D$  no son estocásticas. Por supuesto, un concepto clave en el análisis de los datos lattice es el **vecindario**.

Algunos ejemplos de reticulares incluyen la tasa de desempleo por estados, los datos de delincuencia por comarcas, rendimientos agrícolas en parcelas, precios medios de la vivienda por provincias, etc.

#### # ejemplo

**3. Procesos de puntos:** Mientras que en los datos geoestadísticos y reticulares el dominio  $D$  es fijo, en los datos de patrones puntuales el dominio es discreto o continuo, pero **aleatorio**. Los patrones de puntos surgen cuando el atributo bajo estudio es la ubicación de los eventos (observaciones). Es decir, el interés radica en dónde ocurren eventos de interés.

Algunos ejemplos de patrones de puntos son la ubicación de incendios en una región española, la ubicación de los árboles en un bosque o la ubicación de nidos en una colonia de aves reproductoras, la localización de los delitos en una ciudad, entre muchas otras.

En estos En los casos, es obvio que  $D$  es aleatorio y los puntos de observación no dependen del investigador. El principal objetivo del análisis de patrones de puntos es determinar si la ubicación de los eventos tiende a exhibir un patrón sistemático sobre el área en estudio o, por el contrario, son aleatoriamente repartido.

Más concretamente, nos interesa analizar si la ubicación de los eventos es completamente aleatorio espacialmente (la ubicación donde ocurren los eventos no se ve afectada por la ubicación de otros eventos), uniforme o regular (cada punto está tan lejos de todos sus vecinos como sea posible) o agrupados o agregados (la ubicación de los eventos se concentra en grupos).

#### # ejemplo

## 5 Aplicaciones

### 5.1 Temperaturas mínimas en España

#### Objetivo de aprendizaje

Este caso práctico muestra como leer y graficar datos espaciales en R. Para ello, vamos a trabajar con los datos de temperatura mínima registradas en España por las estaciones meteorológicas de la AEMET.

#### Tarea 1: Abrimos RStudio

El presente análisis se va a realizar empleando RStudio, por lo que empezaremos abriendo el programa y creando un nuevo script de R en *File>New File>R script*.

#### Tarea 2: Importamos y describimos los datos objeto de estudio

El conjunto de datos proporcionado (**tempmin.csv**) contiene el nivel de temperatura del aire en España entre el 6 y el 10 de Enero de 2021<sup>1</sup>. Estos datos han sido descargados usando la librería **climaemet** (Pizarro, Hernández-Gómez, & Fernández-Avilés, 2021) y han sido posteriormente tratados para su uso en esta práctica.

<sup>1</sup><https://www.mitma.gob.es/organos-colegiados/consejo-superior-geografico/csg/etrs89/etrs89-nuevo-sistema-de-referencia-geodesico-oficial-en-espana>

Cuadro 4: Detalle del objeto tmin

fecha	indicativo	tmin	longitud	latitud
2021-01-06	4358X	-4.7	-5.880556	38.95556
2021-01-06	4220X	-7.0	-4.616389	39.08861
2021-01-06	6106X	4.7	-4.748333	37.02944
2021-01-06	9698U	-6.8	0.865278	42.20528
2021-01-06	4410X	-3.4	-6.385556	38.91583
2021-01-06	1331A	1.0	-7.031389	43.52472

El primer paso consiste en importar la base de datos de temperatura mínima. El archivo está en formato csv, por lo cual es un fichero de texto plano. Podemos usar varias funciones para realizar la importación, en este caso vamos a emplear paquetes del `tidyverse` para realizar todo el tratamiento de datos:

```
library(readr)

tmin <- read_csv("data/tempmin.csv")
```

#### Q1: ¿Qué información tiene tmin?

Podemos observar que la tabla generada contiene 5 columnas distintas:

- **fecha**: Indicando la fecha de observación.
- **indicativo**: Es el identificador de la estación de la AEMET que registró el dato.
- **tmin**: Dato de temperatura mínima registrada en cada fecha por la estación correspondiente.
- **longitud, latitud**: Coordenadas geográficas de la estación

#### Tarea 3. Convertir tmin a un objeto de la clase espacial geoR

Para convertir un objeto a geodata (el formato requerido por `geoR`), proporcionaremos una tabla con las coordenadas y los valores a incluir en cada coordenada. En este ejemplo, vamos a emplear sólamente los datos correspondientes al **8 de enero**.

```
library(dplyr)
library(geoR)

tmin_geoR <- tmin %>%
  filter(fecha == "2021-01-08") %>%
  # Seleccionamos las columnas de interés
  dplyr::select(longitud, latitud, tmin) %>%
  # Y creamos el objeto geodata
  as.geodata(
    coords.col = 1:2,
    data.col = 3
  )

summary(tmin_geoR)
#> Number of data points: 211
#>
```

```

#> Coordinates summary
#>   longitud latitud
#>   min -9.291389 35.27639
#>   max 4.215556 43.78611
#>
#> Distance summary
#>   min           max
#> 0.01024389 13.85144264
#>
#> Data summary
#>      Min.    1st Qu.    Median      Mean    3rd Qu.    Max.
#> -14.9000000 -4.6000000 -0.5000000 -0.6293839  3.5000000 13.6000000
plot(tmin_geoR)

```

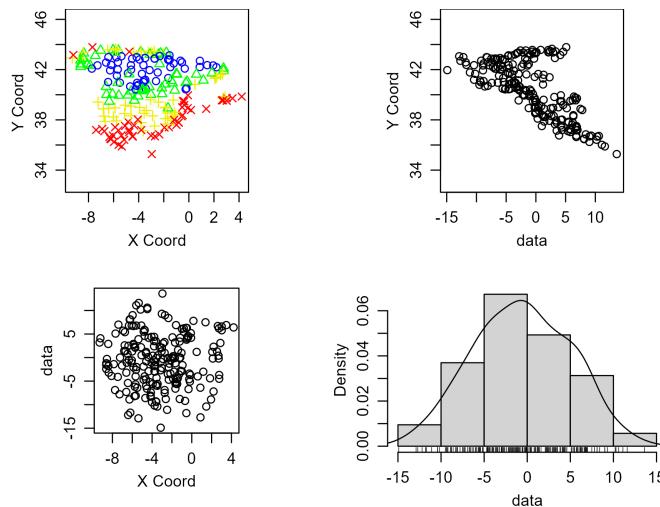


Figura 34: Objetos en geoR

DIEGO, y yo por qué se que España es esto 4326...

Está explicado en la sección de CRS, es el código del CRS que usan los GPS, el habitual para coordenadas longitud/latitud

*Tarea 4. Convertir tmin a un objeto de la clase espacial sf*

En esta tarea, convertiremos los datos de tmin en un objeto espacial sf, es decir, datos espaciales de tipo vector.

Los datos de tmin contienen coordenadas geográficas longitud/latitud, así que como vimos en la sección **Sistema de Referencia de Coordenadas (CRS)** el CRS a emplear ha de ser un CRS geográfico. Usaremos el código EPSG 4326, que corresponde a coordenadas geográficas y suele ser el habitual en este tipo de situaciones.

```

library(sf)

tmin_sf <- st_as_sf(tmin,
  coords = c("longitud", "latitud"),
  crs = 4326
)

```

```
tmin_sf
#> Simple feature collection with 1066 features and 3 fields
#> Geometry type: POINT
#> Dimension: XY
#> Bounding box: xmin: -9.291389 ymin: 35.27639 xmax: 4.215556 ymax: 43.78611
#> Geodetic CRS: WGS 84
#> # A tibble: 1,066 x 4
#>   fecha     indicativo   tmin      geometry
#>   * <date>    <chr>     <dbl>    <POINT [°]>
#> 1 2021-01-06 4358X     -4.7 (-5.880556 38.95556)
#> 2 2021-01-06 4220X     -7  (-4.616389 39.08861)
#> 3 2021-01-06 6106X     4.7  (-4.748333 37.02944)
#> 4 2021-01-06 9698U     -6.8  (0.865278 42.20528)
#> 5 2021-01-06 4410X     -3.4  (-6.385556 38.91583)
#> 6 2021-01-06 1331A      1  (-7.031389 43.52472)
#> 7 2021-01-06 1690A     -0.1  (-7.859722 42.32528)
#> 8 2021-01-06 8489X     -8  (-0.255833 40.43333)
#> 9 2021-01-06 8025      2  (-0.494167 38.3725)
#> 10 2021-01-06 9784P    -10  (0.224722 42.63)
#> # ... with 1,056 more rows
```

*Tarea 5: Dibujemos las estaciones de monitoreo de la temperaria mínima en un mapa de España. Ámbito espacial.*

Vamos además a incluir una capa de las comunidades autónomas de España. Para ello usaremos un paquete API que nos proporciona esta información en formato sf:

```
library(mapSpain)
# sf object
esp <- esp_get_ccaa() %>%
  # No vamos a usar Canarias en este análisis
  filter(ine.ccaa.name != "Canarias")

plot(esp$geometry) # Dibujamo el mapa de España menos las Islas Canarias
```

