

Clase 4 - Visualizacion y Geocomputación

Demian Zayat

6/25/2019

En esta clase vamos a ver una introducción a la semántica de gráficos (paquete ggplot, parte de tidyverse) y una introducción a los mapas (ggplot)

A. Visualizaciones

Las visualizaciones son unos de los puntos fuertes de R, toda la versatilidad de sus gráficos.

Si bien hay varios paquetes de gráficos, el más popular y versatil es **ggplot** creado por Headley Wickham (mismo autor que Tidyverse), por lo que la lógica atrás de estos paquetes es similar, aunque la de ggplot es previa.

En el libro *GGplot2: Elegant Graphics for Data analysis* (se describe la lógica atrás de la gramática de los gráficos. Si bien no es super intuitiva, una vez que se la puede manejar, es sencillo hacerla crecer para lograr el objetivo buscado.

Ggplot tiene una gramática propia. Lo más importante en las visualizaciones es poder pensar cómo graficar del mejor modo posible los datos, conocer qué tipo de variables tenemos, si son continuas o discretas, etc, y qué poner en cada uno.

Ggplot funciona con una serie de *capas* que se van superponiendo.

- Primero, tenemos la **data** que queremos visualizar y un conjunto de **estéticas** de mapeo que describen cómo los datos serán mapeados en atributos estéticos.
- Los objetos geométricos, o **geom** sobre cómo resumir la información: el total de observaciones para un histograma, barras, scatterplot, etc...
- Las **escalas** que van a decir qué significan tanto las posiciones en el espacio, como los colores, las formas y los tamaños. Las escalas establecen leyendas o ejes.
- Un sistema de coordenadas, **coord** que describe cómo la data es mapeada en el plano del gráfico. Usualmente usamos coordenadas cartesianas (ejes x e y) pero hay otras disponibles.
- Una especificación por **facet**as que describe cómo se filtran los datos en subconjuntos y cómo visualizar cada uno de esos subconjuntos.

En ggplot vienen algunas geometrias prefijadas, que pueden unirse en el grafico que tengamos en la cabeza. Las **geom** puras, serían

1. Scatterplot. `geom_point()`
2. Bubblechart. `geom_point(size = variable)`
3. Barchart. `geom_bar` o `geom_col`
4. Box and whisker. `geom_boxplot()`
5. Lineas. `geom_line` o `geom_path`

En la **estetica** o **aes()** debemos poner cómo será la estetica del gráfico: qué variable va en el eje **x**, qué variable va en el eje **y**, qué variable va en el **color**, en el relleno **fill**, en el tamaño **size**, o en la forma **shape**.

Con esto podemos empezar a probar los gráficos.

Como primer ejemplo, vamos a usar los datos de **Iris** que es uno de los ejemplos precargados de Tydiverse. La otra más común es **mpg** o **caret**. Iris son los datos de plantas suculentas.

En primer lugar cargamos la librería tidyverse

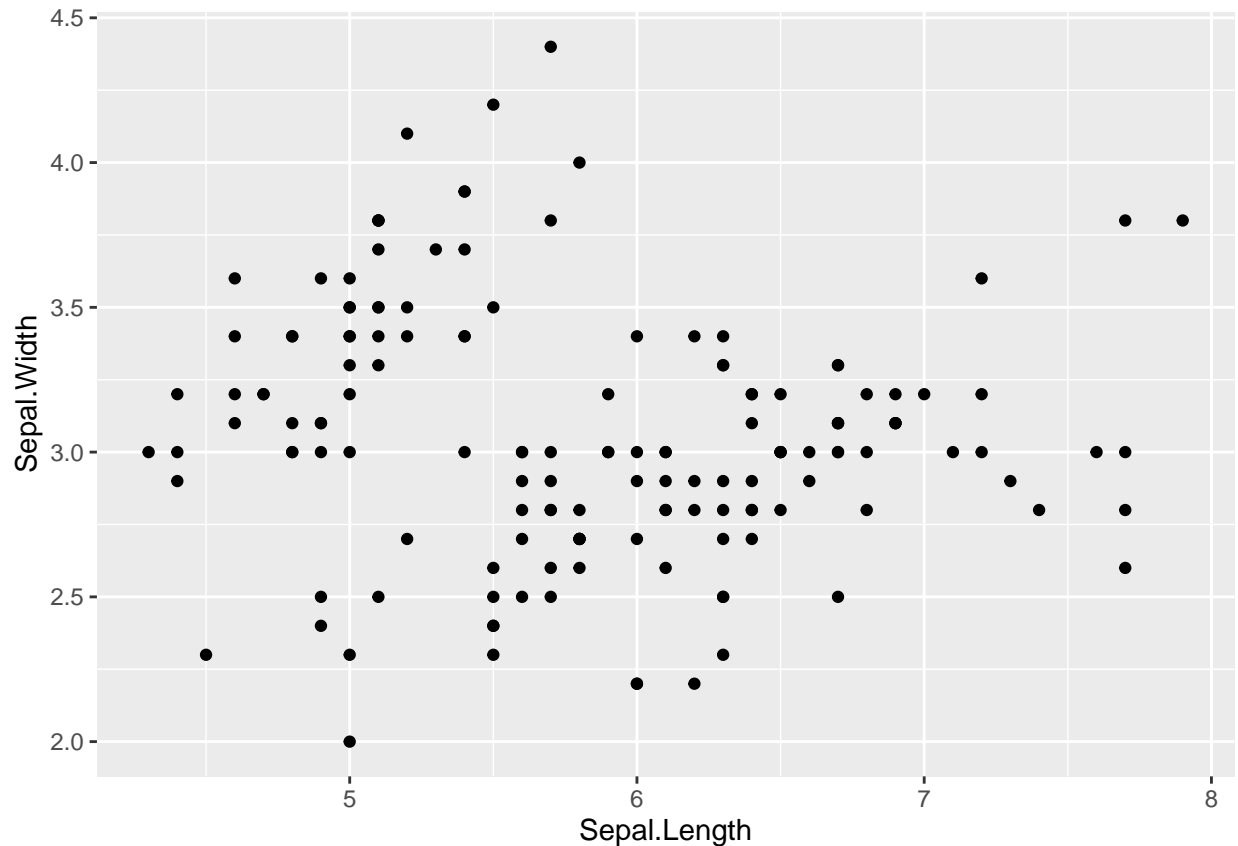
```
library(tidyverse)
library(sf)
library(RColorBrewer)
knitr::opts_chunk$set(fig.align = "center") #fijar opciones globales de chunks
```

