

1 Capítulo 1: Mapeadores y mapeados

1.1 Introducción

Hasta hace relativamente poco, mapear era trabajo de especialistas y, lo que se mapeaba, eran cuestiones relevantes: el palacio municipal, la iglesia más representativa, la escuela y no muchas más cosas¹. Sin embargo, hoy por hoy somos mapeadores casi a diario gracias a los teléfonos celulares que tenemos constantemente en el bolsillo. Por ejemplo, la Figura 1 es el recorrido del 28 de Mayo de 2019 entre los barrios de San Telmo y Microcentro en CABA².

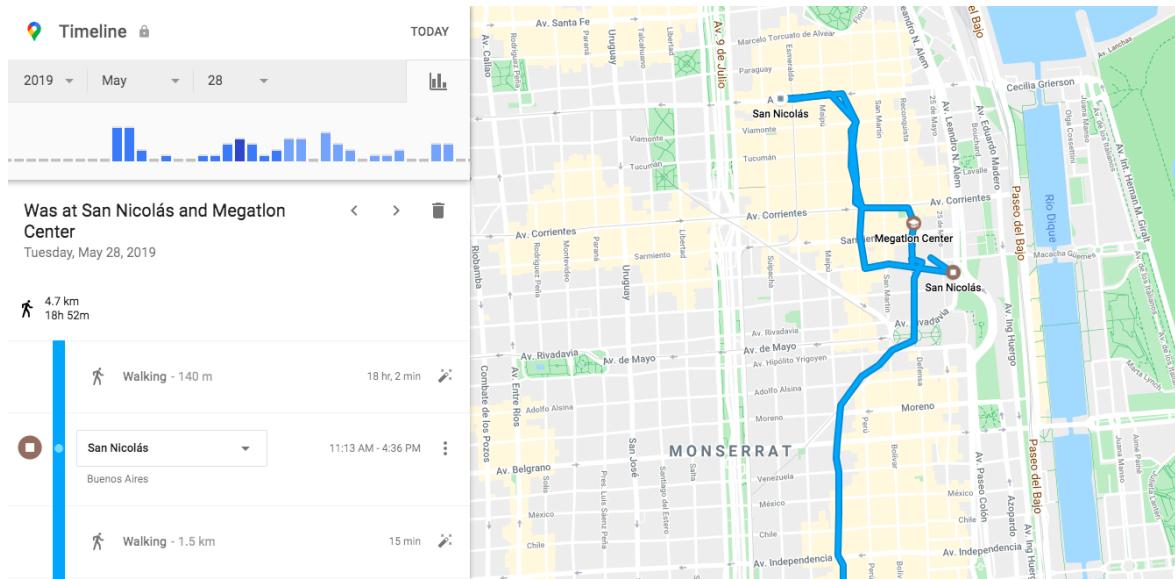


Figure 1: “Google Maps”

1.2 Sistemas de coordenadas y proyecciones

Por más que hacer un mapa hoy por hoy es relativamente sencillo, hay algunas cosas interesantes que antes se tuvieron que resolver. Por ejemplo: ¿Cómo hacemos un mapa lo más realista posible? ¿Cómo representamos en 2 dimensiones una forma geométrica tridimensional? Y si a eso le agregamos que esa forma geométrica no es una esfera perfecta sino que, a su vez, tiene una forma irregular, más “achatada” hacia los polos³.

Sobre esas preguntas vamos a tratar en esta sección con los Sistemas de Coordenadas y las Proyecciones.

1.2.1 Sistemas de Coordenadas de Referencia

Los Sistemas de Coordenadas de Referencia (*CRS*, en inglés) identifican cualquier ubicación en la superficie de la tierra utilizando dos números: funcionan como direcciones.

El tipo de CRS más conocido es el que usa latitud y longitud, para definir posiciones en los ejes norte-sur y este-oeste. La latitud y la longitud son medidas de ángulos, no de distancia. La **latitud** es el ángulo entre el Ecuador y el paralelo en el cual el punto que estamos midiendo cae. La **longitud** es el ángulo entre el Meridiano de Greenwich y el meridiano donde se encuentra nuestra medición.

¹ Recomendación: Video de IPGH.oficial - Así es cómo se hacían los mapas en 1961.

² Entrá en <https://www.google.com/maps/timeline> y sorprendente... o asustate!

³ Video recomendado: Why all world maps are wrong

1.2.2 Proyecciones

Una proyección se utiliza para representar la tierra en una superficie plana. Un **Sistema de Proyección de Coordenadas** es un sistema que define la ubicación en un mapa basado en las coordenadas de x y de y. En este proceso de transladar de la superficie tridimensional de la tierra a un plano se generan ciertas distorsiones en propiedades de la suerficie de la tierra, tales como: área, dirección, distancia y forma.

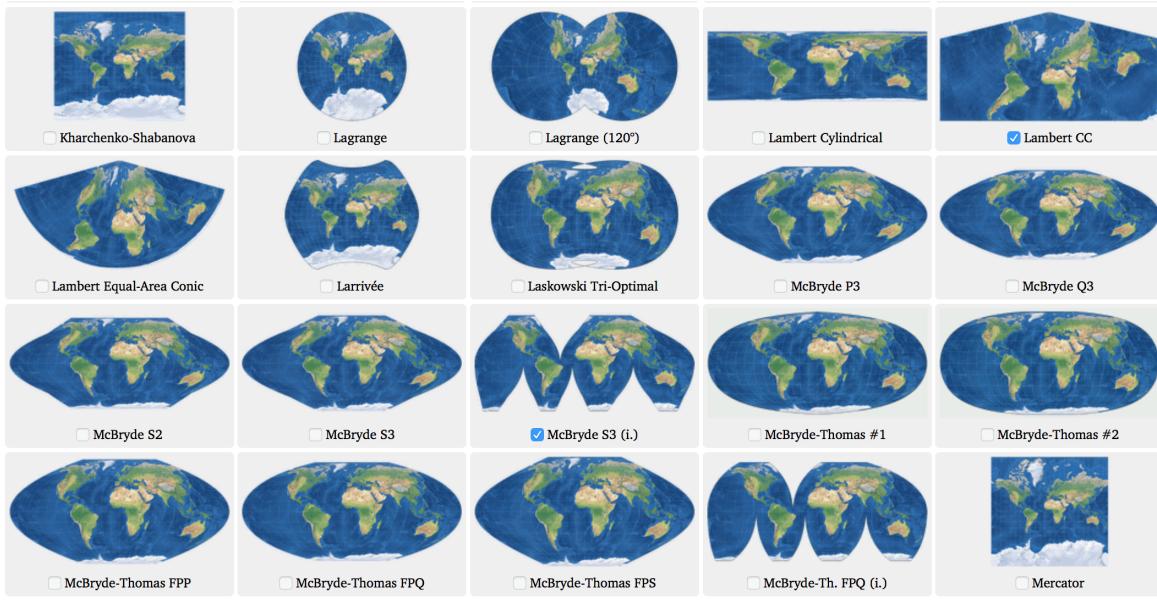
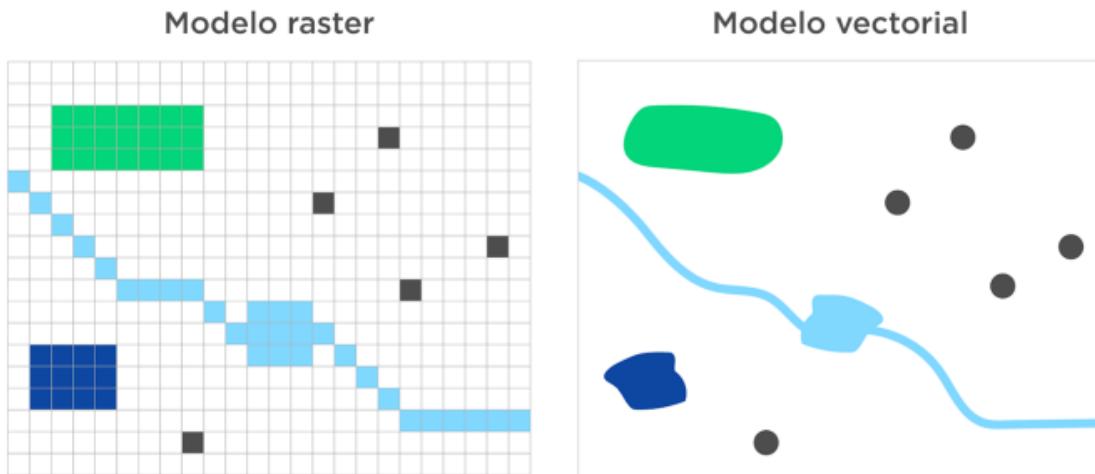