**Q2: ¿Tengo el Sistema de referencia de coordenadas (CRS) de las estaciones de monitoreo en la misma proyección que el contorno de España?**

Como se comentó en la sección correspondiente, cuando se emplean datos geográficos provenientes de varias fuentes, es necesario asegurarse de que ambos objetos están usando el mismo CRS. Vamos a comprobarlo:

```
st_crs(tmin_sf) == st_crs(esp)
#> [1] FALSE
```

Vemos que no lo están, por lo que vamos a proyectar las coordenadas a un CRS común. En este caso usaremos el CRS de referencia de tmin\_sf:

```
esp2 <- st_transform(esp, st_crs(tmin_sf))

st_crs(tmin_sf) == st_crs(esp2)
#> [1] TRUE
```



Figura 35: Mapa de España (Sin Canarias)

Dibujamos las estaciones de monitoreo con el contorno de España. Vamos a usar el paquete `ggplot2` como referencia, sin embargo existen varios paquetes especializados en mapas temáticos, como pueden ser `tmapo` `maps`.

```
library(ggplot2)

ggplot(esp2) +
  # Para graficar objetos sf debemos usar geom_sf()
  geom_sf() +
  geom_sf(data = tmin_sf) +
  theme_light() +
  labs(
    title = "Estaciones de monitoreo AEMET en España",
    subtitle = "excluyendo las Islas Canarias"
  ) +
  theme(
    plot.title = element_text(
      size = 12,
      face = "bold"
    ),
    plot.subtitle = element_text(
      size = 8,
      face = "italic"
    )
  )
}
```

**Q3.** Mis datos y el contorno de España están en el mismo CRS, pero ¿es el adecuado?

**DIEGO, POR QUÉ TRANSFORMAMOS, PARA PASAR A UTM E INTERPRETAR EN METROS?**

Explicado más adelante, para hacer el grid en metros cuadrados. Si no, se haría en grados cuadrados (?) porque long/lat no es medida de superficie

*Tarea 6: Representamos la variable temperatura mínima tmin para el día 8 de enero de 2021.*

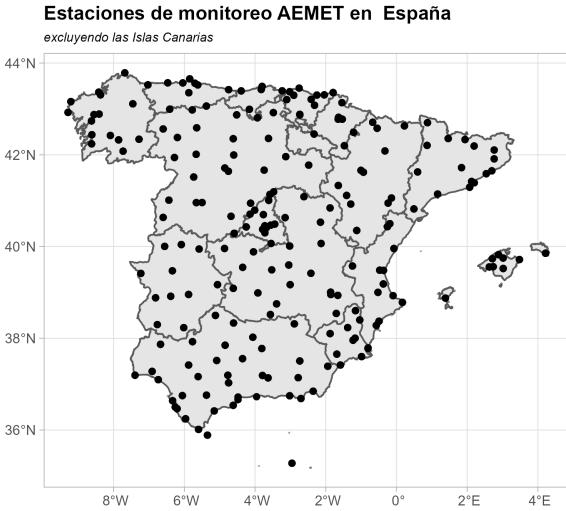


Figura 36: Estaciones de AEMET en la Península Ibérica

En la siguiente tarea, seleccionaremos los datos correspondientes al **8 de enero de 2021** y crearemos un mapa temático en el que representaremos los valores de temperatura mínima registrados en cada estación mediante un código de colores.

```
tmin_8enero <- tmin_sf %>%
  # seleccionamos el día y la variable
  filter(fecha == "2021-01-08")

plot(tmin_8enero[["tmin"]],
  main = "Temperatura mínima (8-enero-2021)",
  pch = 8
)
```

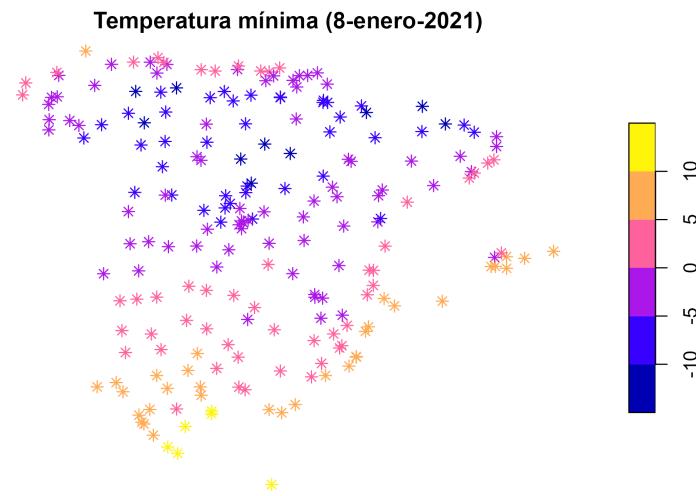


Figura 37: Mapa de puntos con temperatura mínima

Podemos utilizar el ámbito espacial, el contorno de España para graficar y contar la historia de la Filomena

un poco mejor.

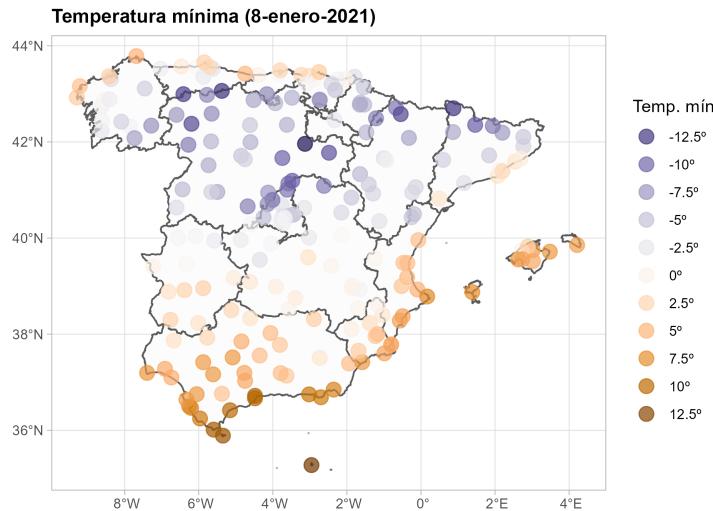


Figura 38: Mapa completo con temperatura mínima

**Q4.** El mapa ha quedado muy claro. Vemos como los datos nos cuentan la historia de Filomena en aquellos sitios donde se tomaron mediciones, pero **¿podríamos tener un mapa de interpolación para tener una estimación de la temperatura mínima en las partes donde la AEMET no tiene estación de monitoreo?**

Tal y como se avanzó en teoría, parece lógico pensar que aquellos puntos que estén cerca tendrán valores similares así que tomemos ventaja de la dependencia espacial y utilicemos un método determinista, como la Distancia Inversa Ponderada, comúnmente conocido por su acrónimo inglés IDW (Inverse distance weighted), el cual es uno de los métodos más simples para llevar para llevar a cabo una interpolación espacial.

En este tipo de análisis, es crucial que el CRS sea el apropiado. En este caso, ya definimos el CRS como un CRS geográfico (es decir, usando coordenadas de longitud y latitud). Sin embargo, para el ejercicio de interpolación es más adecuado usar un CRS local (que provoque pocas deformaciones en la proyección de España) y en alguna unidad de distancia, como metros (ya vimos que en los CRS geográficos las unidades son grados).

Si usamos el paquete `crs_suggest` podemos observar los CRS sugeridos:

```
library(crs_suggest)

sugiere <- suggest_crs(tmin_8enero, units = "m", limit = 5)

# Usamos la sugerencia del paquete
crs_sugerido <- st_crs(sugiere[1, ]$crs_proj4)

esp3 <- st_transform(esp2, crs_sugerido)
tmin_8enero3 <- st_transform(tmin_8enero, crs_sugerido)
```

A continuación, para realizar la interpolación, necesitamos generar una malla que representará las celdas de las que queremos obtener el valor interpolado.

Dado que hemos proyectado nuestros datos a un CRS cuya unidad son los metros, podemos definir el tamaño de cada celda en metros cuadrados. En este caso vamos a usar celdas de 49 kms cuadrados ( $7 \times 7$  kms):

```

malla_sf <- st_make_grid(
  esp3,
  cellsize = 7000
)

```

Graficamos la superficie para ver exactamente qué hemos construido:

```

ggplot(esp3) +
  geom_sf() +
  geom_sf(
    data = malla_sf,
    size = 0.1,
    col = "red", alpha = 1,
    fill = NA
  ) +
  geom_sf(
    data = tmin_8enero3,
    aes(fill = "AEMET Stations"), size = 5, shape = 21,
    color = "blue"
  ) +
  scale_fill_manual(values = adjustcolor("blue", alpha.f = 0.2)) +
  theme_void() +
  theme(legend.position = "bottom") +
  labs(
    title = "Cuadrícula espacial para interpolar",
    fill = ""
  )

```

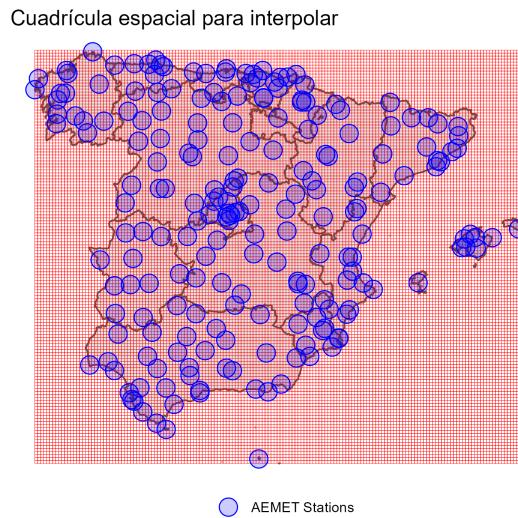


Figura 39: Malla de puntos para interpolación

Se puede observar claramente cada una de las celdas que hemos creado. La interpolación asignará un valor a cada uno de ellas.