Y exploramos la base de datos iris

```
str(iris)

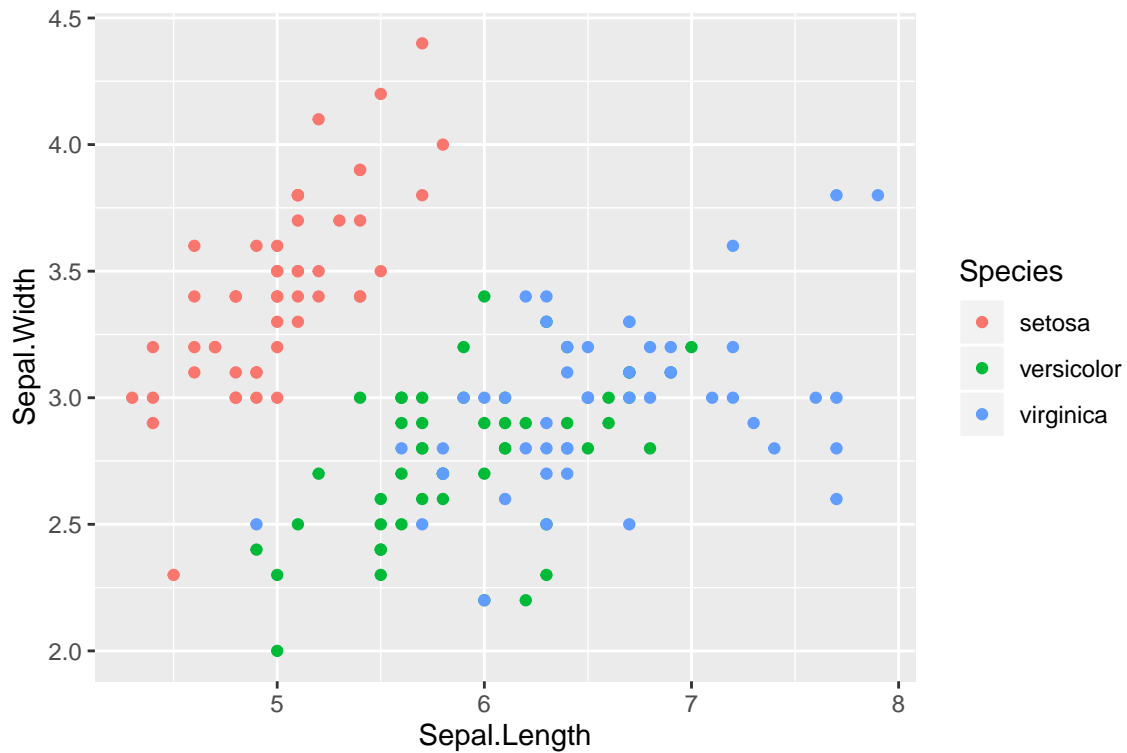
## 'data.frame': 150 obs. of 5 variables:
## $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
## $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
## $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
## $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
## $ Species : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...

#con un gráfico
ggplot()+
  geom_point(data = iris,
    aes(x = Sepal.Length, y = Sepal.Width))
```



Podemos agregarle colores en R. Si googleamos `colors in R` aparece la paleta y se pueden elegir lindos colores. El color tambien puede ser informativo, para ello hay que agregarlo adentro del `aes`.

```
ggplot()+
  geom_point(data = iris,
    aes(x = Sepal.Length, y = Sepal.Width, color = Species))
```



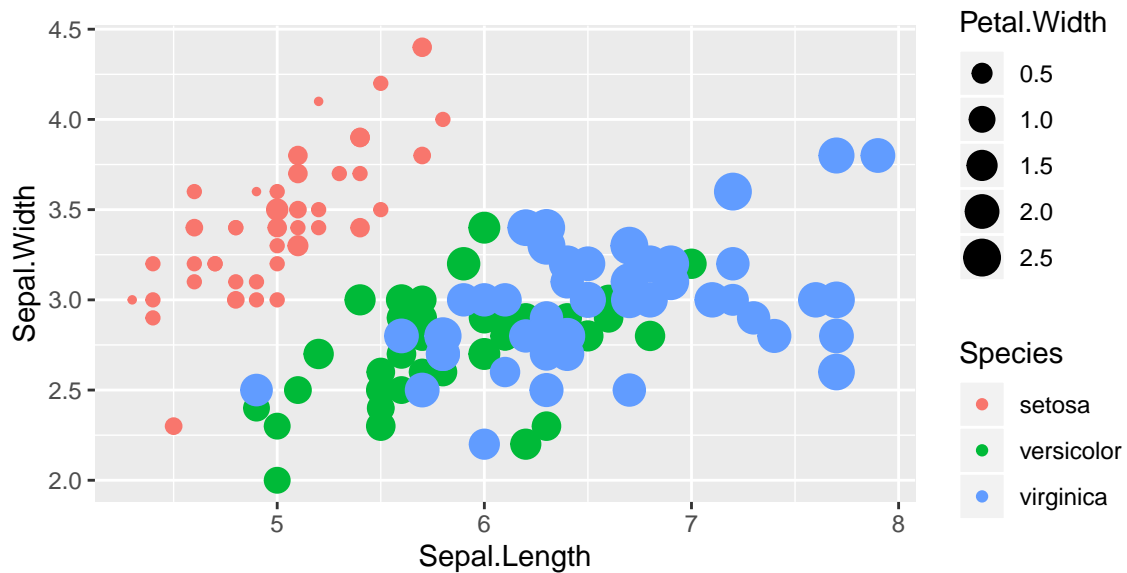
R tiene 12 colores predeterminados, si se requiere hacer 13 colores hay que darle otra paleta.