Figure 2: “Distintos tipos de proyecciones”

1.2.2.1 Proyección Mercator La proyección Mercator es muy utilizada hoy en día, pero sus origines está en un mapa que dibujó Gerardus Mercator en el Siglo XVI. Su popularidad, en esa época, radicaba (y radica) en su facilidad para navegar ya que **no distorciona ni las formas ni las direcciones**. Sin embargo, su problema es que distorciona las áreas de los países: cuanto más cerca de los polos, más distorciónada el área.

1.3 Modelos de datos

Los **Sistemas de Información Geográfica** modelan la realidad territorial para convertirla en datos geográficos. Por ejemplo, la ubicación de una escuela, se traduce en un punto con latitud y longitud que podemos visualizar en un mapa. Para modelar la realidad territorial, los SIG utilizan la representación raster o vectorial.



El **modelo raster** consta de una matriz de celdas (o píxeles) organizadas en filas y columnas (o una cuadrícula) en la que cada celda contiene un valor que representa información, como la temperatura, por ejemplo. Las fotografías aéreas digitales o las imágenes de satelitales son ejemplos de este tipo de modelo.

Los **modelos de representación vectorial** modelizan los datos utilizando formas geométricas básicas: puntos, líneas y polígonos. Las geometrías son enriquecidas con los atributos temáticos de los fenómenos que representan. Por ejemplo, los cursos de agua, son modelizados a través de polilíneas (muchas líneas), y poseen atributos como el nombre y categoría, el régimen hídrico, el caudal anual, entre otros.

1.4 Formato de los archivos

Hay distintos tipos de archivos para trabajar con los datos geográficos: los famosos **shapefiles** y los modernos **GeoJSON**. Respecto a los primeros, hay que decir que es un formato desarrollado por la empresa ESRI (los creadores de *ArcGIS*) y que tienen dos grandes dificultades: en primer lugar, no hay un único archivo, sino que son varios archivos individuales en **.zip**. Cada uno guarda una parte de la información: por ejemplo, la extensión **.dbf** guarda los datos, la extensión **.shp** las formas, etc. En segundo lugar, el nombre de las variables está delimitado a 10 caracteres, que nos hace ejercitarnuestra imaginación al interpretar los nombres de las variables.

Por otro lado, está el formato **GeoJSON** que es un formato abierto que es más liviano y fácil de utilizar. Nosotros a lo largo del curso usaremos de los dos formatos.

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [125.6, 10.1]
  },
  "properties": {
    "name": "Dinagat Islands"
  }
}
```

Figure 3: “Ejemplo de GeoJSON”

Por último, hay datos que suelen venir en otros formatos, tales como un archivo plano (un `.csv` o un `.xlsx`). A lo largo del curso también los iremos incorporando en nuestro trabajo.

2. Datos geográficos en R y modelo vectorial

En este capítulo vamos a empezar a trabajar con datos geográficos que pertenecen al Modelo de Representación Vectorial con lo que vamos a estar trabajando con puntos, líneas y polígonos. Por ser el primer capítulo, vamos a invertir un tiempo a explicar los paquetes que vamos a utilizar y los pasos que vamos llevando a cabo.

En esta capítulo vamos a utilizar los siguientes paquetes que ya deben haber utilizado en otras clases o proyectos: `{tidyverse}`, `{readxl}` y `{ggplot2}` y con dos paquetes nuevos: `{sf}` y `{leaflet}`.

Si bien hay varios paquetes que permiten graficar datos geográficos, tales como `{tmap}`, `{mapview}`, `{base}`, nosotros vamos a utilizar `{ggplot2}` para nuestras gráficas estáticas y `{leaflet}` para nuestras gráficas interactivas.

El paquete `{sf}`⁴(hace referencia a *simple feature*) fue diseñado para trabajar con datos del *modelo vectorial*. Uno de sus mayores atractivos es que permite tratar a los datos geográficos de igual forma que a los datos no geográficos: de esta forma podemos hacer las mismas operaciones que haríamos con un data frame común, tales como aplicar los verbos del paquete `{tidyverse}`.

2.1 Primeros pasos en R

Excelente! Arranquemos por cargar los paquetes que vamos a utilizar:

```
library(tidyverse)      # Manipulación de los datos
library(readxl)         # Abrir archivos de Excel
library(ggplot2)        # Gráficos
library(sf)              # Manipulación de datos geográficos
```

Ahora, vamos a abrir los datos con los que queremos trabajar. Vamos a trabajar con datos públicos publicados por el “Instituto Nacional de Estadística y Censos de la República Argentina” (INDEC). Vamos a bajar los shapefiles de los *departamentos de la Argentina* desde la página de INDEC o utilizando este link de descarga directa⁵.

Para cargar los datos vamos a utilizar la función `st_read()`. Notese que directamente abrimos el archivo que termina en `.shp`.

```
departamentos <- st_read("datos/Codgeo_Pais_x_dpto_con_datos/pxdptodatosok.shp")
```

```
## Reading layer 'pxdptodatosok' from data source '/Users/lauticantar/Google Drive/Clases de R/UdeSA - 1'
## Simple feature collection with 527 features and 10 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:            xmin: -74.02985 ymin: -90 xmax: -25.02314 ymax: -21.74506
## geographic CRS: WGS 84
```

Cuando cargamos el archivo geográfico, R nos muestra el recuadro de arriba. Veamos que significa cada una de las líneas:

- **Simple feature collection with 527 features and 10 fields:** estamos abriendo un dataset que contiene 527 filas y 10 columnas.

⁴Link: <https://r-spatial.github.io/sf/>

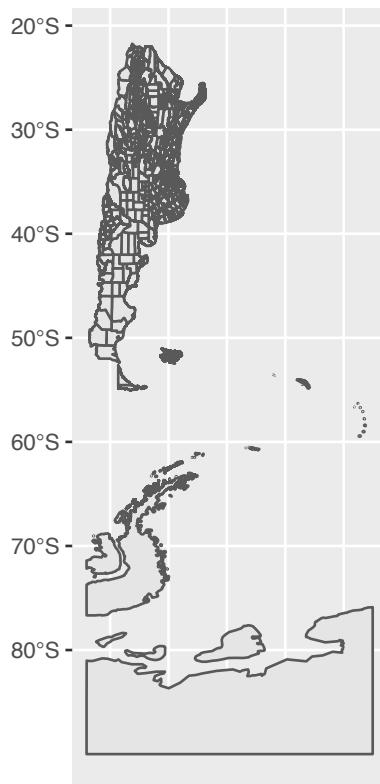
⁵Link: <https://bit.ly/2JySgCK>

- **geometry type:** MULTIPOLYGON: los archivos con información geográfica contienen colecciones de puntos, de líneas, o de polígonos y sus respectivas versiones “múltiples”: múltiples puntos, líneas o polígonos. En este caso, los departamentos son múltiples polígonos.
- **dimension:** XY: es la cantidad de dimensiones con las que se está trabajando, en este caso, dos. Si hubiese tres dimensiones, sería XYZ
- **bbox:** xmin: -74.02985 ymin: -90 xmax: -25.02314 ymax: -21.74506: bbox es la simplificación para “bounding box”, una caja delimitadora. Estos valores son la longitud mínima, latitud mínima, longitud máxima y la latitud máxima del conjunto de datos
- **geographic CRS:** WGS 84: nos brindan información referida al sistema de coordenadas de referencia (Coordinate Reference System (CRS)). Vemos que se utiliza la CRS WGS 84.

Acá tenemos dos opciones: o mapeamos primero y exploramos los datos después o, al revés. Vamos por la primera opción para concentrarnos en la exploración de datos más adelante.

Para nuestro primer mapa rapidito vamos a utilizar el paquete `{ggplot}` y la función `geom_sf()`:

```
ggplot() +
  geom_sf(data = departamentos)
```



Los archivos geográficos, además de la información que nos permite hacer los mapas, suelen traer información sobre cada una de las formas: esa información se llama “*atributos*”. Veamos entonces cuáles son los atributos que tiene el dataset `departamentos`. Acá empezamos a ver las bondades del paquete `{sf}` en la manipulación de datos. Hagamos un `summary()` de nuestro dataset:

```
summary(departamentos)
```

```
##      link      codpcia departamen      provincia
##  Length:527    Length:527  Length:527    Length:527
##  Class :character  Class :character  Class :character  Class :character
##  Mode  :character  Mode  :character  Mode  :character  Mode  :character
##  mujeres      varones     personas      hogares
##  Length:527    Length:527  Length:527    Length:527
##  Class :character  Class :character  Class :character  Class :character
##  Mode  :character  Mode  :character  Mode  :character  Mode  :character
##  viv_part      viv_part_h      geometry
##  Length:527    Length:527  MULTIPOLYGON :527
##  Class :character  Class :character  epsg:4326    : 0
##  Mode  :character  Mode  :character  +proj=long...: 0
```

Aquí vemos que el archivo departamentos tiene las siguientes variables:

- **link**: ID del Departamento
- **departamen**: Nombre del Departamento
- **provincia**: Nombre de la Provincia
- **varon**: cantidad de varones en dicho radio censal
- **mujer**: cantidad de mujeres en dicho radio censal
- **personas**: población total
- **hogares**: cantidad total de hogares
- **viviendasp**: total de viviendas particulares
- **viv_part_h**: total de viviendas particulares habitadas
- **geometry**: información geográfica que utiliza R para hacer la visualización del mapa.

Algo interesante para mencionar respecto a esta última columna **geometry**. Esta columna estará siempre presente cuando trabajemos con datasets geográficos (que hayamos abierto con la función `st_read()`) y, a diferencia del resto de las filas y columnas del dataset, es una **lista** que contiene los valores para mapear. Si queremos dejar de contar con esos datos, tenemos que usar la función `st_drop_geometry()`.

2.2. Explorando los datos

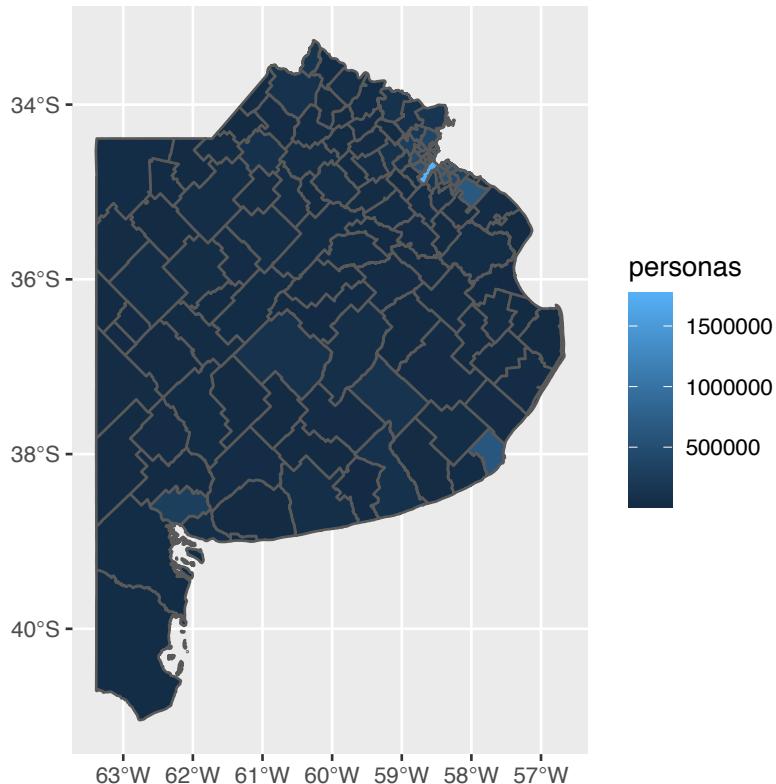
Ahora que ya abrimos el archivo, hicimos el primer mapa y vimos que información contienen los atributos del dataset **departamentos**, podemos empezar a explorar los datos y empezar a hacer distintos análisis. Empecemos por seleccionar únicamente los departamentos de la Provincia de Buenos Aires. Para eso utilizaremos la función `select()`, como cuando trabajamos con un dataset no geográfico.

```
departamentos_pba <- departamentos %>%
  filter(provincia == "Buenos Aires") %>%
  mutate(personas = personas %>% as.character() %>% as.numeric())
```

El resultado de la operación anterior es un nuevo dataset llamado **departamentos_pba** que tienen 134 filas⁶. Ahora podemos volver a graficar y, además, colorear en base a la población.

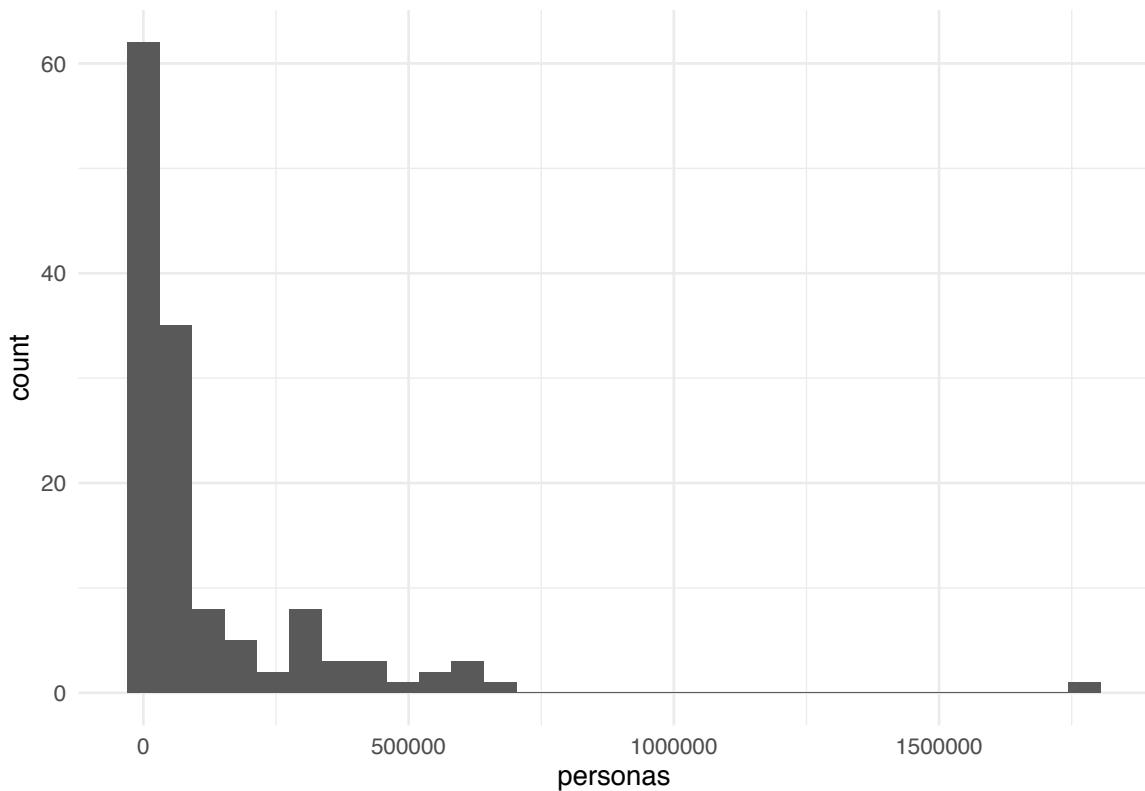
⁶En la actualidad y desde el año 2009, en la Provincia de Buenos Aires hay 135 departamentos por la división entre el Partido de Chascomús y el Partido de Lezama. La fuente de este archivo no llegó a reflejar ese cambio.