A continuación podemos llevar a cabo la interpolación usando el paquete `gstat`. Además, en lugar de celdas (polígonos) es necesario usar puntos en la interpolación. Calcularemos por tanto un punto representativo de

cada celda, el centroide, que es el punto resultante de realizar la media aritmética de las coordenadas de los puntos que componen los lados de cada celda

```
# Calculamos centroide
malla_sf_cent <- st_centroid(malla_sf, of_largest_polygon = TRUE)

library(gstat)
tmin_idw <- idw(
  # Indicamos la variable que queremos interpolar
  tmin ~ 1,
  # Indicamos el conjunto de datos donde está la variable
  tmin_8enero3,
  # Indicamos la malla de destino, en sf
  newdata = malla_sf_cent,
  idp = 2.0 # Especifica la potencia de la IDW
)
#> [inverse distance weighted interpolation]
head(tmin_idw)
#> Simple feature collection with 6 features and 2 fields
#> Geometry type: POINT
#> Dimension: XY
#> Bounding box: xmin: 145790.9 ymin: 68957.31 xmax: 180790.9 ymax: 68957.31
#> CRS: +proj=lcc +lat_1=40 +lat_0=40 +lon_0=0 +k_0=0.9988085293 +x_0=600000 +y_0=600000 +a=600000
#> var1.pred var1.var geometry
#> 1 2.513414 NA POINT (145790.9 68957.31)
#> 2 2.572912 NA POINT (152790.9 68957.31)
#> 3 2.633632 NA POINT (159790.9 68957.31)
#> 4 2.695589 NA POINT (166790.9 68957.31)
#> 5 2.758802 NA POINT (173790.9 68957.31)
#> 6 2.823285 NA POINT (180790.9 68957.31)
```

Representamos la interpolación con un mapa y mapa de contorno muy utilizado para representar datos espaciales. Para ello, vamos a usar el paquete **raster** convirtiendo nuestro objeto interpolado.

```
# Convertimos de sf a SpatialPixels
# Esto funciona porque nuestros puntos sf están espaciados regularmente

tmin_pixels <- tmin_idw %>%
  as("Spatial") %>%
  as("SpatialPixels")

library(raster)
# Creamos un raster de nuestros pixels
rast_esp <- raster(tmin_pixels)

# Transferimos valores del objeto sf al raster
rast_esp2 <- rasterize(
  tmin_idw,
  rast_esp,
  field = "var1.pred",
  fun = mean
)
```

```
# Además, podemos recortar el raster a la forma de España

rast_esp_mask <- mask(rast_esp2, esp3)

plot(rast_esp_mask, col = colores)
contour(rast_esp2, add = TRUE)
```

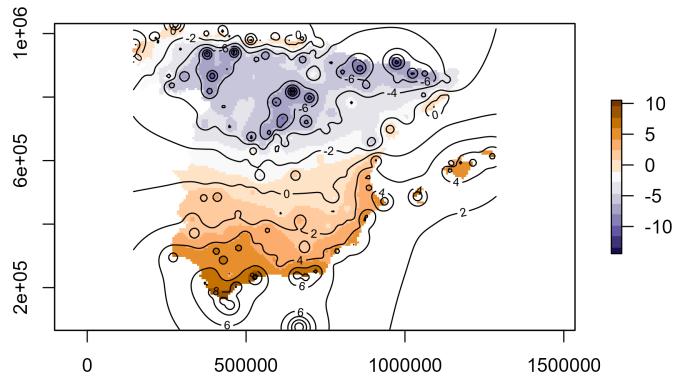


Figura 40: Mapa raster con líneas de nivel

Podemos realizar el mismo mapa usando ggplot2 y la función geom\_contour\_filled:

```
# Creo una tabla para geom contour
coordenadas <- st_coordinates(tmin_idw)
valor <- tmin_idw$var1.pred

idw_df <- data.frame(
  # Necesitamos redondear las coordenadas
  latitud = round(coordenadas[, 2], 6),
  longitud = round(coordenadas[, 1], 6),
  tmin = valor
)

ggplot() +
  geom_contour_filled(
    data = idw_df,
    aes(x = longitud, y = latitud, z = tmin),
    na.rm = TRUE,
    breaks = cortes
  ) +
  # Reajustamos la escala de colores
  scale_fill_manual(values = colores) +
  # CCAA
  geom_sf(data = esp3, fill = NA) +
  theme_minimal() +
  theme(axis.title = element_blank()) +
```

```

  labs(
    fill = "Temp. (°)",
    title = "Temperatura mínima interpolada",
    subtitle = "8 de Enero 2021",
    caption = "Datos: AEMET"
  )

```

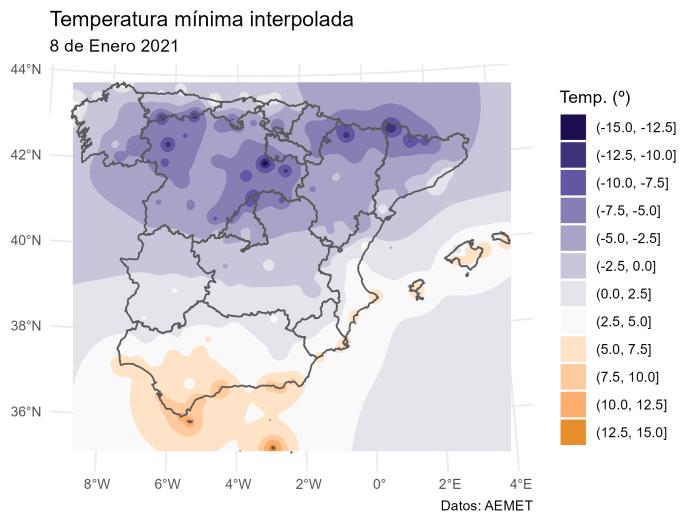


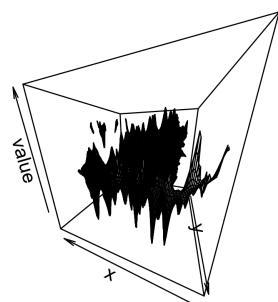
Figura 41: Mapa con ggplot2 y lineas de nivel

Representamos en 3D el mapa anterior, muy utilizado también en datos espaciales

**DIEGO, se utiliza mucho en espacial pero no se si este caso es muy ilustrativo....r**

Por mi lo quitamos

```
persp(rast_esp_mask, theta = 2000, d = 0.1)
```



## 5.2 Renta media por municipios

Esta sección presenta un caso de uso en el que aprenderemos a realizar las siguientes tareas básicas:

- Importar datos tabulares y datos espaciales.
- Realizar un tratamiento de limpieza de datos y cruzar tablas.
- Hacer mapas temáticos. Aprenderemos también algunas nociones básicas sobre cómo crear diferentes clases para un conjunto de datos continuo.

Para ello, partiremos de dos ficheros:

1. Fichero `renta_municipio.csv`: Este fichero contiene información de la Renta Neta per cápita por municipios, distritos y secciones censales. Esta información se ha extraído del Atlas de distribución de renta de los hogares proporcionado por el INE, y ha sido tratado previamente para adaptar la información al presente ejercicio.
2. Fichero `municipios.gpkg`: Es un fichero que contiene datos espaciales (polígonos) de los municipios en España en el año 2019. Se ha extraído del Instituto Geográfico Nacional (IGN) usando el paquete `mapSpain`.

