

Spatial Machine Learning

Maximiliano S. Lioi

2023-01-29

Documentación

- Analyzing US Census Data - Kyle Walker
- Spatial Machine Learning - J. Morgan
- Import and Map NYC Census data into R with tidycensus

En *Analyzing US Census Data - Kyle Walker* se muestra como hacer uso de las librerías, entre ellas:

- tidycensus : Paquete de R diseñado para facilitar procesos de adquirir y trabajar data de US Census, busca distribuir los datos del censo en un formato compatible con tidyverse, además busca agilizar el proceso del tratado de datos para aquel que esté trabajando en el análisis de datos. Ch2.
- tidyverse : Colección de paquetes de R diseñados para la ciencia de datos tales como *ggplot2* para la visualización de data, *readr* para importar y exportar bases de datos, *tidyr* para la remodelación de datos, entre otros. Ch3.
- tigris : Paquete de R que busca simplificar procesos para los usuarios de obtención de información y uso de data con atributos geográficos (data espacial, Census geographic dataset), data tipo *sf* (simple features) viene con atributos de geometría (vector data type, típicamente representados por puntos líneas o polígonos). Ch5.
- ggplot2 : Paquete de R enfocado en la visualización de data, nos permite realizar mapas con información de US Census data. Ch6.

Spatial Data

Viene representada en formas como:

- puntos (point reference data), i.e, ciudades en el mapa
- líneas (line string), i.e, caminos en el mapa
- polígonos (shapes) , i.e, distritos censales

Librerías

```
library(tidycensus)
library(tidyr)
library(tidyverse)
library(censusapi)
library(tmap)
library(ggplot2)
library(dplyr)
library(stringr)
library(units)
library(stats)
library(grDevices)
library(dotenv)
library(sf)
library(corr)
library(spatialreg)
library(spdep)
library(jtools)
library(huxtable)
library(GWmodel)
library(spgwr)
library(SpatialML)
library(ggthemes)
```

Activacion API key

El siguiente código ejecuta la activación de la llave ‘API Key’ que nos permite descargar Census Data, mediante funciones como `get_acs`, el primer argumento es la llave utilizada en este código.

```
census_api_key("6034739b488f5fc230e467601ed20256bb25831b", install = TRUE)
```

Este comando tiene la estructura

```
census_api_key(key, overwrite = BOOL, install = BOOL)}
```

Argumentos

- `key`: La API Key entregada por el Censo, ingresar con “. Se obtiene en API Census.
- `overwrite`: Si está en `TRUE`, sobrescribirá sobre una ya existente `CENSUS_API_KEY` que tengamos instalado en nuestro archivo `.Renviron`
- `install`: Si está en `TRUE`, instalará la llave en nuestro archivo `.Renviron` para las futuras sesiones, de no existir crea uno. Viene en `FALSE` por defecto.

Despues de instalada la llave, puede usarse en cualquier momento llamando el siguiente comando

```
Sys.getenv("CENSUS_API_KEY")
```

```
## [1] "6034739b488f5fc230e467601ed20256bb25831b"
```

Reload del enviroment para poder usar la llave sin tener que resetear R

```
readRenviron("~/Renviron")
```

NY US Census data

Tomamos como caso de prueba al estado de New York, para visualizar el valor medio de las viviendas a nivel de condados, podemos variar el nivel geográfico con el parámetro *geography* (Walker K., 2023) [7]

```
# Creamos el objeto median_nyc que contiene la variable "median income"
# con nivel geografico condados(county).
medianincomenystate <- get_acs(geography = "county",
  state = "New York",
  geometry = TRUE, #descarga el componente espacial del tramo censal
  variable = "B19013_001" #median income
)
```

Tidycensus, funciones principales para obtener data

Para obtener datos de las distintas bases *tidycensus* ofrece las siguientes funciones

- `get_decennial()` : Solicita datos de las API US Decennial Census para 2000, 2010 y 2020.
- `get_acs()` : Solicita datos de las muestras de la American Community Survey de 1 y 5 años. Los datos están disponibles desde el ACS de 1 año hasta 2005 y el ACS de 5 años hasta 2005-2009.
- `get_estimates()` : Interfaz para las Population Estimates APIs. Estos conjuntos de datos incluyen estimaciones anuales de las características de la población por estado, condado y área metropolitana, junto con componentes de estimaciones demográficas de cambio como nacimientos, muertes y tasas de migración.
- `get_pums()` : Accede a los datos de ACS Public Use Microdata Sample APIs, Estas muestras incluyen registros anónimos a nivel individual de la ACS organizados por hogar y son muy útiles para muchos análisis de ciencias sociales, `get_pums()` se cubre con más profundidad en los Capítulos 9 y 10 de *Analyzing US Census data*.
- `get_flows()` : Interfaz para la ACS Migration Flows APIs. Incluye información sobre los flujos de entrada y salida de varias geografías para las muestras de ACS de 5 años, lo que permite realizar análisis de origen y destino