```
ggplot() +  
  geom_sf(data = departamentos_pba,  
          aes(fill = personas))
```



Cuando vemos el mapa, nos damos cuenta que el Partido de La Matanza es un outlier en la población del resto de la provincia. Esto lo podemos ver en el siguiente gráfico.

```
ggplot() +  
  geom_histogram(data = departamentos_pba,  
                 aes(x = personas)) +  
  theme_minimal()
```

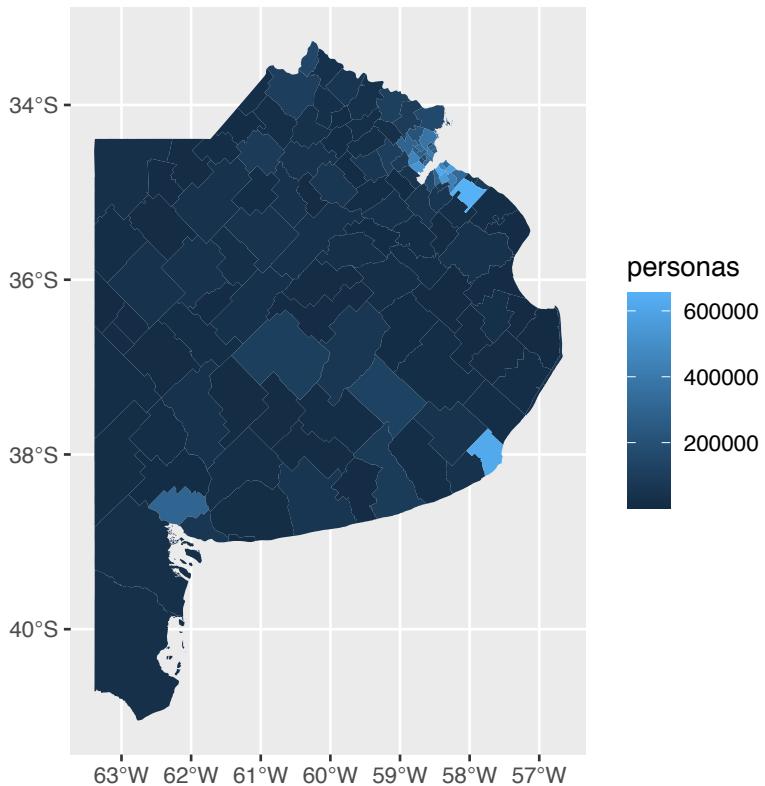


Por lo tanto, a fines puramente didácticas, hagamos el ejercicio de filtrarla momentaneamente. Y también saquemos los bordes grises de los contornos de los polígonos. Para eso dejemos únicamente los departamentos con menos de 1.000.000 de habitantes.

```
departamentos_pba_sin_LM <- departamentos_pba %>%
  filter(personas < 1000000)
```

Ouala! Ahora vamos a ver como resaltan otros municipios tales como los municipios del Área Metropolitana de Buenos Aires (AMBA), La Plata, General Pueyredón (Mar del Plata) y Bahía Blanca.

```
ggplot() +
  geom_sf(data = departamentos_pba_sin_LM,
    aes(fill = personas),
    color = NA)
```



2.3 Puntos

En la presentación vimos que en el Modelo Vectorial hay puntos, líneas y polígonos. En la sección anterior graficamos polígonos, ahora vamos a graficar puntos. Para eso, vamos a utilizar un archivo plano, un .csv que tiene dos columnas que referieren a la latitud y longitud. Vamos a utilizar un dataset de “Escuelas Primarias Estatales en Argentina”. Para eso, vamos a bajar los datos directamente desde la página donde están publicados.

```
escuelas.link <- "https://sig.planificacion.gob.ar/layers/download/mardis_escuelasprimarias_estatales/csv"
escuelas <- read.csv(escuelas.link,
                      stringsAsFactors = FALSE,
                      fileEncoding = "UTF-8")
```

Ahora, filtremos los datos de las escuelas que están en la Provincia de Buenos Aires.

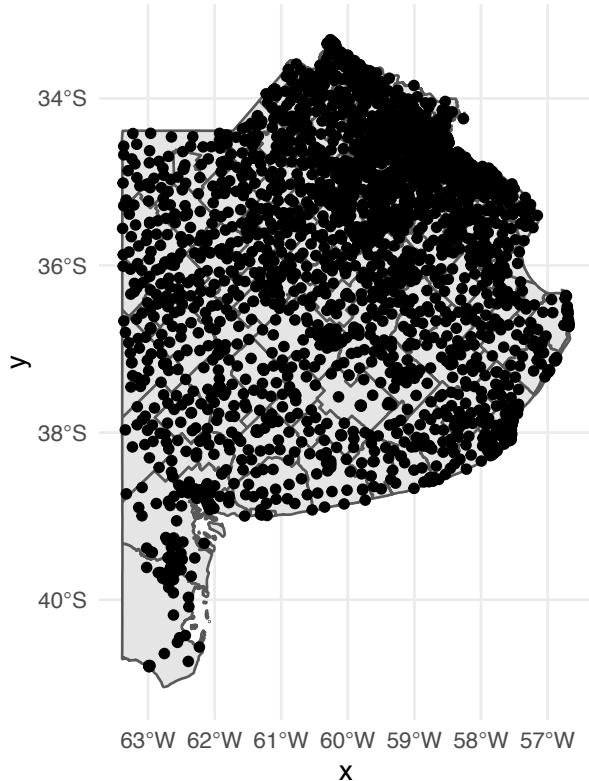
```
escuelas_pba <- escuelas %>%
  filter(provincia == "Buenos Aires")
```

Y veamos como quedan en el mapa. Vamos a usar como base el mapa de los `departamentos_pba` y vamos a utilizar dos columnas que están en el dataset que se llaman `x` e `y`, suelen hacer referencia a Longitud y Latitud respectivamente.

```

ggplot() +
  # Mapa base de la sección anterior
  geom_sf(data = departamentos_pba) +
  # Mapa de los puntos
  geom_point(data = escuelas_pba,
             aes(x = x, y = y)) +
  theme_minimal()

```



¿Cómo podemos hacer para transformar un archivo plano en un archivo geográfico? Para eso vamos a utilizar la función `st_as_sf()`. Para eso vamos a implementar el siguiente proceso: primero vamos a eliminar (si existen), todos los puntos que no tengan datos de latitud o longitud. Luego, ya en la función `st_as_sf()` uno de los parámetros (`crs = 4326`) será indicarle qué columnas hacen referencia a las coordenadas. El número 4326 es el código de la proyección Mercator.