El primer paso en cualquier tipo de análisis de datos es importar los datos al software de tratamiento (en nuestro caso, R) y analizarlos para conocer el tipo de información que contiene:

```
# Usaremos paquetes del tidyverse
library(dplyr)
library(readr)

renta <- read_csv("data/renta_municipio.csv", na = ".")  
  
head(renta)
#> # A tibble: 6 x 6
#>   Unidad      `2019` `2018` `2017` `2016` `2015`
#>   <chr>       <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
#> 1 44001 Ababuj        NA     NA     NA     NA     NA
#> 2 4400101 Ababuj distrito 01      NA     NA     NA     NA
#> 3 4400101001 Ababuj sección 01001    NA     NA     NA     NA
#> 4 40001 Abades        11429  10731  10314  9816  9904
#> 5 4000101 Abades distrito 01        11429  10731  10314  9816  9904
#> 6 4000101001 Abades sección 01001    11429  10731  10314  9816  9904
```

Podemos comprobar que tenemos información para los años 2015 a 2019. Además, la columna `Unidad` contiene un literal con el municipio o sección correspondiente.

```
library(sf)

munis <- st_read("data/municipios.gpkg", quiet = TRUE)

head(munis)
#> Simple feature collection with 6 features and 7 fields
#> Geometry type: MULTIPOLYGON
```

```

#> Dimension:      XY
#> Bounding box:  xmin: -3.140179 ymin: 36.73817 xmax: -2.057058 ymax: 37.54579
#> Geodetic CRS:  ETRS89
#>   codauto ine.ccaa.name cpro ine.prov.name cmun      name LAU_CODE           geom
#> 1    01 Andalucía 04 Almería 001 Abla 04001 MULTIPOLYGON (((-2.775594 3...
#> 2    01 Andalucía 04 Almería 002 Abrucena 04002 MULTIPOLYGON (((-2.787566 3...
#> 3    01 Andalucía 04 Almería 003 Adra 04003 MULTIPOLYGON (((-3.051988 3...
#> 4    01 Andalucía 04 Almería 004 Albánchez 04004 MULTIPOLYGON (((-2.181086 3...
#> 5    01 Andalucía 04 Almería 005 Alboloduy 04005 MULTIPOLYGON (((-2.572442 3...
#> 6    01 Andalucía 04 Almería 006 Albox 04006 MULTIPOLYGON (((-2.128106 3...

```

Podemos comprobar que `munis` es un objeto que contiene Polígonos y varias columnas, entre ellas dos especialmente relevantes: `cpro` y `cmun`, que corresponden a los códigos de provincia y de municipio respectivamente. Podemos comprobar que este código también se encuentra en nuestro dataset `renta`:

```

# Miro un municipio: Noblejas

renta[grep("Noblejas", renta$Unidad), ]
#> # A tibble: 5 x 6
#>   Unidad          `2019` `2018` `2017` `2016` `2015`
#>   <chr>          <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
#> 1 45115 Noblejas 10591  10314  9751  9484  9124
#> 2 4511501 Noblejas distrito 01 11039  10717  10135  9711  9386
#> 3 4511501001 Noblejas sección 01001 11039  10717  10135  9711  9386
#> 4 4511502 Noblejas distrito 02 10276  10029  9475  9319  8938
#> 5 4511502001 Noblejas sección 02001 10276  10029  9475  9319  8938

munis[grep("Noblejas", munis$name), c("name", "cpro", "cmun")]
#> Simple feature collection with 1 feature and 3 fields
#> Geometry type: MULTIPOLYGON
#> Dimension:      XY
#> Bounding box:  xmin: -3.489824 ymin: 39.93003 xmax: -3.372611 ymax: 40.05017
#> Geodetic CRS:  ETRS89
#>   name cpro cmun           geom
#> 1 4985 Noblejas 45 115 MULTIPOLYGON (((-3.44681 40...

```

En el caso de Noblejas, el código completo es 45115. Sin embargo, en el caso de la tabla `renta`, debemos extraer ese valor del literal. Para ello debemos manipular la columna y extraer la primera palabra de la columna `Unidad`:

```

# Creo una función y la aplico a toda la columna
extrae_codigo <- function(x) {
  unlist(strsplit(x, " "))[1]
}

renta$codigo_ine <- sapply(as.character(renta$Unidad), extrae_codigo)

head(renta[c("Unidad", "codigo_ine")])
#> # A tibble: 6 x 2
#>   Unidad           codigo_ine
#>   <chr>            <chr>
#> 1 44001 Ababuj 44001

```

```
#> 2 4400101 Ababuj distrito 01      4400101
#> 3 4400101001 Ababuj sección 01001 4400101001
#> 4 40001 Abades                  40001
#> 5 4000101 Abades distrito 01      4000101
#> 6 4000101001 Abades sección 01001 4000101001
```

Ahora, es necesario crear la misma variable en `munis` para poder realizar el cruce:

```
munis$codigo_ine <- paste0(munis$cpro, munis$cmun)

head(munis[, c("name", "codigo_ine")])
#> Simple feature collection with 6 features and 2 fields
#> Geometry type: MULTIPOLYGON
#> Dimension:      XY
#> Bounding box:  xmin: -3.140179 ymin: 36.73817 xmax: -2.057058 ymax: 37.54579
#> Geodetic CRS:  ETRS89
#>           name codigo_ine          geom
#> 1     Abla    04001 MULTIPOLYGON (((-2.775594 3...
#> 2 Abrucena   04002 MULTIPOLYGON (((-2.787566 3...
#> 3     Adra    04003 MULTIPOLYGON (((-3.051988 3...
#> 4 Albánchez  04004 MULTIPOLYGON (((-2.181086 3...
#> 5 Alboloduy   04005 MULTIPOLYGON (((-2.572442 3...
#> 6     Albox    04006 MULTIPOLYGON (((-2.128106 3...
```

Ya estamos listos para realizar el cruce. Además, seleccionaremos sólo las columnas que vamos a usar, en este caso la del año 2019:

```
munis_renta <- munis %>%
  left_join(renta) %>%
  dplyr::select(name, cpro, cmun, `2019`)
```

**Cuando crucemos datos espaciales con datos no espaciales en R, es importante que el primer dataset sea el que contiene los datos espaciales.** Esto es así porque el objeto resultante “hereda” la clase del primer objeto. A modo de ejemplo, si realizáramos el proceso poniendo los datos espaciales en el lado derecho del join, los datos finales no serán espaciales:

```
# Miramos la clase de munis_renta

class(munis_renta)
#> [1] "sf"           "data.frame"

# Es un sf, por tanto espacial

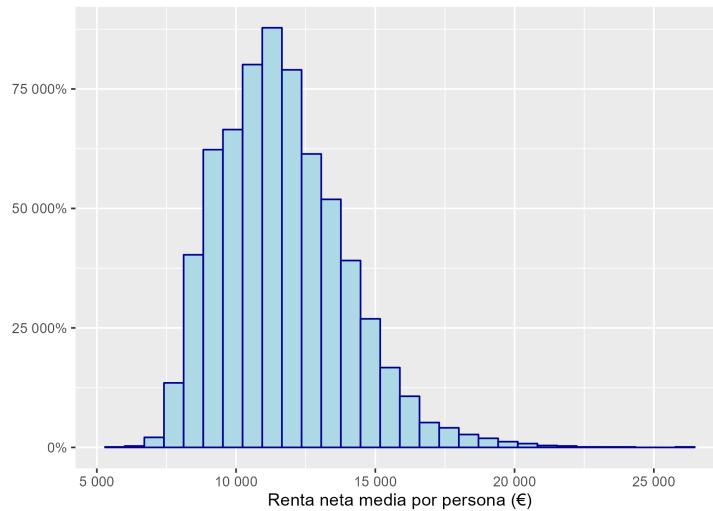
# ¿Qué pasa si realizamos el cruce de la otra manera?
renta %>%
  left_join(munis) %>%
  dplyr::select(name, cpro, cmun, `2019`) %>%
  class()
#> [1] "tbl_df"       "tbl"          "data.frame"

# Es un tibble o data.frame, pero no es espacial!
```

Una vez que tenemos los datos unidos podemos realizar algunos análisis básicos, como la realización de un histograma

```
library(ggplot2)

munis_renta %>%
  ggplot(aes(x = `2019`)) +
  geom_histogram(color = "darkblue", fill = "lightblue") +
  scale_x_continuous(labels = scales::label_number_auto()) +
  scale_y_continuous(labels = scales::label_percent()) +
  labs(
    y = "",
    x = "Renta neta media por persona (€)"
  )
```



Podemos observar que la renta presenta una distribución Gamma con un gran de municipios concentrados en zonas medias de renta y pocos municipios en tramos de rentas altas. Como veremos más adelante, esta distribución va a afectar a la información que transmite el mapa.