De manera más general, para ver que variables se pueden obtener, *tidycensus* nos provee de la función `load_variables()`, dicha función requiere de 2 argumentos, *year* que toma el año de referencia de la data, y *dataset*.

Para el Decennial Census 2000 a 2010, usar “sf1” o “sf2”, el 2020 Decennial Census tambien acepta “sf3” y “sf4”, sf hace referencia a Summary Files.

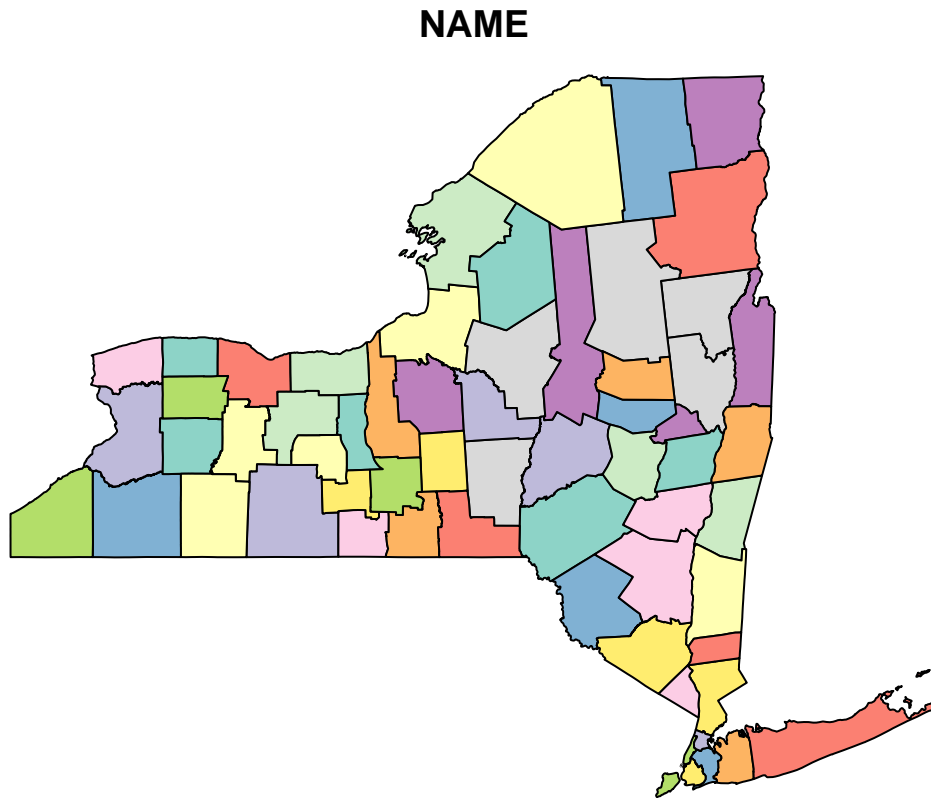
Para variables de la American Community Survey, debemos especificar el año de la encuesta, por ejemplo “acs1” para el primer año de la ACS, por ejemplo si se quiere acceder a la data *5-year ACS*

```
load_acs = load_variables(year = 2020, dataset = "acs5")
```

Plot del estado New York

Comencemos a visualizar la información, probaremos primeramente con plot, y luego usaremos herramientas mas avanzadas que nos ofrecen los paquetes

```
plot(medianincomenystate["NAME"])
```