Dentro del `aes` el color va sin comillas porque es una variable, no un nombre de un color. La información puede ir en `ggplot()` como en `geom_point()`. Si va a `ggplot()` se aplica a todas las capas de abajo. Si va en `geom_point()` va solo en esa layer. Si tenemos bases distintas, es mejor ponerlo por `geom()`.

Se puede poner `shape` en vez de colores, y hay menos factores, 5 como máximo. Se puede poner color también.

Se puede usar también `size` que es para marcar otro tipo de información.

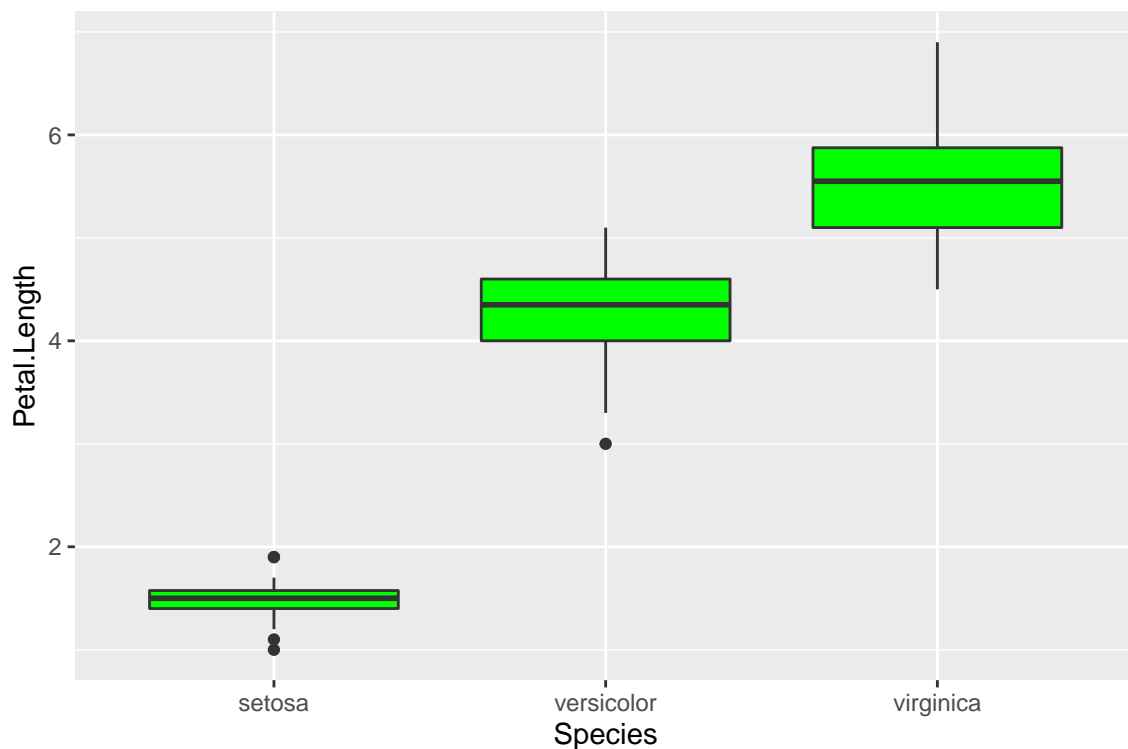
```
ggplot()+
  geom_point(data = iris,
             aes(x = Sepal.Length, y = Sepal.Width,
                 color = Species,
                 size = Petal.Width))
```



Un grafico tiene ventajas sobre las tablas. Permite absorber mucha más información de modo más rápido. Por eso tiene ventajas.

Vemos otro gráfico, si usamos uno con barras, y queremos colorear lo de adentro, la función no es `color` sino `fill` para el relleno