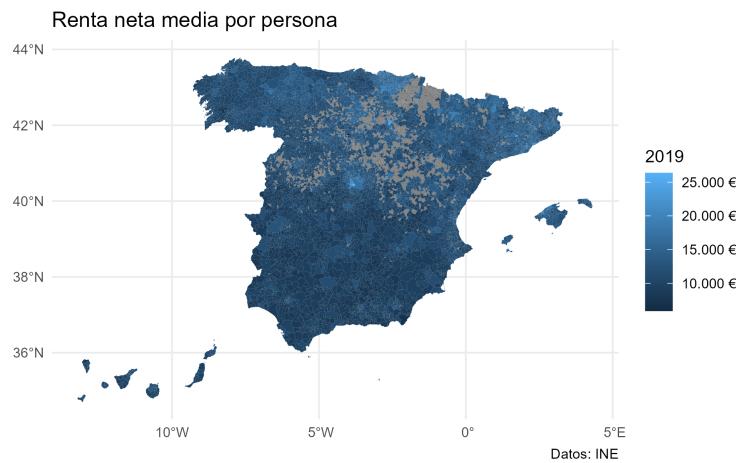
Vamos a realizar ahora un mapa de coropletas mostrando la distribución de la renta usando los valores brutos de renta sin modificar:

```
ggplot(munis_renta) +
  # Usamos geom_sf, y como aes() lo que queremos mostrar, en este caso, el
  # color del polígono representa la renta. Vamos a retirar los bordes con
  # color = NA
  geom_sf(aes(fill = `2019`), color = NA) +
  theme_minimal() +
  scale_fill_continuous(labels = scales::label_number(
    big.mark = ".",
    decimal.mark = ",",
    suffix = " €"
  )) +
  labs(
    title = "Renta neta media por persona",
```

```

    caption = "Datos: INE"
)

```



Este primer mapa no es demasiado informativo, por los siguientes motivos:

- Existe una serie de municipios para los que no tenemos datos.
- La escala de color no es la más adecuada.
- Dada la distribución de los datos, puede ser adecuado crear grupos de renta para que el mapa sea más interpretable.

Vamos a probar a eliminar los municipios sin datos y a cambiar la escala de color:

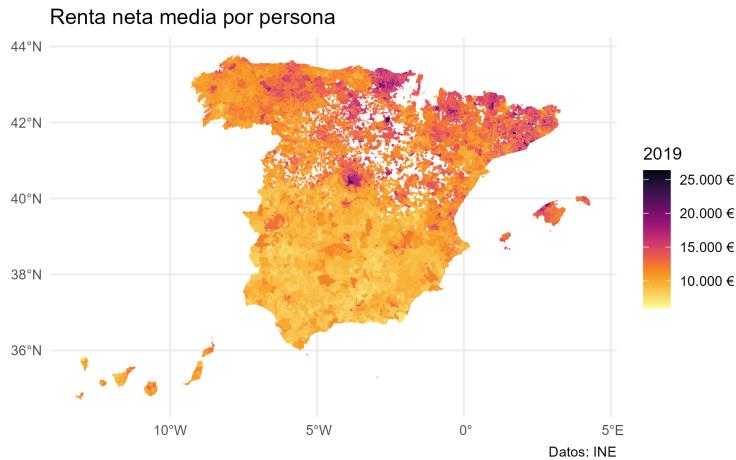
```

munis_renta_clean <- munis_renta %>% filter(!is.na(`2019`))

ggplot(munis_renta_clean) +
  geom_sf(aes(fill = `2019`), color = NA) +
  # Cambiamos la paleta de colores, vamos a usar una paleta denominada Inferno,
  # ya incluida en base R con hcl.colors

  # Como son datos continuos, puedo usar Inferno
  scale_fill_gradientn(
    colours = hcl.colors(20, "Inferno", rev = TRUE),
    labels = scales::label_number(
      big.mark = ".",
      decimal.mark = ",",
      suffix = " €"
    )
  ) +
  theme_minimal() +
  labs(
    title = "Renta neta media por persona",
    caption = "Datos: INE"
)

```



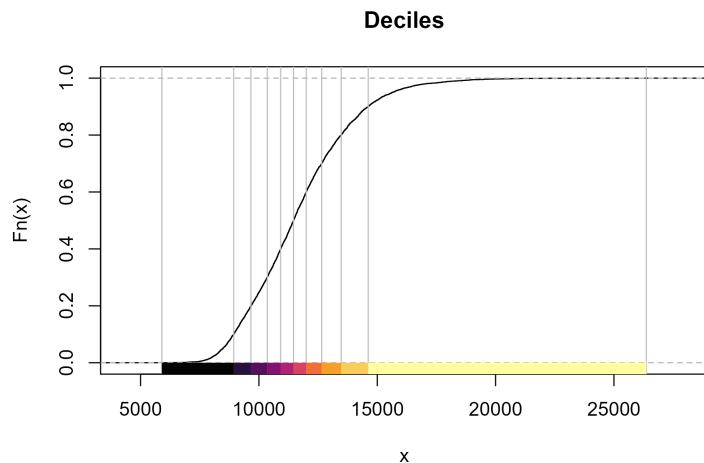
Este mapa nos da algo más de información, y parece intuirse que las rentas más altas se encuentran en zonas de País Vasco, Madrid y Cataluña. Sin embargo, el hecho de que la distribución de los datos no sea normal está afectando a la visualización.

Para intentar atajar este problema, podemos dividir nuestros datos en clases, por ejemplo cuartiles o deciles. Existen varios métodos de clasificación de datos, que en R se encuentran implementados en el paquete `classInt`. A continuación vamos a plantear diversos métodos de clasificación y observar cómo la “historia” que cuenta el mapa varía en función de dichas clases. En este ejemplo planteamos los siguientes métodos de clasificación:

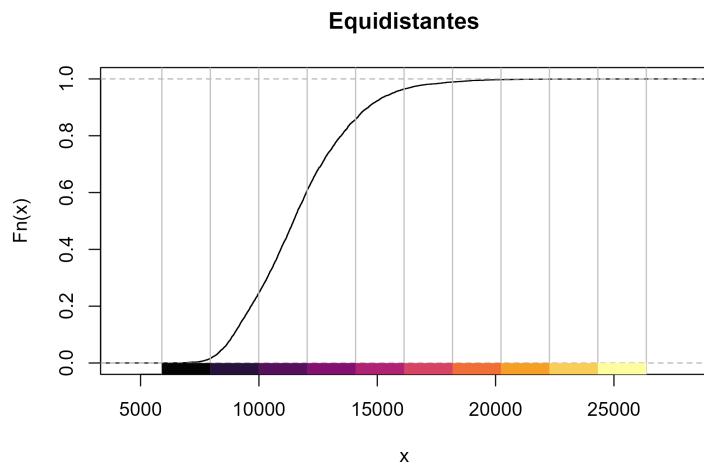
- El método de deciles: Consiste en crear 10 categorías incluyendo el mismo número de registros en cada una de ellas.
- El método de intervalos equivalentes: Este método divide el rango de valores en un número de grupos definido. La distancia de todos los intervalos es idéntica, por lo que este método no tiene en cuenta la distribución de los registros.
- El método de Fisher-Jenks: Este método se desarrolló específicamente para la clasificación de datos espaciales y su visualización en mapas. Produce agrupaciones de tal manera que los datos de cada grupo son “cercanas” entre sí y sustancialmente distintas de los valores de otros grupos.

```
library(classInt)
# Vamos a probar 3 métodos de clasificación: Deciles, tramos de Renta
# equidistantes y Fisher and Jenks

deciles <- classIntervals(munis_renta_clean$`2019`,
  style = "quantile", n = 10
)
deciles
#> style: quantile
#>      [5898,8935.6)  [8935.6,9662.2)  [9662.2,10352.8)  [10352.8,10918)      [10918,11462)      [11462,1
#>          656           656           655           654           655
#> [11998.6,12651.4) [12651.4,13475.8) [13475.8,14618.4)  [14618.4,26367]
#>          656           655           656           656
plot(deciles, pal = hcl.colors(20, "Inferno"), main = "Deciles")
```



```
# Tramos equidistantes en términos de renta
equal <- classIntervals(munis_renta_clean$`2019`,
  style = "equal", n = 10
)
equal
## style: equal
##      [5898,7944.9]  [7944.9,9991.8]  [9991.8,12038.7]  [12038.7,14085.6]  [14085.6,16132.5]  [16132.5,18179.4]
##          103           1510           2374           1637             702
##  [18179.4,20226.3]  [20226.3,22273.2]  [22273.2,24320.1]  [24320.1,26367]
##          52            14              3              1
plot(equal, pal = hcl.colors(20, "Inferno"), main = "Equidistantes")
```

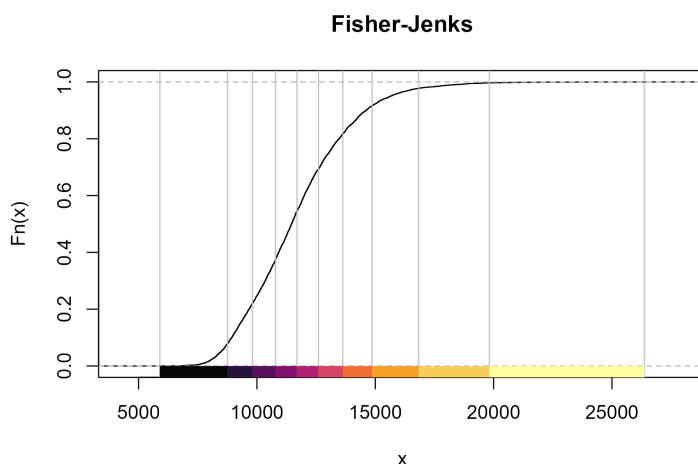


```
fisher <- classIntervals(munis_renta_clean$`2019`,
  style = "fisher",
  # Fuerzo para mejorar la comparación entre métodos
  n = 10
```

```

)
fisher
#> style: fisher
#>      [5898,8758)      [8758,9808)      [9808,10786.5)  [10786.5,11690.5)  [11690.5,12603)  [12603,1
#>      513               928               1012              1120              970
#>  [13626.5,14870)  [14870,16833)  [16833,19819)  [19819,26367]
#>      650               402               125               22
plot(fisher,
  pal = hcl.colors(20, "Inferno"),
  main = "Fisher-Jenks"
)

```



Podemos observar lo siguiente:

- El último decil de renta se corresponde a un rango de entre 15.000 y 25.000 €.
- El método por deciles proporciona unos grupos con valores de renta muy parecidos entre sí en los valores medios. Esto es debido a la propia distribución de la variable.
- El método de rangos equidistantes proporciona algunos grupos con un número muy reducido de municipios.
- El método de Fisher-Jenks puede proporcionar unas clases con unos rangos más apropiados para los tramos altos de renta.