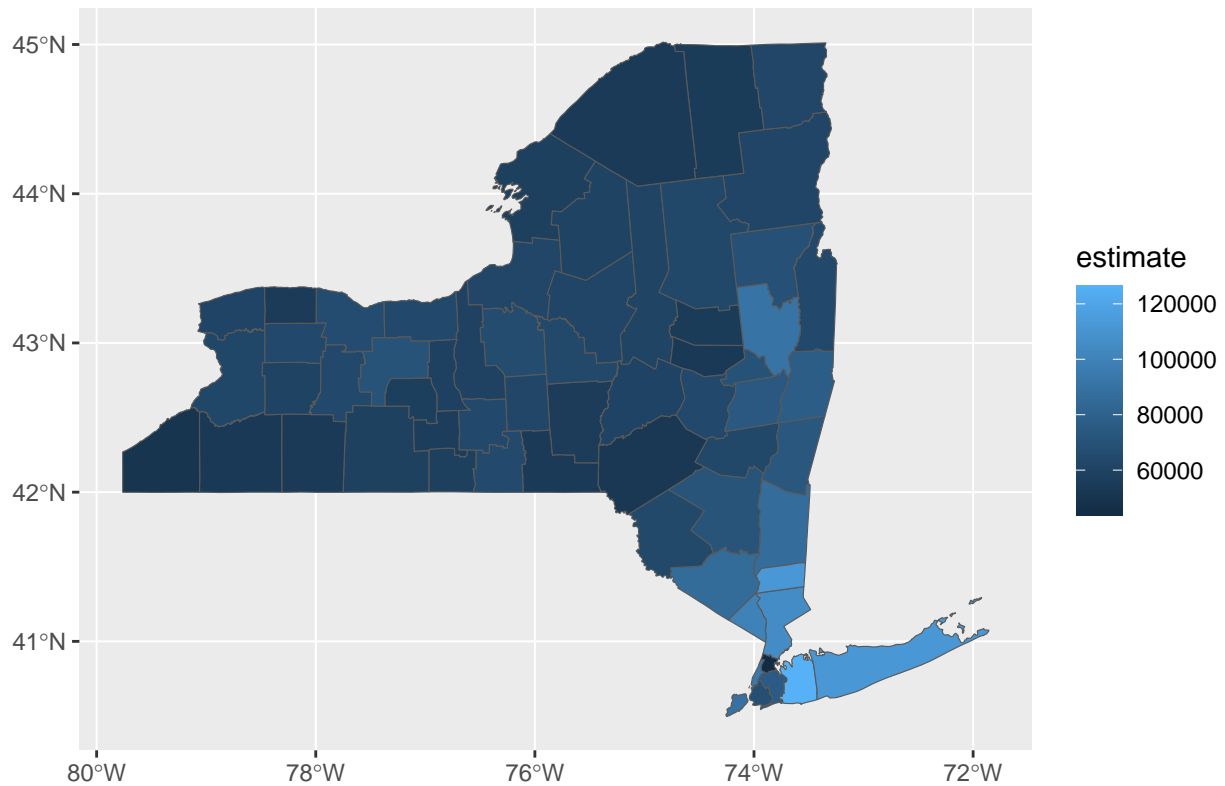


Plot del valor medio de las viviendas en New York

Map-making con ggplot2 y geom_sf

En *ggplot2* podemos plotear rapidamente objetos de tipo *sf* mediante `geom_sf()`, para entender la sintaxis realizamos el siguiente plot de el estimado de la variable “Median income New York”.

```
ggplot(data = medianincomenystate, aes(fill = estimate)) +  
  geom_sf()
```