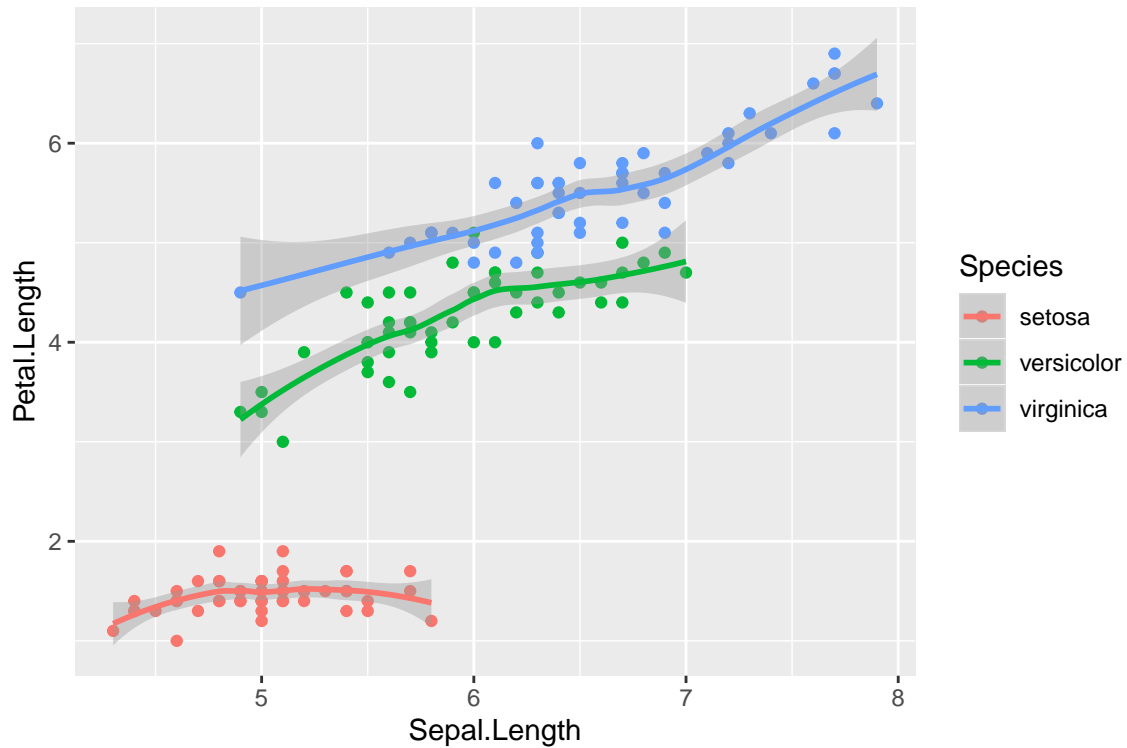
```
ggplot()+
  geom_boxplot(data = iris,
    aes(x = Species, y = Petal.Length),
    fill = "green")
```



`geom_violin()` es bueno para cosas que tienden a la media, es un boxplot pero con forma de violin

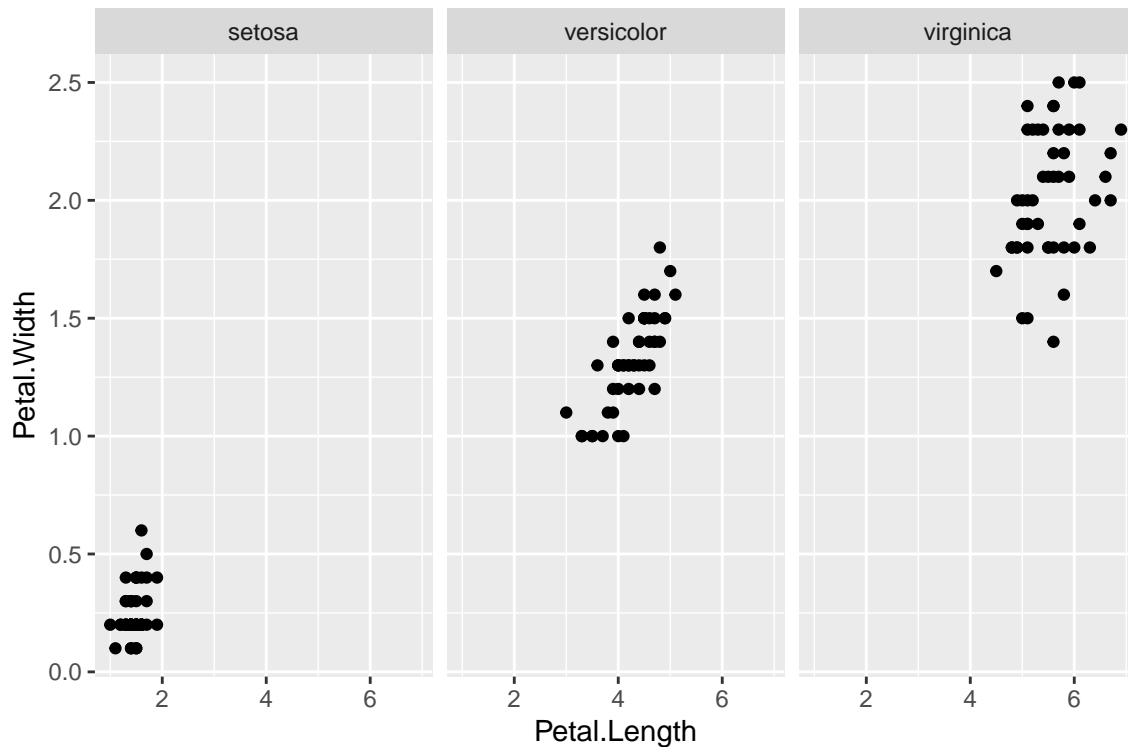
Se le pueden ir agregando capas con distintos atributos para el grafico, otros graficos, etc

```
ggplot(data = iris,  
       aes(x = Sepal.Length,  
           y = Petal.Length,  
           color = Species))+  
geom_point()+  
stat_smooth()
```



Y tambien por facets

```
ggplot(data = iris)+  
geom_point(aes(x = Petal.Length,  
              y = Petal.Width))+  
facet_grid(~Species) #se traduce "facetealo por Species"
```



Entonces, para hacer un gráfico hay que buscar la semántica de ggplot, que no es tan intuitiva. Primero habilitar la función, y luego ir agregando las distintas capas con información, la geometría, los textos, las escalas, y los temas...

En los últimos años, las visualizaciones cambiaron mucho. Ya no se piensa sólo una visualización en dos dimensiones para un paper blanco y negro, sino algo pensado para una web, en colores, o aún en movimiento, siempre que sea necesario para la visualización.