Vamos ahora a realizar 3 mapas distintos, creando clases de renta según cada uno de los métodos anteriormente mostrados.

### Deciles

```

# Extraigo los valores de corte
breaks_d <- deciles$brks

# Y creo unas etiquetas básicas para cada clase
# Creo una función específica para crear etiquetas formateadas
label_fun <- function(x) {

```

```

l <- length(x)
eur <- paste0(prettyNum(round(x, 0),
  decimal.mark = ",",
  big.mark = "."),
  " €")

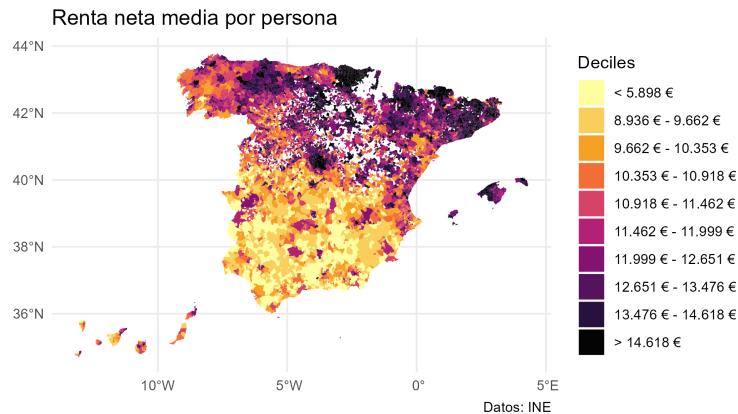
labels <- paste(eur[-1], "-", eur[-1])
labels[1] <- paste("<", eur[1])
labels[l - 1] <- paste(">", eur[l - 1])
return(labels)
}

labels_d <- label_fun(breaks_d)

munis_renta_clean$Deciles <- cut(munis_renta_clean$`2019`,
  breaks = breaks_d,
  labels = labels_d,
  include.lowest = TRUE
)

ggplot(munis_renta_clean) +
  # Cambiamos la variable que usamos para crear el mapa
  geom_sf(aes(fill = Deciles), color = NA) +
  # Necesito cambiar el scale, ya no es continua
  scale_fill_manual(values = hcl.colors(length(labels_d),
    "Inferno",
    rev = TRUE
  )) +
  theme_minimal() +
  labs(
    title = "Renta neta media por persona",
    caption = "Datos: INE"
  )

```



Este mapa ya nos permite observar patrones geográficos, donde se ve una clara diferencia entre la Comunidades Autónomas del Norte y las del Sur. Veamos una representación distinta usando otras clases diferentes:

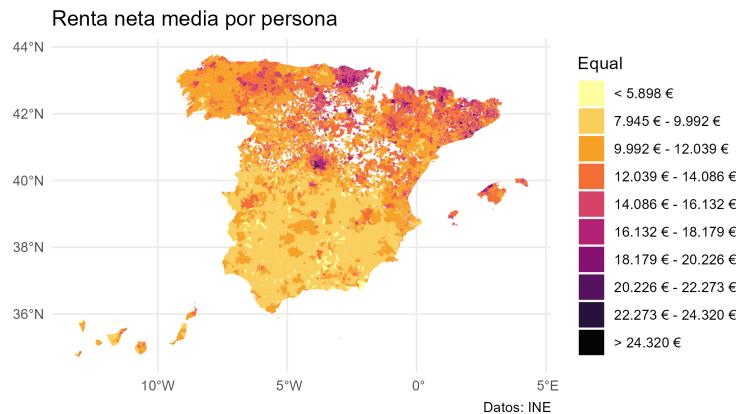
```

breaks_e <- equal$brks
labels_e <- label_fun(breaks_e)

munis_renta_clean$Equal <- cut(munis_renta_clean$`2019`,
  breaks = breaks_e,
  labels = labels_e,
  include.lowest = TRUE
)

ggplot(munis_renta_clean) +
  # Cambiamos la variable que usamos para crear el mapa
  geom_sf(aes(fill = Equal), color = NA) +
  scale_fill_manual(values = hcl.colors(length(labels_e),
    "Inferno",
    rev = TRUE
  )) +
  theme_minimal() +
  labs(
    title = "Renta neta media por persona",
    caption = "Datos: INE"
  )

```



Este otro mapa, sin embargo, se parece más al mapa que hicimos con los datos sin clasificar, donde el peso visual se concentra más bien en los municipios con rentas mucho más altas que el resto (por encima de 18.000 €).

Veamos el mismo mapa usando Fisher-Jenks:

```

breaks_f <- fisher$brks
labels_f <- label_fun(breaks_f)

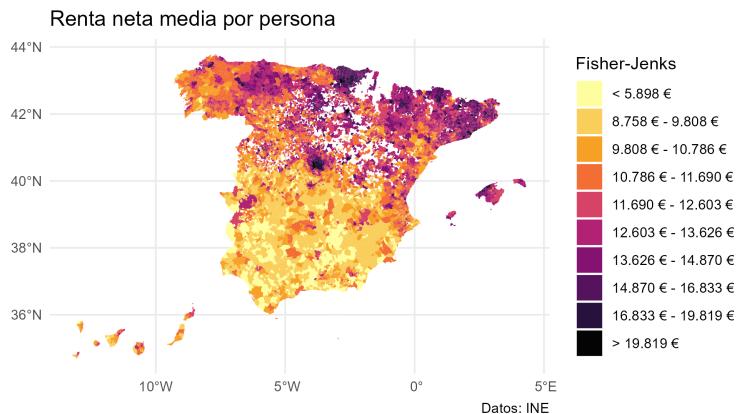
munis_renta_clean$`Fisher-Jenks` <- cut(munis_renta_clean$`2019`,
  breaks = breaks_f,
  labels = labels_f,
  include.lowest = TRUE
)

```

```

)
ggplot(munis_renta_clean) +
  # Cambiamos la variable que usamos para crear el mapa
  geom_sf(aes(fill = `Fisher-Jenks`), color = NA) +
  scale_fill_manual(values = hcl.colors(length(labels_f),
    "Inferno",
    rev = TRUE
  )) +
  theme_minimal() +
  labs(
    title = "Renta neta media por persona",
    caption = "Datos: INE"
  )

```



En este mapa se puede observar de una manera más clara un cluster adicional de renta en la zona de Asturias y el norte de León. Además, gracias a la escala de colores puede intuirse que este clúster de renta no presenta valores tan altos como los observados en País Vasco o Madrid.

En conclusión, en el momento de realizar una visualización de datos es importante conocer el dato a representar, así como entender algunas propiedades básicas de la distribución subyacente. También hemos podido observar que hay ciertas decisiones estéticas (datos continuos vs. agrupados, escala de colores) que tienen una influencia significativa en cómo se percibe la información representada. Es responsabilidad del creador de la visualización el conocer todos estos factores y aplicarlos de manera conveniente.

Como ejemplo, aquí tenemos un mapa de los mismos datos pero la información se presenta de manera sesgada, ¿puedes identificar los motivos?

En este mapa parece que la renta per cápita de las comunidades del norte es desproporcionadamente superior a las del sur. Resumimos aquí los sesgos introducidos en el mapa:

- En primer lugar, se han creado un elevado número de grupos en las zonas de rentas bajas. De esta manera, la escala del mapa parece estar muy fragmentada, sin embargo muchos de esos grupos apenas contienen municipios. A modo de ejemplo, los primeros cuatro grupos únicamente contienen 32 municipios.
- Los grupos no se adaptan a la distribución subyacente de los datos. La mediana de los datos (11.462 €) estaría situada en la antepenúltima clase, de manera que los dos grupos de mayor renta contienen el 50% de los municipios.