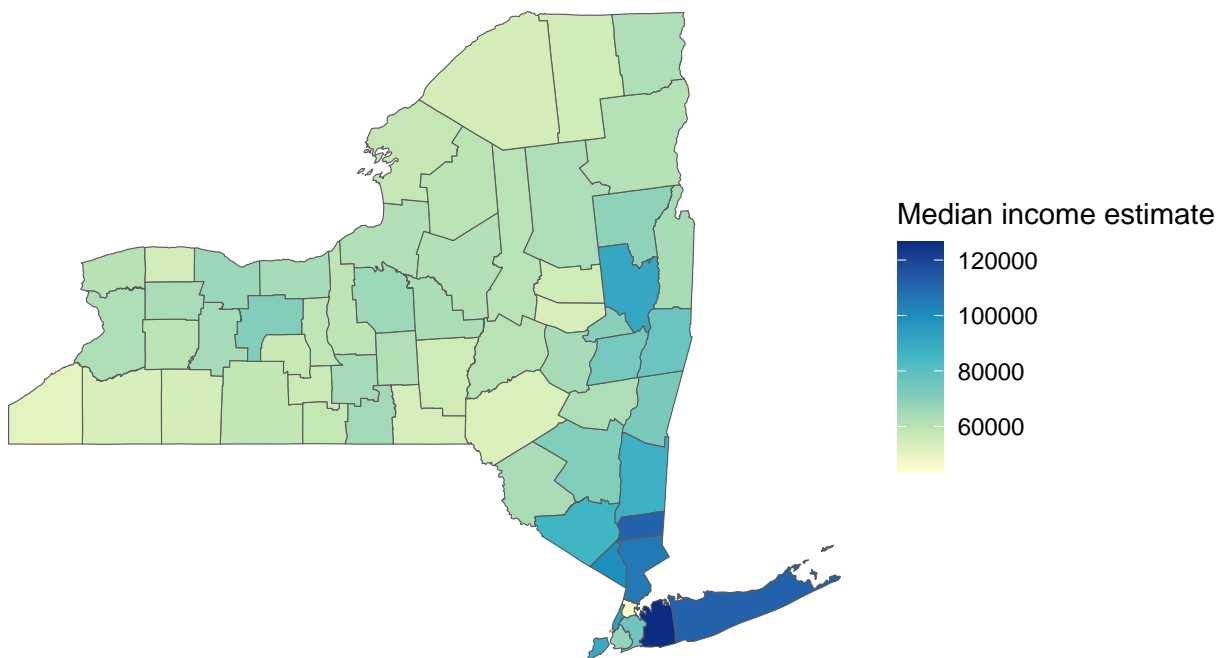


Customizing ggplot2 maps

Podemos customizar nuestros plots en ggplot2, la estructura es la siguiente

```
ggplot(data = medianincomenystate, aes(fill = estimate)) +  
  geom_sf() +  
  scale_fill_distiller(palette = "YlGnBu",  
                      direction = 1) +  
  labs(title = "Median income New York, 2016-2020",  
       caption = "Data source: 2016-2020, US Census",  
       fill = "Median income estimate") +  
  theme_void()
```

Median income New York, 2016–2020



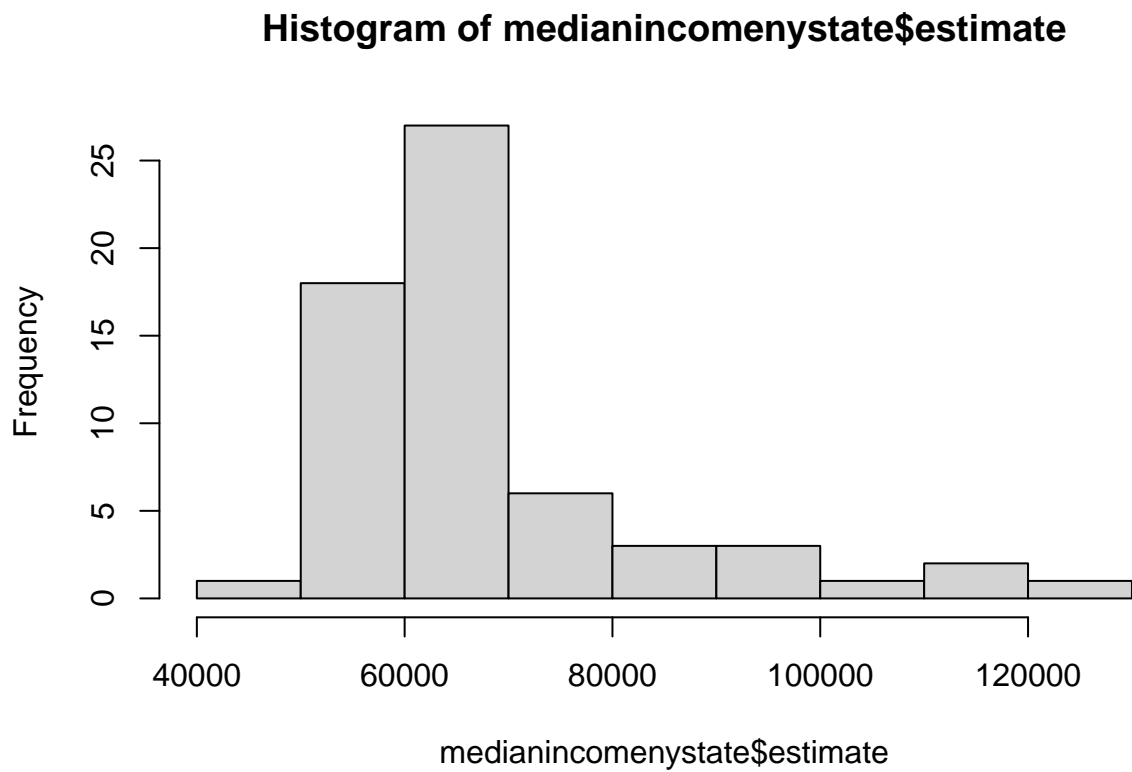
Data source: 2016–2020, US Census

Las funciones que acompañan la sintaxis nos permiten customizar nuestro plot en ggplot2.