Por ejemplo, <https://www.gapminder.org/videos/hans-rosling-on-cnn-us-in-a-converging-world/>

O un homenaje a Rosling <https://twitter.com/demzayat/status/1121924353673846786...>

B. Geocomputación

¿Se acuerdan de los datos de la EAH de la clase pasada? Volvamos allí.

```
individuos <- read.csv2("/Users/demian/Documents/GitHub/derecho_y_datos/Clase2/eah2018_usuarios_ind.txt")
individuos_chica <- individuos %>%
  select(comuna, dominio, edad, sexo, estado, ingtot_2, ipcfb_2, nivel, aesc, m1_2, tipcob2_2, fexp)
```

Con esto conseguimos la base chica. Ahora la vamos a filtrar los ingresos por comuna, y unirlo a una base con datos geográficos o `geometry`.

```
xingresos_comuna <- individuos_chica %>%
  group_by(comuna) %>%
  summarize(prom = weighted.mean(ingtot_2, fexp))

#cargamos el geojson de las comunas
comunidades <- st_read("/Users/demian/Documents/GitHub/derecho_y_datos/Clase1/CABA_comunas.geojson")
```

```
## Reading layer 'OGRGeoJSON' from data source '/Users/demian/Documents/GitHub/derecho_y_datos/Clase1/CABA_comunas.geojson'
```

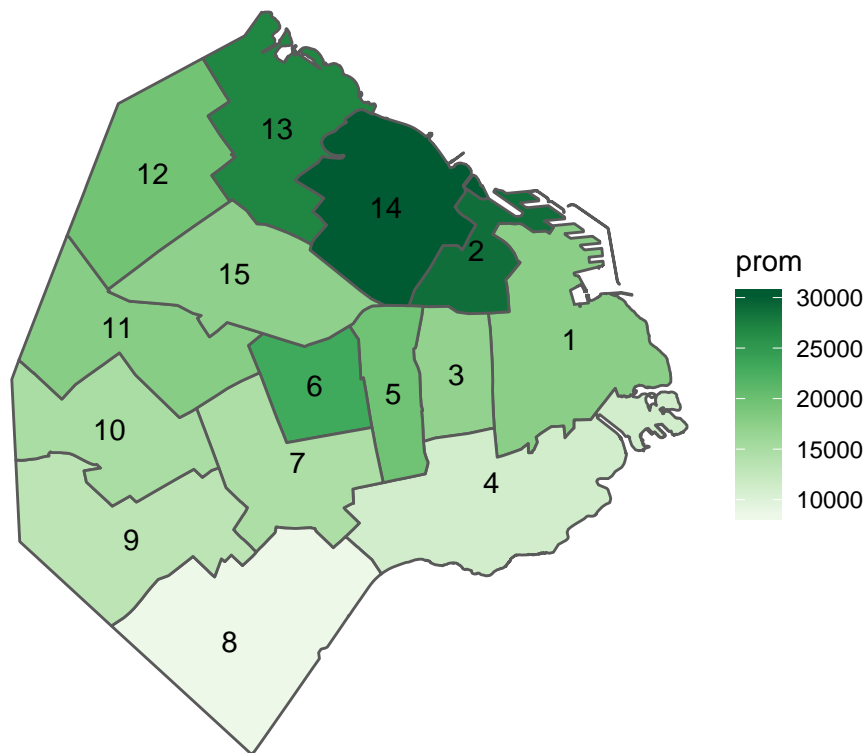
```
## Simple feature collection with 15 features and 4 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -58.53152 ymin: -34.70529 xmax: -58.33514 ymax: -34.52754
## epsg (SRID):    4326
## proj4string:     +proj=longlat +datum=WGS84 +no_defs

#convierto las comunas de la base de dato a factores
xingresos_comuna$comuna <- factor(xingresos_comuna$comuna)

#agrego la geometry al xingresos_comuna
xingresos_comuna_geo <- comunas %>%
  left_join(xingresos_comuna, by = c("comunas" = "comuna"))

#y grafico
ggplot()+
  geom_sf(data = xingresos_comuna_geo, aes(fill = prom))+
  geom_sf_text(data = xingresos_comuna_geo, aes(label = comunas))+
  scale_fill_distiller(direction = 1, palette = "Bugn")+
  labs(title = "Ingresos por Comuna")+
  theme_void()
```