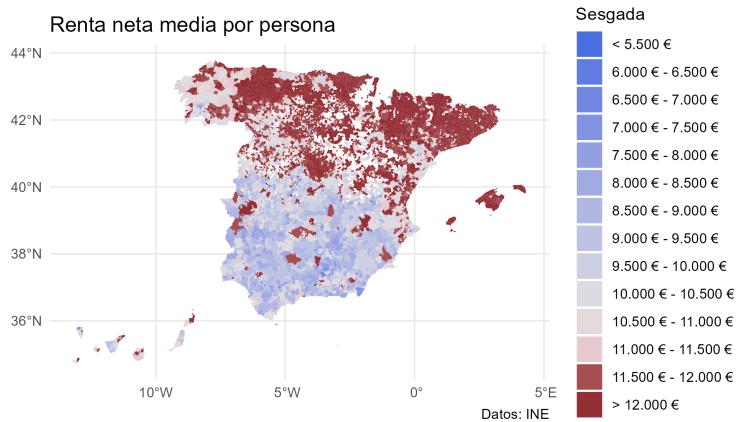


Figura 42: Ejemplo de visualización sesgada

- Además, la escala de color se ha manipulado, de manera que los grupos de mayores renta destaque más que el resto de manera notoria.

### 5.3 Análisis de patrones de puntos

#### GEMA: CAMBIA LO QUE CONSIDERES, QUE DE TODO ESTO NO TENGO NI IDEA

Las técnicas de análisis de patrones de puntos analizan la distribución de eventos geolocalizados. La diferencia fundamental con otros análisis que comprenden también el uso de localizaciones (como las temperaturas mínimas medidas por estaciones meteorológicas) es que en este caso los puntos representan eventos conocidos y aleatorios (por ejemplo, los crímenes ocurridos en una ciudad, accidentes de tráfico o incendios en una región). A diferencia de otros eventos, como el ejemplo de las temperaturas mínimas, la ausencia de datos no se debe a la ausencia de medición (e.g. no existe una estación meteorológica en ese lugar), si no a que no se ha producido el evento en dicha localización.

##### Objetivo de aprendizaje:

##### GEMA

##### Tarea 1: Abrimos RStudio

El presente análisis se va a realizar empleando RStudio, por lo que empezaremos abriendo el programa y creando un nuevo script de R en *Proyecto>File>New File>R script*.

##### Tarea 2: Importamos y describimos los datos objeto de estudio

El primer paso consiste en importar la base de datos de accidentes de tráfico en la ciudad de Madrid para el año 2020 (Portal de datos abiertos del Ayuntamiento de Madrid). El archivo está en formato csv, por lo que usaremos el paquete **readr** para importar los datos:

```
library(readr)
library(dplyr)

# En este caso el archivo está en la carpeta "data" de nuestro proyecto
accidentes2020 <- read_delim("data/2020_Accidentalidad.csv",
  # Configuración adicional
  delim = ";",
```

```

locale = locale(decimal_mark = ",", grouping_mark = "."),
col_types = cols(
    num_expediente = col_character(),
    fecha = col_date(format = "%d/%m/%Y")
),
na = "NULL"
)

summary(accidentes2020)
#> num_expediente           fecha                  hora          localizacion      numero
#> Length:32427            Min.   :2020-01-01  Length:32427  Length:32427  Length:32427
#> Class :character        1st Qu.:2020-03-01  Class1:hms   Class :character  Class :character
#> Mode  :character        Median :2020-07-19  Class2:difftime  Mode  :character  Mode  :character
#>                   Mean   :2020-07-07  Mode  :numeric
#>                   3rd Qu.:2020-10-22
#>                   Max.   :2020-12-31
#>
#> distrito                tipo_accidente      estado_meteorológico tipo_vehiculo      tipo_persona
#> Length:32427             Length:32427       Length:32427  Length:32427  Length:32427
#> Class :character         Class :character   Class :character  Class :character  Class :character
#> Mode  :character         Mode  :character   Mode  :character  Mode  :character  Mode  :character
#>
#>
#>
#> rango_edad               sexo                  lesividad      coordenada_x_utm  coordenada_y_utm  positiva_
#> Length:32427              Length:32427       Min.   : 1.000  Min.   :429069  Min.   :4463495  Length:32427
#> Class :character          Class :character   1st Qu.: 7.000  1st Qu.:439962  1st Qu.:4471803  Class :character
#> Mode  :character          Mode  :character   Median :14.000  Median :441917  Median :4475026  Mode  :character
#>                   Mean   : 9.862  Mean   :442218  Mean   :4474898
#>                   3rd Qu.:14.000  3rd Qu.:444328  3rd Qu.:4477652
#>                   Max.   :77.000  Max.   :454624  Max.   :4488100
#>                   NA's    :14796
#> positiva_droga           Mode:logical
#> TRUE:82
#> NA's:32345
#>
#>
#>
#>

```

Este archivo contiene en total 32.427 registros, y proporciona 17 campos asociados a cada registro. Los metadatos de esta información están disponibles en el Portal de datos abiertos del Ayuntamiento de Madrid.

Entre los campos disponibles, podemos destacar los siguientes:

- **distrito**: Nos proporciona el distrito de Madrid en el que se produjo el accidente.
- **fecha**: En la que se produjo el accidente.
- **coordenada\_x\_utm, coordenada\_y\_utm**: Son las coordenadas en las que se produjo el accidente.

#### Q1: ¿En qué CRS se encuentran las coordenadas?

Cuadro 5: Accidentes de tráfico en Madrid 2020: Coordenadas

coordenada_x_utm	coordenada_y_utm
444597.8	4475156
444597.8	4475156
447710.6	4477156
447710.6	4477156
445076.3	4478372
445076.3	4478372

Si nos fijamos, las coordenadas no parecen corresponder con longitudes o latitudes, ya que como se explicó el rango posible de valores es  $[-180, 180]$  (para longitudes) y  $[-90, 90]$  (para latitudes):

Tenemos una indicación en el propio nombre de la variable: “UTM” se corresponde con las siglas de “Universal Transverse Mercator”, otra proyección muy utilizada en GIS. En la página web del Ministerio de Agricultura, Pesca y Alimentación podemos comprobar los códigos EPSG habitualmente empleados en la cartografía de España, entre la que se incluyen varias proyecciones UTM.

Como nota, el sistema geodésico de referencia oficial en España es el sistema ETRS89<sup>2</sup>, y el huso UTM de Madrid es el 30 N<sup>3</sup>, por lo que vamos a probar con el código EPSG 25830, que corresponde con la proyección ETRS89 / UTM zone 30N:

```
library(sf)
# Objeto sf sin CRS

accidentes2020_sf <- st_as_sf(
  accidentes2020,
  coords = c(
    "coordenada_x_utm",
    "coordenada_y_utm"
  ),
  crs = st_crs(25830)
)

# Comprobamos con la geometría de Madrid

library(mapSpain)
library(ggplot2)
madrid <- esp_get_munic(munic = "Madrid") %>%
  st_transform(25830)

# Usamos imagen como mapa de fondo
tile <- esp_getTiles(madrid, "IDErioja", zoommin = 2)

ggplot() +
  layer_spatraster(tile) +
  geom_sf(
    data = accidentes2020_sf,
    col = "blue",
```

<sup>2</sup><https://www.mitma.gob.es/organos-colegiados/consejo-superior-geografico/csg/etrs89/etrs89-nuevo-sistema-de-referencia-geodesico-oficial-en-espana>

<sup>3</sup>Se puede localizar la zona UTM de una localización mediante la siguiente web: <https://mangomap.com/robertyoung/maps/69585/what-utm-zone-am-i-in-#>

```

    size = 0.3,
    alpha = 0.3
)

```

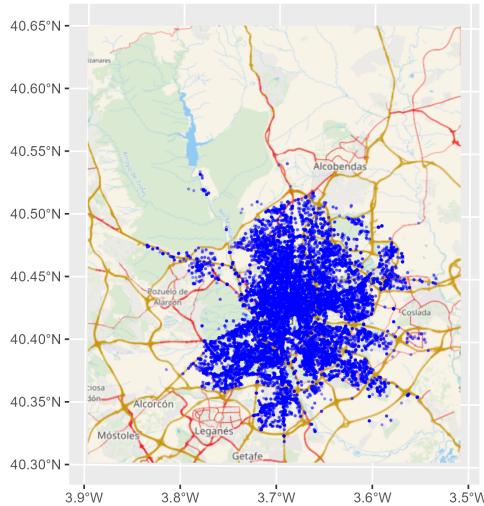


Figura 43: Accidentes de Tráfico en Madrid (2020)

## Q2: ¿Hay algún patrón en la ocurrencia de accidentes de tráfico?

GEMA: CAMBIA LO QUE CONSIDERES, QUE DE TODO ESTO NO TENGO NI IDEA

En la figura 43 podemos intuir ciertos patrones en la ocurrencia de accidentes. Por ejemplo, apenas se producen accidentes en la Casa de Campo o en el Monte del Pardo, y parece observarse cierta concentración en las autopistas de salida de la ciudad

Para el siguiente análisis, vamos a analizar el patrón de accidentes en el mes de febrero.

```

acc_feb <- accidentes2020_sf %>%
  filter(
    fecha >= "2020-02-01",
    fecha < "2020-03-01"
  )

ggplot() +
  layer_spatraster(tile) +
  geom_sf(
    data = accidentes2020_sf,
    col = "blue",
    size = 0.3,
    alpha = 0.3
  )

```

## Tarea 3: Análisis de patrones with spatstat

El paquete **spatstat** (**GEMA: REFERENCIA;?**) está diseñado para realizar este tipo de análisis.