- `scale_fill_distiller()` : Nos permite especificar una paleta de colores de ColorBrewer en el plot.
- `labs()` : Nos permite añadir título, caption, legend label en el plot.
- `theme_void()` : Nos permite remover el fondo y la grilla cuadrícula.

Histograma del valor medio de las viviendas en el estado New York

```
hist(medianincomenystate$estimate)
```



Map-making con tmap

La sintaxis es similar a la usada en *ggplot2*, el objeto mapa se inicializa con la función `tm_shape()` y nos permite visualizar los distritos censales con `tm_polygons()`

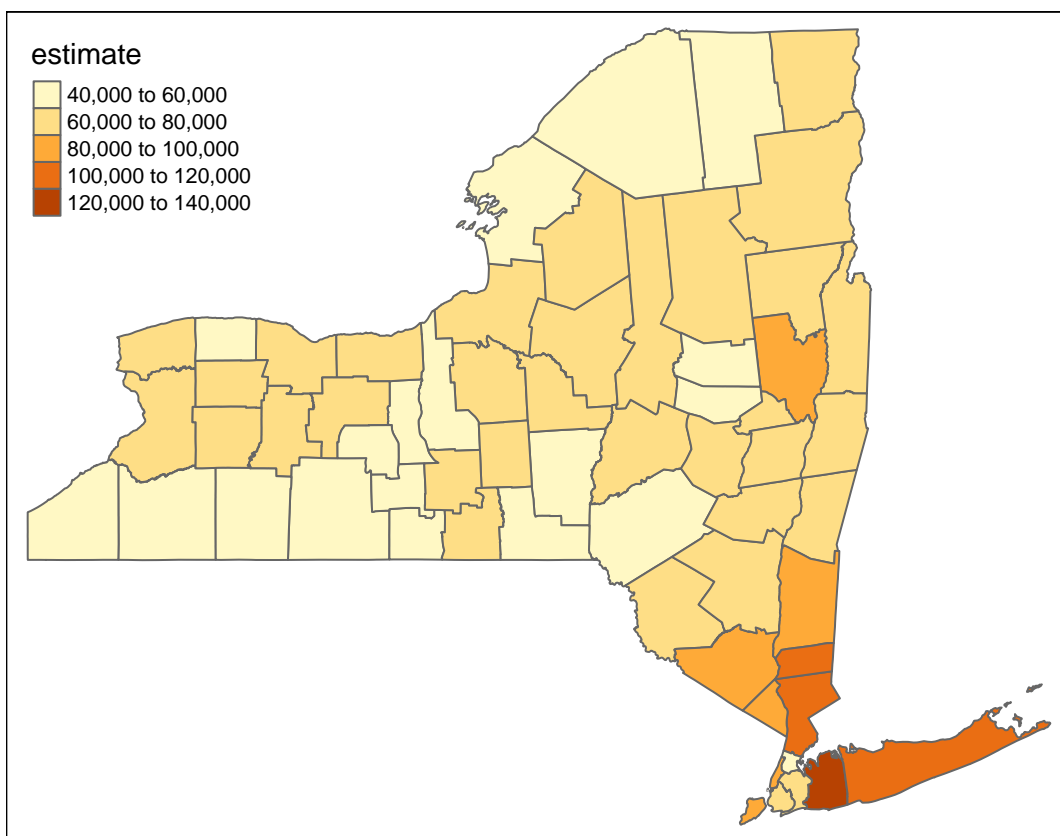
```
library(tmap)
tm_shape(medianincomenystate) +
  tm_polygons()
```



Variables en tmap

Veamos nuevamente la variable “median income”