Ingresos por Comuna



Datos del CENSO 2010 en CABA

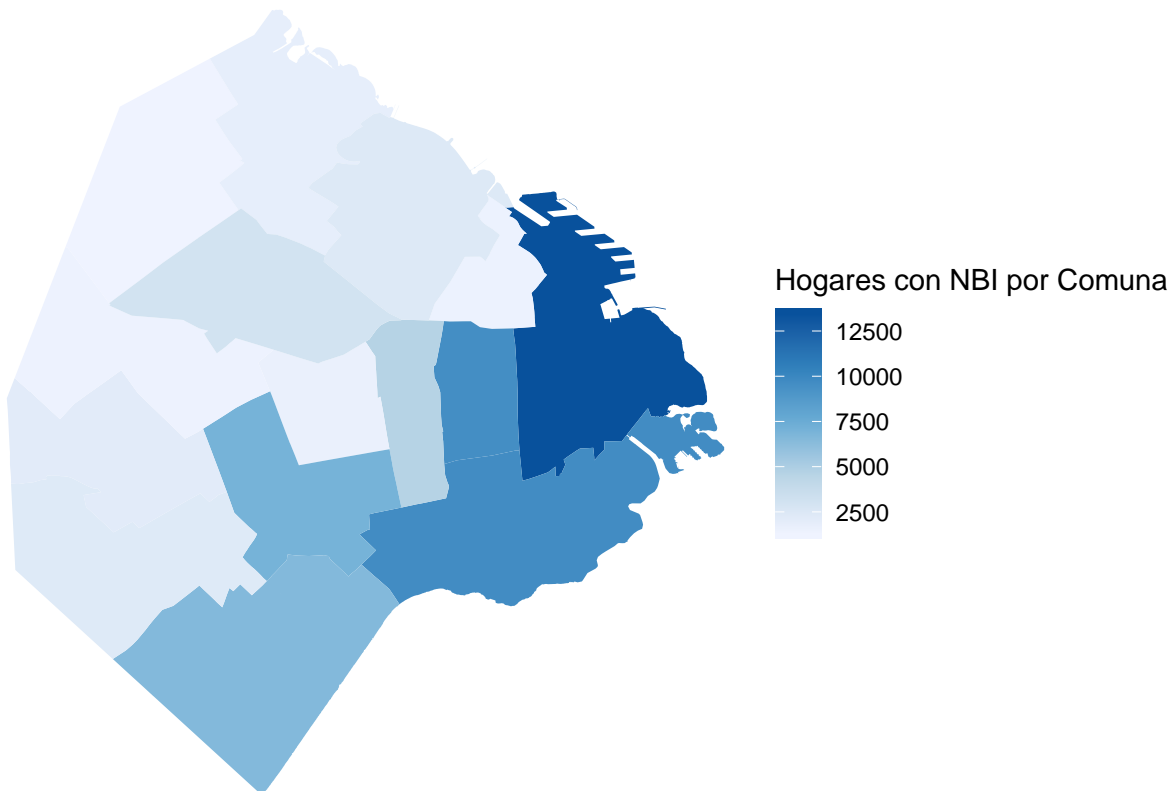
Vamos a ver datos del censo 2010 de CABA, disponibles en el sitio <https://data.buenosaires.gob.ar/dataset/informacion-censal-por-radio>. Habrá que descargar y descomprimir el archivo **shapefile** y cargarlo con **st_read**

```
censo <- st_read(dsn = "/Users/demian/Documents/GitHub/derecho_y_datos/Clase4", layer = "informacion_cen")

## Reading layer `informacion_censal_por_radio_2010' from data source `/Users/demian/Documents/GitHub/d
## Simple feature collection with 3554 features and 14 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 93743.42 ymin: 91566.42 xmax: 111752.2 ymax: 111284.4
## epsg (SRID):    NA
## proj4string:     +proj=tmerc +lat_0=-34.6297166 +lon_0=-58.4627 +k=0.9999980000000001 +x_0=100000 +y_0=6200000
```

Este shapefile es bastante completo, ya que además de la geolocalización y tiene la info del censo, por comuna, fracción y radio censal. Podemos filtrar la cantidad de hogares con NBI por comuna

```
xcomuna <- censo %>%
  group_by(COMUNA) %>%
  summarise(nbi = sum(H_CON_NBI))
ggplot()+
  geom_sf(data = xcomuna, aes(fill = nbi), col = NA)+
  scale_fill_gradientn(colours = brewer.pal(n = 5, name = "Blues"),
    name = "Hogares con NBI por Comuna")+
  theme_void()
```