Además, necesitaremos una ventana de observación (owin). Podemos en este caso obtener los datos espaciales de los distritos de Madrid en la siguiente dirección <https://datos.madrid.es/portal/site/egob/menuitem>.

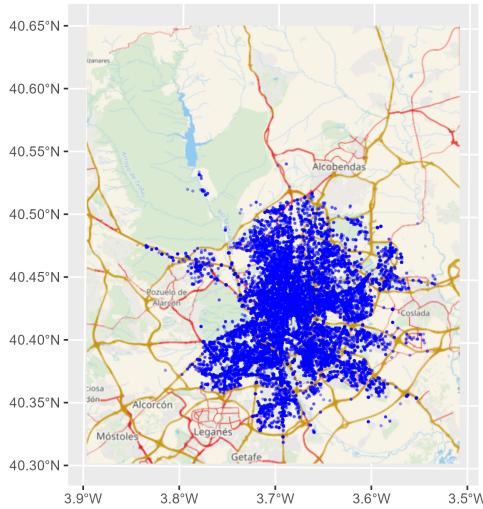


Figura 44: Accidentes de Tráfico en Madrid (Feb-2020)

c05c1f754a33a9fbe4b2e4b284f1a5a0/?vgnnextoid=7d6e5eb0d73a7710VgnVCM2000001f4a900aRCRD&vgnextchannel=374512b9ace9f310VgnVCM100000171f5a0aRCRD&vgnextfmt=default

Siguiendo el anterior ejemplo, vamos a analizar el patrón de accidentes en el mes de febrero. Además, en el análisis de patrones de puntos es necesario delimitar la ventana espacial de observación. En este caso será el municipio de Madrid.

```
library(spatstat)
# Necesitamos un recinto de observación: owin
mad_owin <- as.owin(madrid)

# Seleccionamos del data.frame inicial

acc_feb_df <- accidentes2020 %>%
  filter(
    fecha >= "2020-02-01",
    fecha < "2020-03-01"
  )

# Creamos el objeto ppp
# Es necesario que todas las coordenadas estén proyectadas en el mismo CRS!
feb_ppp <- ppp(
  x = acc_feb_df$coordenada_x_utm,
  y = acc_feb_df$coordenada_y_utm,
  window = mad_owin
)

summary(feb_ppp)
#> Planar point pattern: 4047 points
#> Average intensity 6.698035e-06 points per square unit
#>
#> *Pattern contains duplicated points*
#>
#> Coordinates are given to 1 decimal place
```

```
#> i.e. rounded to the nearest multiple of 0.1 units
#>
#> Window: polygonal boundary
#> single connected closed polygon with 118 vertices
#> enclosing rectangle: [424891.6, 455899.8] x [4462568, 4499231] units
#> (31010 x 36660 units)
#> Window area = 604207000 square units
#> Fraction of frame area: 0.531
```

**Q3: ¿Cuál es la intensidad de los accidentes?**

(GEMA: ESTO ES COPIADO TAL CUAL DEL PAPER!!!)

```
diggle_feb <- bw.diggle(feb_ppp)
scott_feb <- bw.scott(feb_ppp)

dens_diggle <- density.ppp(feb_ppp, sigma = diggle_feb)
dens_scott <- density.ppp(feb_ppp, sigma = scott_feb)
dens_300 <- density.ppp(feb_ppp, sigma = 300)
dens_500 <- density.ppp(feb_ppp, sigma = 500)

par(mfrow = c(2, 2), mar = c(1, 1, 1, 1))
plot(dens_diggle)
plot(dens_scott)
plot(dens_300)
plot(dens_500)
```

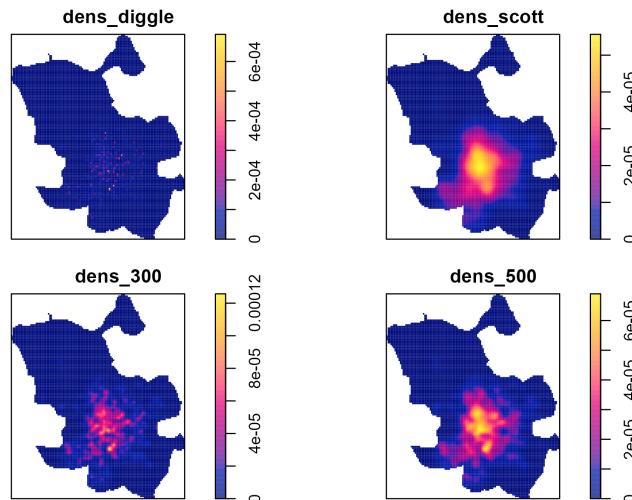


Figura 45: Intensidad de accidentes de tráfico

(GEMA: ESTO ES PARA CONVERTIR ppp a otros formatos, vale la pena?)

```
# Convertimos a otros objetos
library(raster)

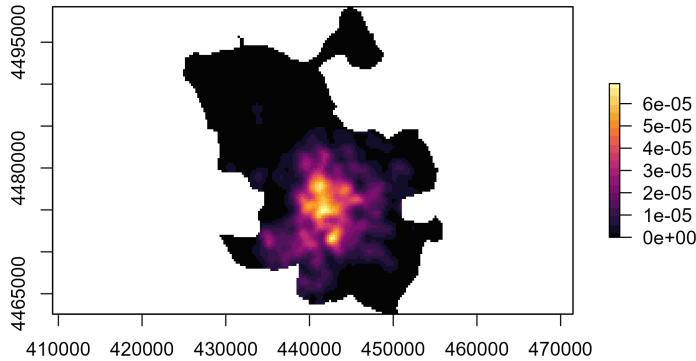
dens_500_rast <- raster(dens_500)
```

```

crs(dens_500_rast) <- raster::crs(st_crs(accidentes2020_sf)$proj4string)

plot(dens_500_rast, col = hcl.colors(20, "Inferno"))

```



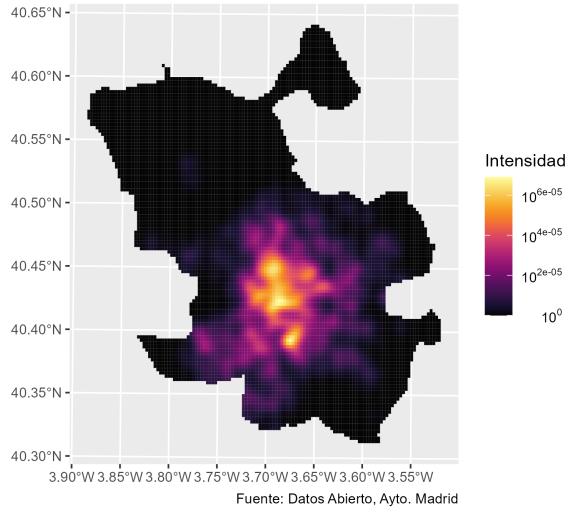
```

# A poligonos sf
pols <- st_as_sf(rasterToPolygons(dens_500_rast))
class(pols)
#> [1] "sf"           "data.frame"

# Con ggplot
library(ggplot2)

ggplot(pols) +
  geom_sf(aes(fill = layer), col = NA, size = 0.01) +
  scale_fill_gradientn(
    colors = hcl.colors(20, "Inferno"),
    labels = scales::label_math()
  ) +
  labs(
    fill = "Intensidad",
    caption = "Fuente: Datos Abierto, Ayto. Madrid"
  )

```



## 6 Extensiones

### Referencias

- Abrahart, R. J., Openshaw, S., Abrahart, R. J., & See, L. M. (Eds.). (2000). *Geocomputation*. CRC Press. <https://doi.org/10.4324/9780203305805>
- Cressie, N. A. C. (1993). *Statistics for Spatial Data*. John Wiley & Sons, Inc. <https://doi.org/10.1002/9781119115151>
- Lovelace, R., Nowosad, J., & Muenchow, J. (2019). *Geocomputation with R*. CRC Press.
- Pizarro, M., Hernangómez, D., & Fernández-Avilés, G. (2021). *climaemet: Climate AEMET Tools*. <https://doi.org/10.5281/zenodo.5205573>
- Rees, I., P y Turton. (1998). Guest Editorial. *Environment and Planning A: Economy and Space*, 30(10), 1835-1838. <https://doi.org/10.1068/a301835>
- Walker, K. (2021). *crsuggest: Obtain Suggested Coordinate Reference System Information for Spatial Data*.