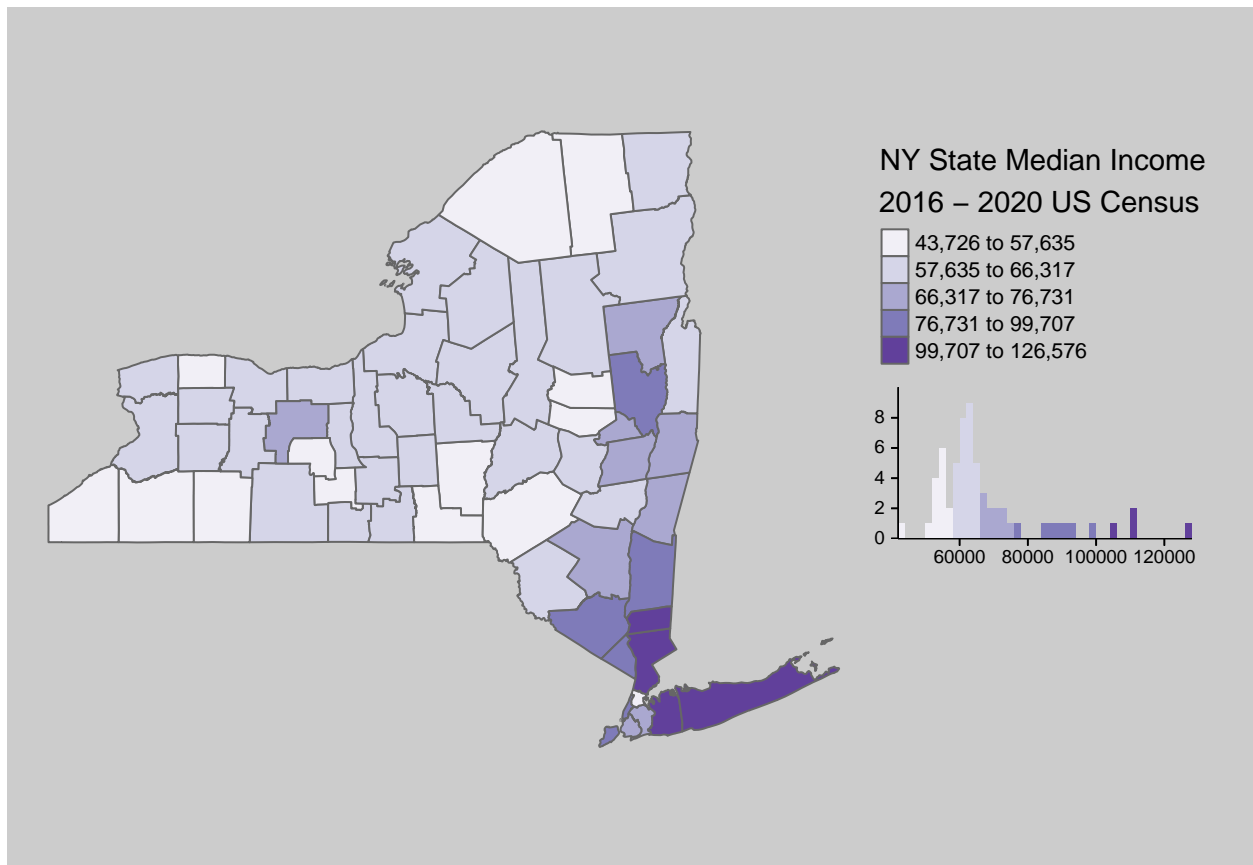
```
library(tmap)
tm_shape(medianincomenystate) +
  tm_polygons(col = "estimate")
```



Labels y otras opciones de diseño

Podemos añadir más variables y seguir personalizando nuestros plots, por ejemplo añadiendo histogramas por distintos tipos de clasificación, quantiles (“quantile”), equal intervals (“equal”) y Jenks natural breaks (“jenks”), con `tm_layout()` nos permite customizar el estilo del mapa, del histograma y añadir leyendas.

```
tm_shape(medianincomenystate) +  
  tm_polygons(col = "estimate",  
              style = "jenks",  
              n = 5, #num de intervalos  
              palette = "Purples",  
              title = "2016 - 2020 US Census",  
              legend.hist = TRUE) +  
  tm_layout(title = "NY State Median Income",  
            frame = FALSE,  
            legend.outside = TRUE,  
            bg.color = "grey80", #backgroundcolor  
            legend.hist.width = 7)
```