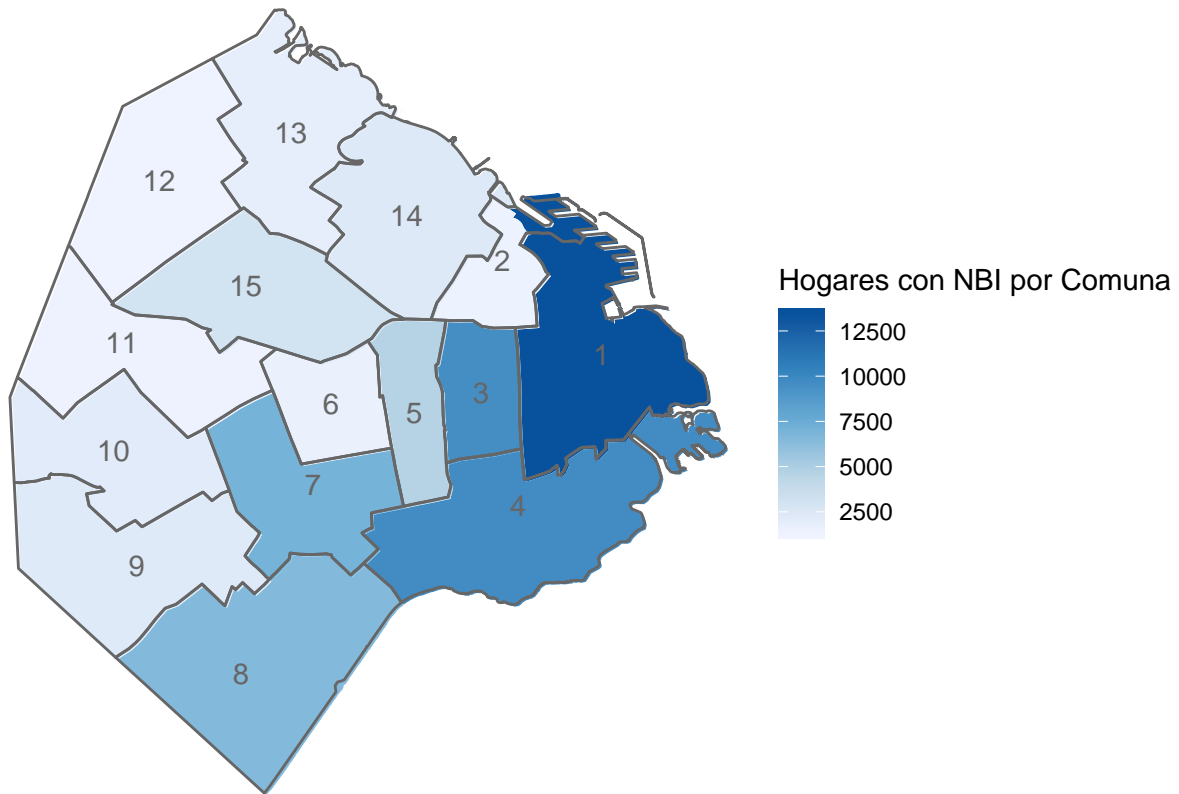


Y le agregamos la división por comunas del shapefile comunas (sin relleno), y agregamos también el número de la comuna

```
ggplot()+
  geom_sf(data = xcomuna, aes(fill = nbi), col = NA)+
  scale_fill_gradientn(colours = brewer.pal(n = 5, name = "Blues"),
    name = "Hogares con NBI por Comuna")+
  geom_sf(data = comunas, fill = NA, col = "grey40")+
```



```
geom_sf_text(data = comunas, aes(label = comunas), col = "grey40")+
theme_void()
```

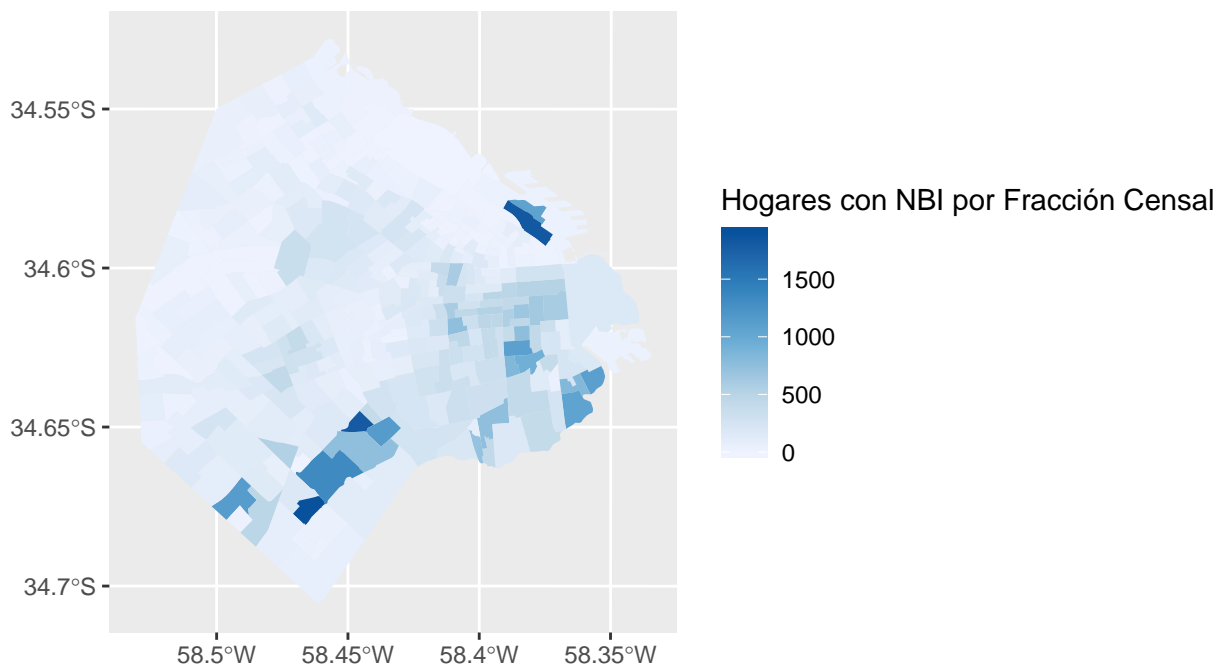


También podemos calcular la cantidad de viviendas por Fracción censal. Primero tenemos que crear la variable de comuna_fracción, para tener todas las fracciones con un nombre distinto, y luego agrupar por esta variable y calcular la cantidad de Hogares con NBI

```
censo_fra_co <- censo %>%
  unite(CO_FRA, COMUNA, FRACCION, sep = "_", remove = FALSE)

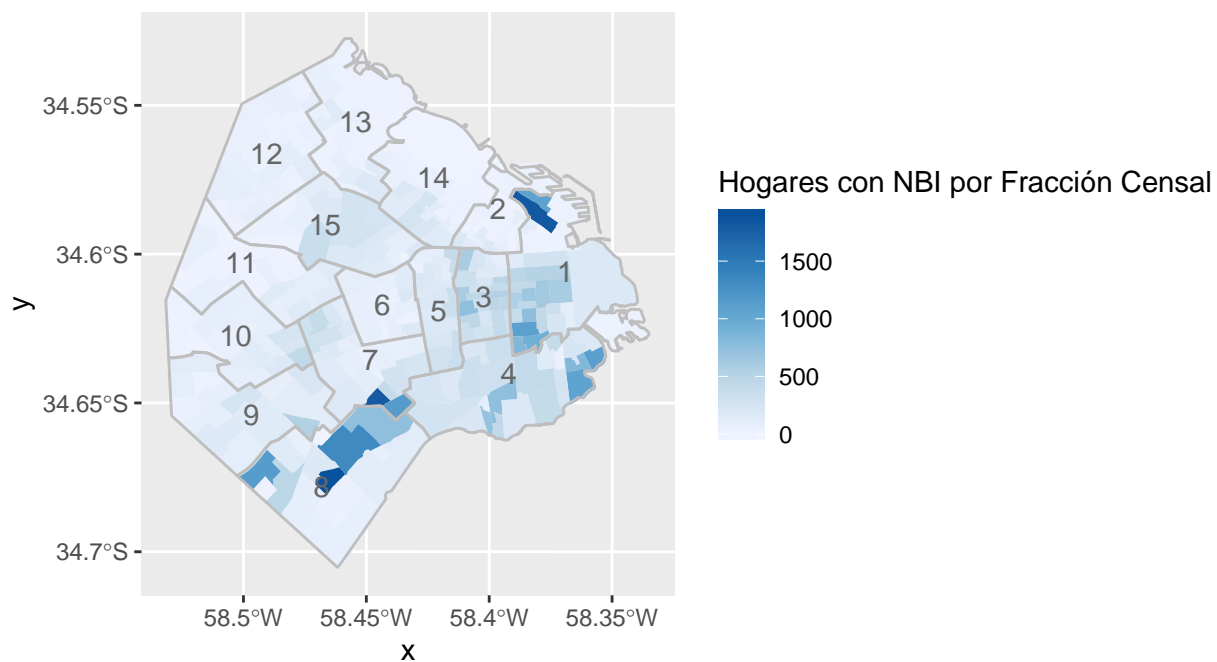
xfraccion <- censo_fra_co %>%
  group_by(CO_FRA) %>%
  summarise(nbi = sum(H_CON_NBI))

ggplot()+
  geom_sf(data = xfraccion, aes(fill = nbi), col = NA)+
  scale_fill_gradientn(colours = brewer.pal(n = 5, name = "Blues"),
    name = "Hogares con NBI por Fracción Censal")
```