```

escuelas_pba_shp <- escuelas_pba %>%
  filter(!is.na(x) & !is.na(y)) %>%
  st_as_sf(coords = c("x", "y"),
           crs = 4326)

```

Por último, chequeamos qué clase tiene este nuevo dataset:

```

class(escuelas_pba_shp)

## [1] "sf"          "data.frame"

```

2.4 Operaciones

En esta sección vamos a analizar algunas operaciones muy interesantes, sin embargo es una simple selección de muchas de funciones interesantes que tiene el paquete `{sf}`.

2.4.1 `st_area()`:

Esta función nos permite calcular el área de un polígono en metros cuadrados. Por ejemplo, calculemos los metros cuadrados que hay en cada uno de los municipios de la Provincia de Buenos Aires. En la primer línea, le solicito a la función `st_area()` que calcule el área y con la segunda línea le pido que la asigne en kilómetros cuadrados.

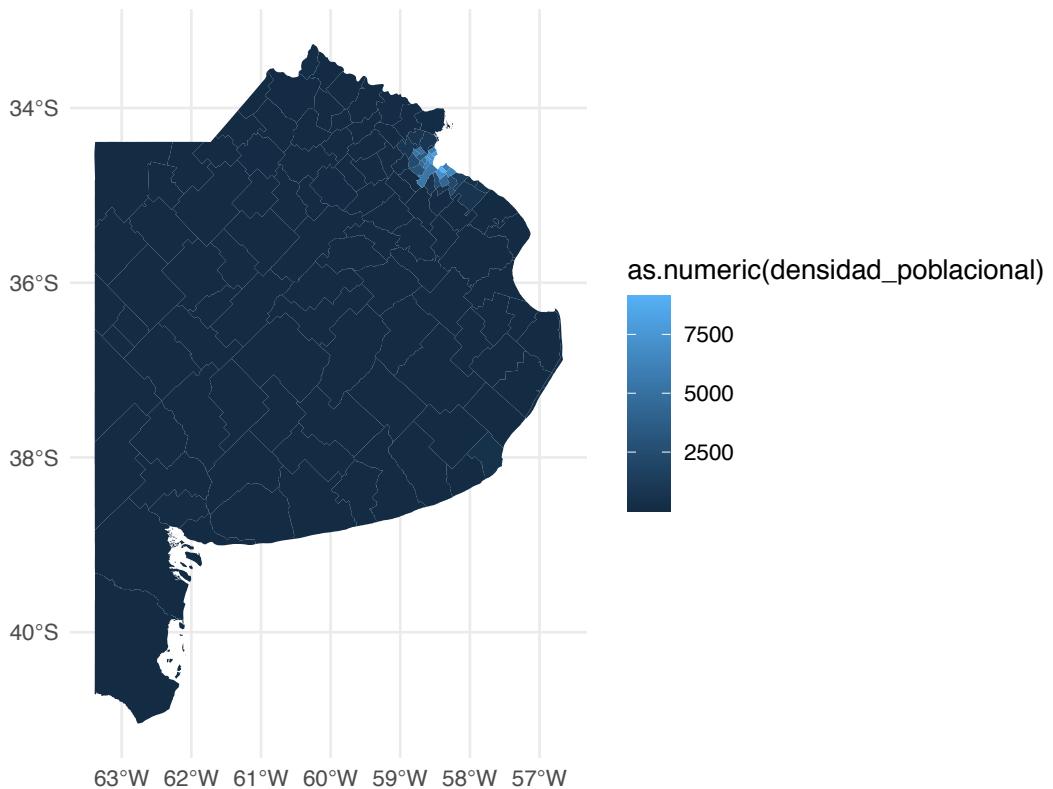
```
departamentos_pba <- departamentos_pba %>%
  mutate(area = st_area(departamentos_pba),
        area = units::set_units(area, km^2)
  )
```

Ahora podemos calcular la densidad poblacional de cada uno de los partidos:

```
departamentos_pba <- departamentos_pba %>%
  mutate(personas = personas %>% as.numeric(),
        densidad_poblacional = personas / area)
```

Y ahora graficamos nuevamente pero coloreando por la densidad poblacional.

```
ggplot() +
  geom_sf(data = departamentos_pba,
          aes(fill = as.numeric(densidad_poblacional)),
          color = NA) +
  theme_minimal()
```



Supongamos que queremos hacer foco en los departamentos que se encuentran en el Área Metropolitana de Buenos Aires. Para eso vamos a cargar el archivo `municipios_conurbano.xlsx`. Para eso vamos a utilizar la función `read_excel()` del paquete `{readxl}`.

```
amba <- read_excel("datos/municipios_conurbano.xlsx")
```

```
summary(amba)
```

```
##   Provincia          Partido          Zona      Conurbano
##   Length:33          Length:33          Length:33      Length:33
##   Class :character  Class :character  Class :character  Class :character
##   Mode  :character  Mode  :character  Mode  :character  Mode  :character
```

Y vamos a unirlo a nuestro dataset de `departamentos_pba` y filtrar por la columna `Conurbano`:

```
departamentos_amba <- departamentos_pba %>%
  # Unimos el archivo
  left_join(amba %>%
    select(Partido, Conurbano),
    by = c("departamen"= "Partido")) %>%
  # Filtramos
  filter(!is.na(Conurbano))
```

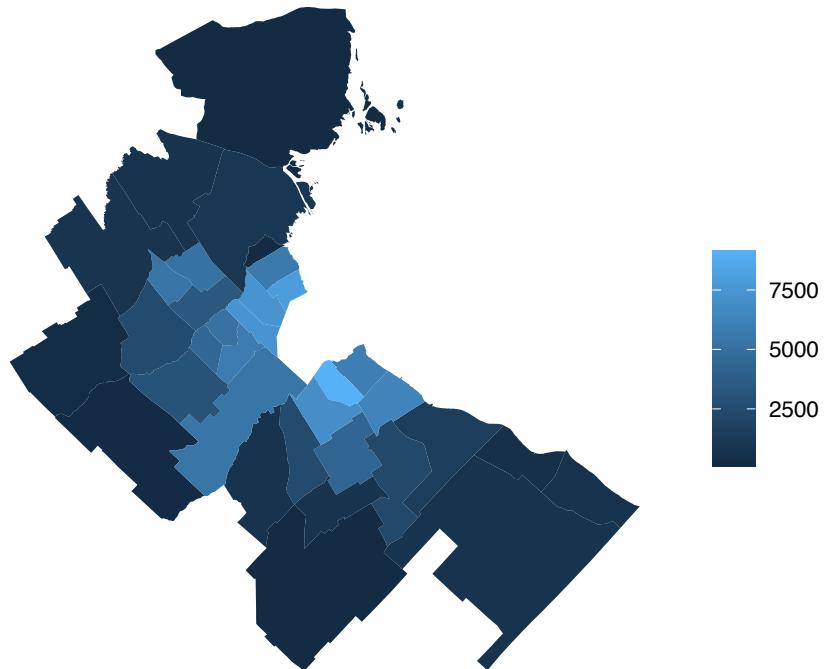
Y ahora lo podemos graficar. Aprovechamos y limpiemos un poco el gráfico. Con la función `labs()` le ponemos título y le cambiamos el nombre a la leyenda y con la función `theme()` le cambiamos la estética al mapa:

```

ggplot() +
  geom_sf(data = departamentos_amba,
           aes(fill = as.numeric(densidad_poblacional)),
           color = NA) +
  labs(title = "Densidad poblacional en los partidos del AMBA",
       fill = "") +
  theme(panel.background = element_blank(),
        axis.text.x = element_blank(),
        axis.text.y = element_blank(),
        axis.title = element_blank(),
        axis.ticks = element_blank())
)

```

Densidad poblacional en los partidos del AMBA



2.4.2 `st_intersection()`:

Una función muy interesante es la función `st_intersection()`, que nos permite superponer dos archivos geográficos y asignarle los atributos de un archivo al otro. Por ejemplo: si tenemos un archivo con puntos y otro con polígonos, podemos calcular cuántos de esos puntos caen dentro de cada polígono.

En esta sección vamos a trabajar con datos de **radios censales**. ¿Qué son los radios censales? Son una unidad geográfica que agrupa, en promedio 300 viviendas en las ciudades. Si los radios son rurales o rurales mixtos, la cantidad promedio es menor. Actualmente Argentina se compone de un total de 51.408 radios censales, diseñados y mantenidos por el INDEC. El identificador de un radio censal es un código numérico único de 9 cifras. Este código relaciona a diferentes unidades censales que pueden ordenarse jerárquicamente de modo tal que unas contienen a las otras, aunque no en todos los casos.

Los datos se descargan de la página de INDEC⁷ y se descargan por provincia. En este caso vamos a descargar los de la Provincia de Buenos Aires. Vamos a abrir utilizando nuevamente la función `st_read()`.

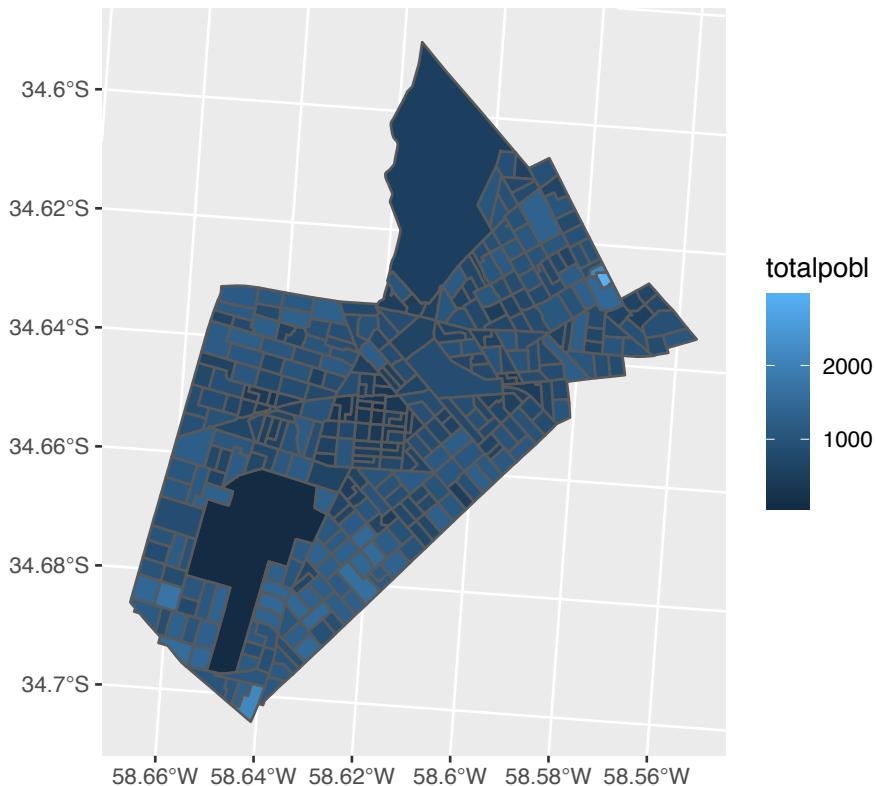
```
radios <- st_read("datos/Codgeo_Buenos_Aires_con_datos/Buenos_Aires_con_datos.shp")

## Reading layer 'Buenos_Aires_con_datos' from data source '/Users/lauticantar/Google Drive/Clases de R'
## Simple feature collection with 19577 features and 8 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:            xmin: 3721440 ymin: 5452221 xmax: 4335413 ymax: 6305225
## projected CRS: POSGAR_94_Argentina_3
```

Si volvemos a leer el mensaje que nos aparece al abrir los datos, vemos que ahora la proyección es diferente: ahora la proyección es *POSGAR_94_Argentina_3* y sobre esto vamos a trabajar en unas líneas más abajo. Para nuestro ejemplo vamos a trabajar con el Partido de Morón, cuyos radios censales comienzan con “06568”, por lo tanto hagamos un subset que llamaremos `radios_moron` y vamos a graficarlo, coloreando el total de la población:

```
radios_moron <- radios %>% filter(str_detect(link, "06568"))

ggplot() +
  geom_sf(data = radios_moron,
          aes(fill = totalpobl))
```



⁷Link: <https://www.indec.gob.ar/indec/web/Institucional-Indec-Codgeo>

Hagamos el mismo ejercicio para las escuelas de Morón: filtremos aquellas que se encuentran en las localidades que comienzan con “06568” y grafiquemos sobre nuestro mapa de los radios censales.

```
escuelas_moron <- escuelas_pba_shp %>%
  filter(str_detect(cod_loc, "6568"))

ggplot() +
  geom_sf(data = radios_moron) +
  geom_sf(data = escuelas_moron) +
  theme_minimal()
```



La función `st_intersection()` nos pide dos requisitos: que los dosa archivos sean archivos geográficos y que posean la misma proyección, caso contrario, nos dará un error. Por lo tanto, controlaremos la proyección de ambos archivos:

```
# Controlando que tengan la misma proyección
st_crs(escuelas_moron)

## 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["unknown"],
##               AREA["World"],
##               BBOX[-90,-180,90,180]],
##           ID["EPSG",4326]

st_crs(radios_moron)

## Coordinate Reference System:
##   User input: POSGAR_94_Argentina_3
##   wkt:
##     PROJCRS["POSGAR_94_Argentina_3",
##           BASEGEOGCRS["GCS_POSGAR 94",
##                   DATUM["Posiciones Geodesicas Argentinas 1994",
##                         ELLIPSOID["WGS 84",6378137,298.257223563,
##                               LENGTHUNIT["metre",1]],
##                         ID["EPSG",6694]],
##                   PRIMEM["Greenwich",0,
##                         ANGLEUNIT["Degree",0.0174532925199433]]],
##           CONVERSION["unnamed",
##                   METHOD["Transverse Mercator",
##                         ID["EPSG",9807]],
##                   PARAMETER["Latitude of natural origin",-90,
##                             ANGLEUNIT["Degree",0.0174532925199433],
##                             ID["EPSG",8801]],
##                   PARAMETER["Longitude of natural origin",-66,
##                             ANGLEUNIT["Degree",0.0174532925199433],
##                             ID["EPSG",8802]],
##                   PARAMETER["Scale factor at natural origin",1,
##                             SCALEUNIT["unity",1],
##                             ID["EPSG",8805]],
##                   PARAMETER["False easting",3500000,
##                             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",9002]]]

```

```
##           ID["EPSG",9001]]])
```

Vemos que `escuelas_moron` tiene una proyección WSG 84 (Mercator) y `radios_moron` tiene otra proyección. Por lo tanto, para poder utilizar la función `st_intersection()`, tenemos que cambiar la proyección de alguno de los dos datasets. Para eso utilizaremos la función `st_transform()`.

```
radios_moron <- st_transform(radios_moron, crs = 4326)
```

Ahora si estamos en condiciones de hacer la intersección. Para eso creamos un nuevo dataset que llamaremos `inter`

```
# Creando la interseccion
inter <- st_intersection(radios_moron, escuelas_moron)

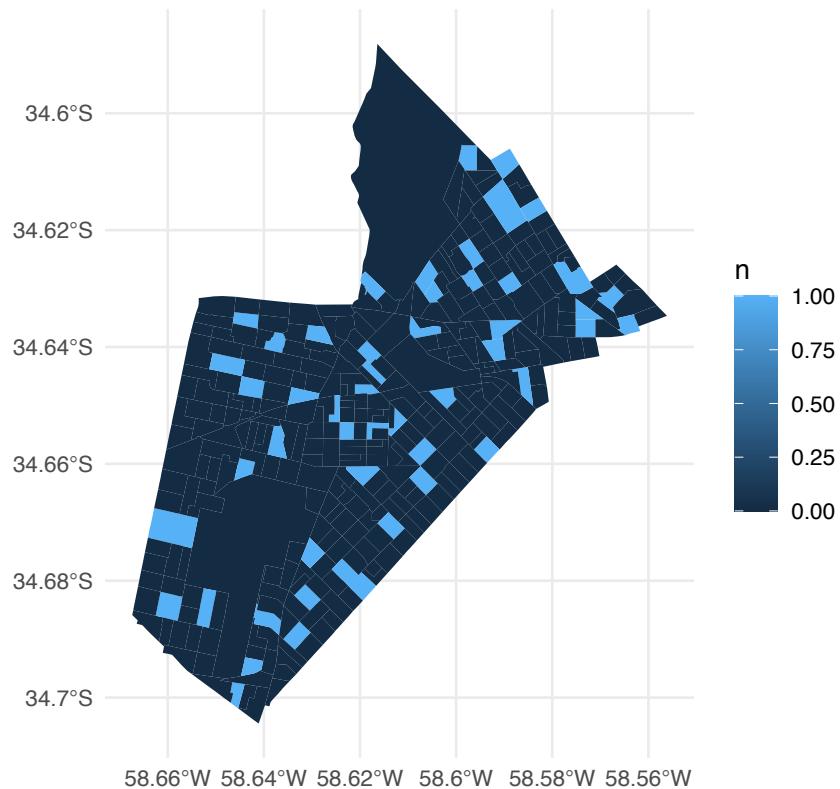
inter <- inter %>%
  group_by(link) %>%
  tally() %>%
  st_drop_geometry()
```

Y ahora unimos el dataset `inter` a nuestro archivo de `radios_moron` y aprovechamos para editar los datos.