Y le agregamos la división por comuna

```
ggplot()+
  geom_sf(data = xfraccion, aes(fill = nbi), col = NA)+
  scale_fill_gradientn(colours = brewer.pal(n = 5, name = "Blues"),
    name = "Hogares con NBI por Fracción Censal")+
  geom_sf(data = comunas, fill = NA, col = "grey")+
  geom_sf_text(data = comunas, aes(label = comunas), col = "grey40")
```



Y aún lo podemos hacer por radio censal. La variable CO_FRA_RA ya venía en la data del censo

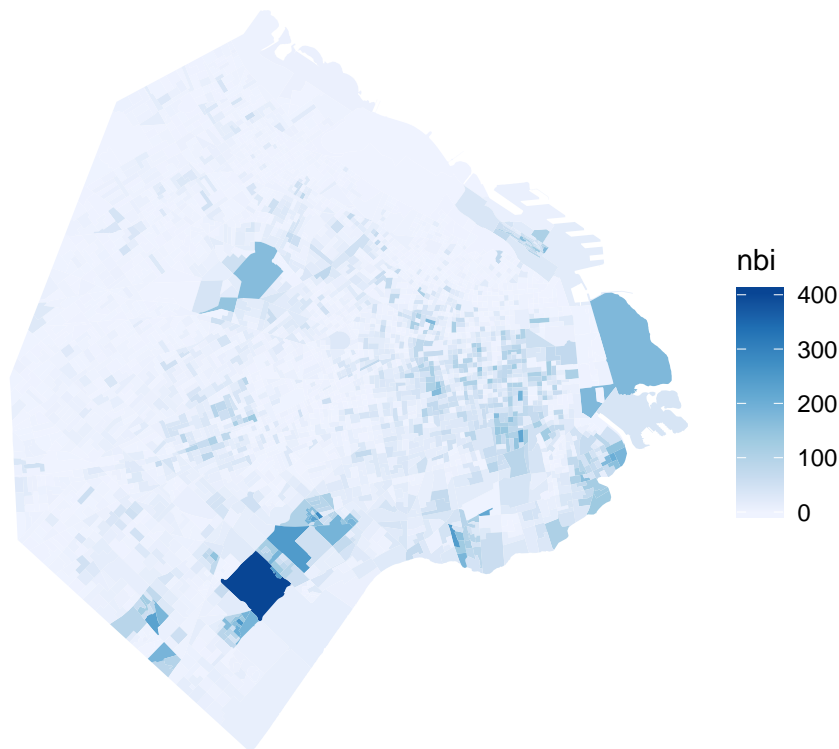
```
xradio <- censo %>%
  group_by(CO_FRAC_RA) %>%
```

```

summarise(nbi = sum(H_CON_NBI))
ggplot()+
  geom_sf(data = xradio, aes(fill = nbi), col = NA)+
  scale_fill_distiller(direction = 1)+
  labs(title = "Hogares con NBI")+
  theme_void()

```

Hogares con NBI



Y podemos agregarle la división por Barrios, pero para eso necesitamos un shapefile con los barrios. Y luego agregarlo como una capa al mapa

```

barrios <- st_read("/Users/demian/Documents/GitHub/derecho_y_datos/Clase4/barrios.geojson")

```

```

## Reading layer `OGRGeoJSON' from data source `/Users/demian/Documents/GitHub/derecho_y_datos/Clase4/b
## Simple feature collection with 48 features and 4 fields
## geometry type: POLYGON
## dimension: XY
## bbox: xmin: -58.53092 ymin: -34.70574 xmax: -58.33455 ymax: -34.52799
## epsg (SRID): 4326
## proj4string: +proj=longlat +datum=WGS84 +no_defs

```

```

ggplot()+
  geom_sf(data = xradio, aes(fill = nbi), col = NA)+
  geom_sf(data = barrios, fill = NA, col = "grey60")+
  geom_sf_text(data= barrios, aes(label = BARRIO), size = 1.7, col="grey40")+
  scale_fill_distiller(direction = 1)+
  labs(title = "Hogares con NBI en la CABA por Barrios", fill = "Hogares")+
  theme_void()

```

Hogares con NBI en la CABA por Barrios