```
radios_moron <- radios_moron %>%
  left_join(inter %>% select(link, n)) %>%
  mutate(n = n %>% as.numeric(),
        n = case_when(is.na(n) ~ 0,
                      TRUE ~ n))
```

Por último, graficamos el dataset final coloreando aquellos radios censales que tienen escuelas dentro de su área. En nuestro ejemplo, vemos que hay las escuelas están distribuidas en distintos radios censales, por eso el mapa nos queda de dos colores únicamente.

```
ggplot() +
  geom_sf(data = radios_moron,
          aes(fill = n),
          color = NA) +
  theme_minimal()
```



2.4.3 st_buffer():

La función `st_buffer()` es una forma geográfica que rodea a otra forma geográfica a una determinada distancia. Por ejemplo, un círculo que rodea a un punto a 200 metros a la redonda. Es una forma que se utiliza para análisis de proximidad.

Supongamos que queremos crear un círculo de 500 metros alrededor de 10 escuelas en Morón donde no queremos que haya kioscos. Por lo tanto, seleccionemos 10 escuelas:

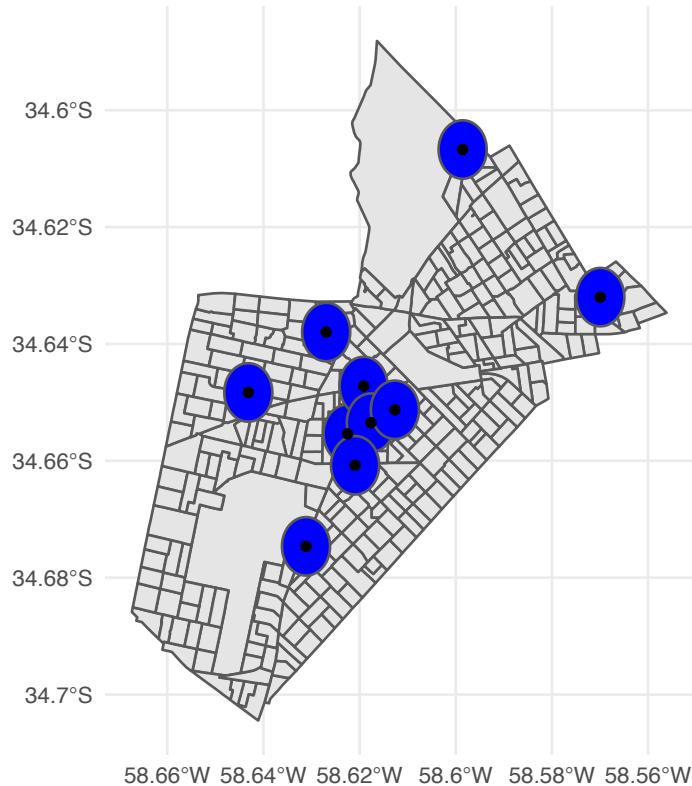
```
escuelas_10 <- escuelas_moron[1:10,]
```

Y creamos un buffer alrededor de cada una de esas escuelas:

```
buffers_escuelas <- st_buffer(escuelas_10, dist = .005)
```

Ahora, veamos como quedan en el mapa:

```
ggplot() +
  geom_sf(data = radios_moron) +
  # Graficando los buffers
  geom_sf(data = buffers_escuelas,
          fill = "blue") +
  # Graficando las escuelas
  geom_sf(data = escuelas_10,
          color = "black") +
  theme_minimal()
```



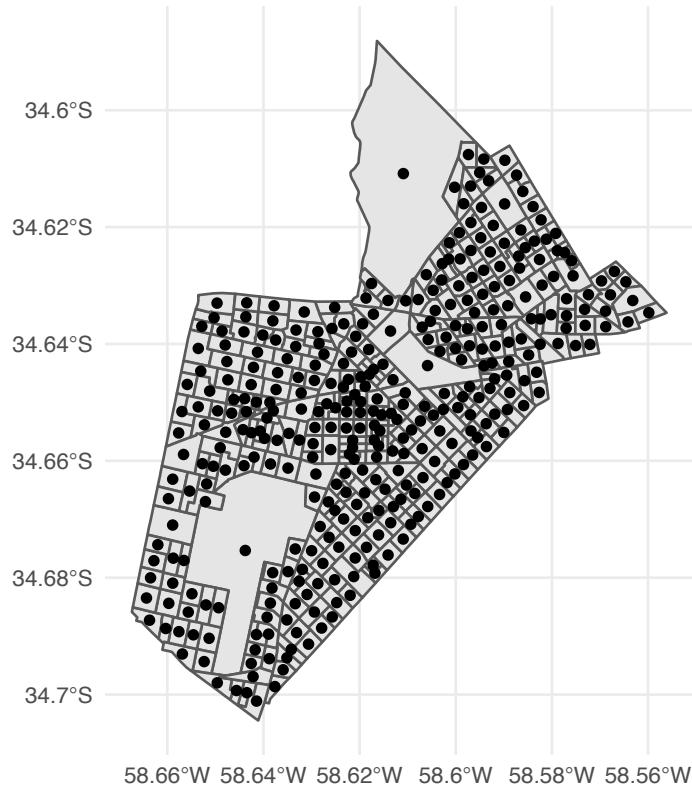
2.4.4 st_centroid():

La función `st_centroid()` nos devuelve como par de coordenadas el epicentro de un polígono. Por ejemplo, supongamos que queremos saber cuál es el centro de cada uno de los radios censales de Morón. Para eso utilizamos el siguiente código:

```
centroides_moron <- st_centroid(radios_moron)
```

Y ahora lo vemos gráficamente:

```
ggplot() +
  geom_sf(data = radios_moron) +
  geom_sf(data = centroides_moron) +
  theme_minimal()
```



2.4.5 `st_distance()`:

Por último, vamos a analizar la función `st_distance()`. Esta función nos permite calcular la distancia lineal entre dos puntos de interés. Por ejemplo, supongamos que queremos calcular la distancia de una escuela con un radio censal en particular con fines de hacer un asignación más eficiente de la matrícula escolar.

Entonces, seleccionemos una escuela y un radio censal:

```
centroide_1 <- centroides_moron[1,]
escuela_1 <- escuelas_moron[1,]
```

Y ahora calculemos la distancia entre ellos: vemos que la distancia entre ambos es de 3958 metros.

```
st_distance(centroide_1, escuela_1)

## Units: [m]
##          [,1]
## [1,] 3958.056
```

2.5 Mapas interactivos

Llegando hacia el fin de este capítulo, vamos a realizar algunos mapas interactivos y, para eso, vamos a utilizar el paquete `{leaflet}`⁸. Esta librería es una librería de JavaScript que fue adaptada para poder ser utilizada en R nos va a permitir trabajar con mapas interactivos.

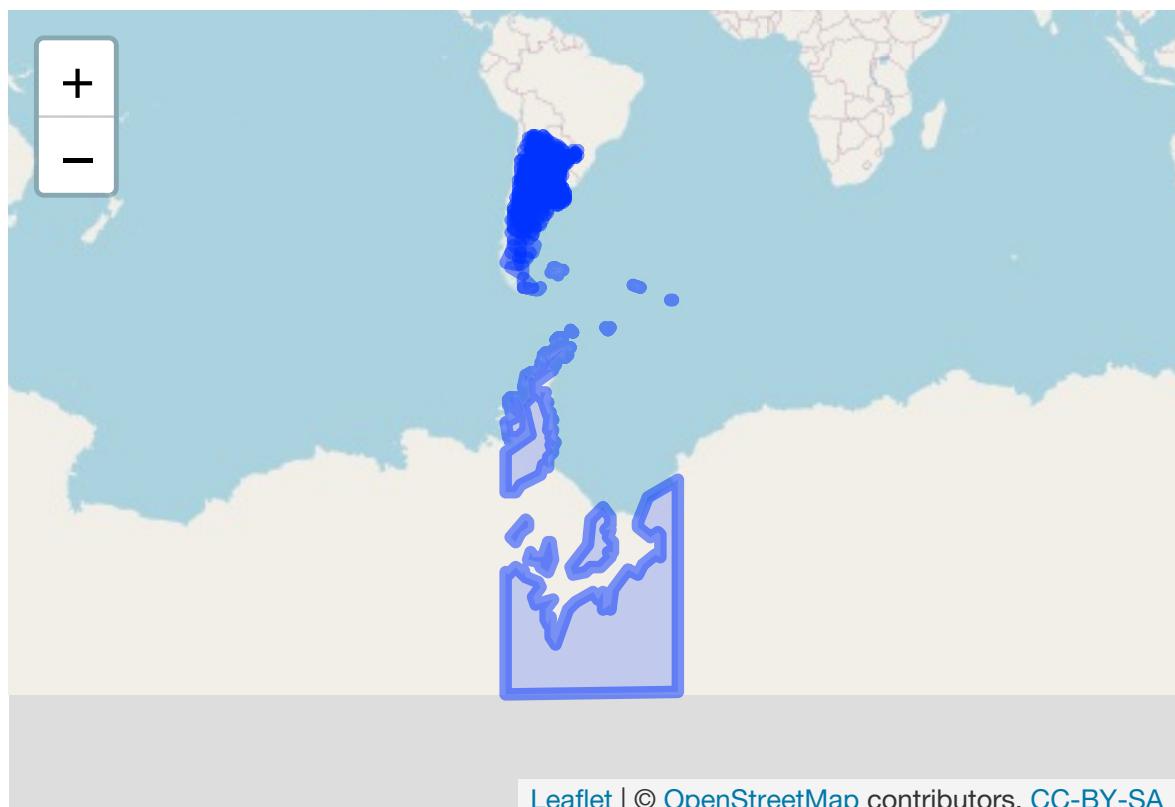
⁸Link a la página: <https://rstudio.github.io/leaflet/>

Para empezar a trabajar, vamos a cargar el paquete:

```
library(leaflet)      # Para realizar mapas interactivos
```

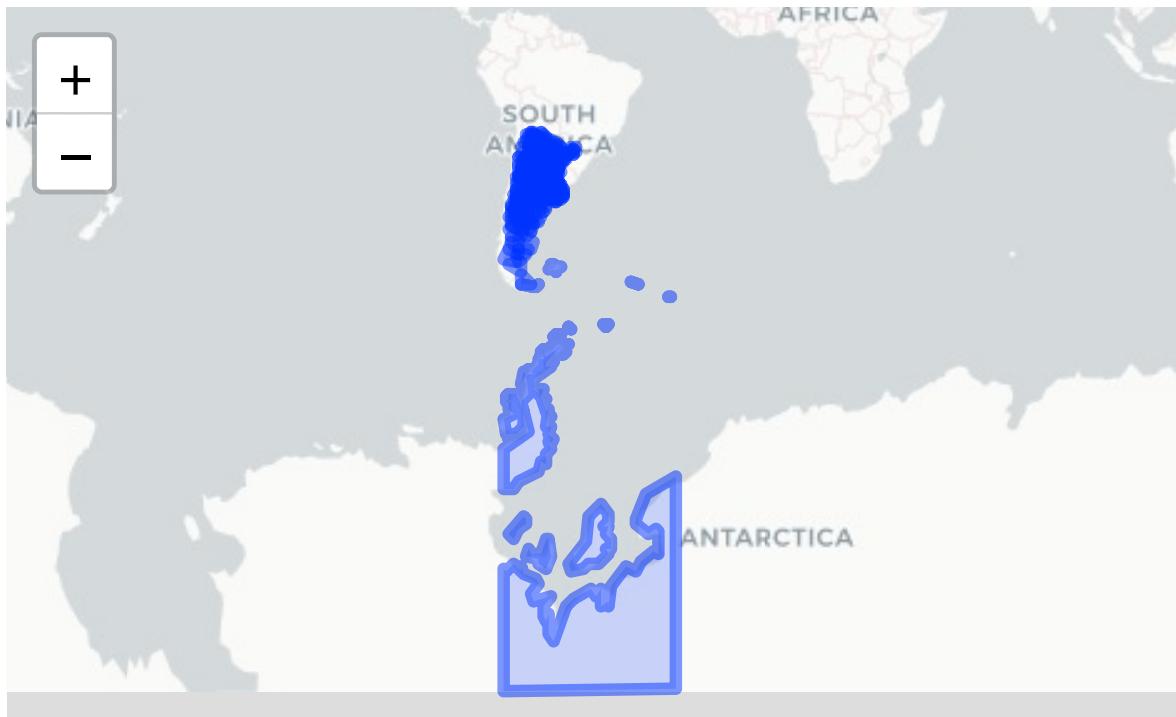
Para nuestro primer mapa interactivo, vamos a utilizar nuestro dataset original `departamentos`. Para agregar datos geográficos que son polígonos, vamos a usar la función `addPolygons()`. Ahí veremos que nuevamente tenemos el mapa de la República Argentina pero ahora podemos movernos por el mapa y hacer zoom.

```
leaflet() %>%  
  addTiles() %>%  
  addPolygons(data = departamentos)
```



El mapa base de `{leaflet}` es un mapa de Open Street Map y es lo que estamos haciendo cuando usamos la función `addTiles()`. Sin embargo, `{leaflet}` ofrece una amplia gama de mapas base diferentes que se pueden utilizar cuando usamos la función `addProviderTiles()`. Por ejemplo, en nuestro nuevo mapa queremos usar un mapa base gris de la empresa “Carto DB” y para eso utilizamos el siguiente código:

```
leaflet() %>%  
  addTiles() %>%  
  addProviderTiles("CartoDB.Positron") %>%  
  addPolygons(data = departamentos)
```



Leaflet | © OpenStreetMap contributors, CC-BY-SA, © OpenStreetMap contributors © CARTO

Algo que hay tener en cuenta, sobre todo cuando se trabaja desde ámbitos públicos en Argentina, es que los mapas base de Leaflet suelen estar escritos en inglés y a las Islas Malvinas las llaman “*Falkland Islands*”. Es por eso que el Instituto Geográfico Nacional creó un mapa base donde se soluciona este problema. En este link encontrarán más información.

Estos mapas son fácilmente exportables ya que hay una amplia variedad de paquetes disponibles para eso. La forma más sencilla, es *tejer* un archivo .Rmd en formato HTML y ahí el mapa se podrá utilizar en su versión interactiva. Otra opción muy popular es crear una *Shiny app* (una aplicación interactiva) utilizando el paquete {shiny}⁹.

⁹Link a la página: <https://shiny.rstudio.com/>