NYC Median Income

Podemos trabajar con niveles geográficos de menor nivel, como condados y tracts, trabajamos la ciudad de New York, formado por ciertos condados.

```
# Creamos el objeto medianincomenyc que contiene la variable "median income"  
# con nivel geografico de tract.  
medianincomenyc <- get_acs(geography = "tract",  
  state = "New York",  
  geometry = TRUE, #descarga el componente espacial del tramo censal  
  variable = "B19013_001" #median income  
)
```

Filtramos respecto de los condados que queremos visualizar, aquellos cerca de la ciudad de NY, usamos *tidyr* con la función `separate()`, de manera de poder filtrar los condados que nos interesan

El siguiente código nos permite crear nuevas columnas “tract” y “county”, de manera que podemos filtra aquellos condados de interes con otras funciones, ademas usamos `na.omit()` para botar aquellos valores con NA y así limpiar la data

```
medianincomenyc <- separate(medianincomenyc,  
  NAME,  
  into = c("tract", "county"),  
  sep = ", ")  
  
medianincomenyc <- medianincomenyc %>% filter(grepl('Bronx County|New York County|Queens County',  
  NAME))  
  
medianincomenyc <- na.omit(medianincomenyc)
```

Plot con tmap de NYC sobre los ingresos promedios

```
tm_shape(medianincomenyc) +  
  tm_polygons(col = "estimate",  
              style = "equal",  
              palette = "Purples",  
              title = "2016 - 2020 US Census",  
              legend.hist = TRUE) +  
tm_layout(title = "NYC Median Income by Census Tract",  
          frame = FALSE,  
          legend.outside = TRUE,  
          bg.color = "grey80", #backgroundcolor  
          legend.hist.width = 7)
```



Spatial Machine Learning

Replicamos la simulación Spatial Machine Learning, Justin Morgan Williams, con el fin de entender los desafíos con los que se encuentra el Machine Learning con los datos espaciales, dicha simulación esta fuertemente basada en el capítulo 8 de *Analyzing US Census Data, Modeling US Census Data*[7], respecto de los desafíos al tratar datos espaciales, citando.

- *Autocorrelación espacial* → autocorrelación debida a la similitud en la ubicación del componente espacial de los datos
- *Heterogeneidad espacial* → datos que no siguen una distribución idéntica dentro del área de muestra
- *Limited Ground Truth* → muchas variables explicativas, verdad de terreno limitada
- *Multiple Scales and Resolutions* → puede existir en múltiples escalas y resoluciones

Si no se toman en cuenta, pueden tener efecto en la predicción de Machine Learning, entregando resultados que no son óptimos, la simulación trata los primeros 2 puntos, autocorrelación espacial y heterogeneidad espacial.

Importando data

Importamos data de NYC US Census con *tidycensus*

```
#load package
library(dotenv)
library(sf)
library(tidycensus)
library(dplyr)

# set county variables, condados que forman NYC
nyc_counties <- c("Bronx","Kings","New York","Queens","Richmond")

# set list of census variables

# variable list
variables <- c(
  median_value = "B25077_001",
  median_rooms = "B25018_001",
  median_income = "DP03_0062",
  total_population = "B01003_001",
  median_age = "B01002_001",
  pct_college = "DP02_0068P",
  pct_foreign_born = "DP02_0094P",
  pct_white = "DP05_0077P",
  pct_black = "DP05_0078P",
  pct_hispanic = "DP05_0070P",
```

```
pct_asian = "DP05_0080P",  
median_year_built = "B25037_001",  
percent_ooh = "DP04_0046P"  
)
```

Import usando get_acs()

Llamamos las variables antes definidas, en los condados que forman NYC usando get_acs()

```
# get acs data y transforma a tipo NYC EPSG
nyc_census_data <- get_acs(
  geography = "tract",
  variables = variables,
  state = "NY",
  county = nyc_counties,
  geometry = TRUE,
  output = "wide",
  year = 2020,
  key = Sys.getenv("CENSUS_API")) %>%
  st_transform(2263)
```

Las variables que importamos, que se corresponden a estimados que entrega la ACS sobre los condados de NY, son las siguientes:

- median_valueE : El valor medio de la vivienda del tramo censal (nuestro outcome)
- median_roomsE : Cantidad media de habitaciones por casa en el tramo censal
- total_populationE : Población total
- median_ageE : Edad media de la población en el tramo censal
- median_year_builtE : Año promedio donde se construyó la vivienda
- median_incomeE : Ingreso medio de los hogares en el tramo censal
- pct_collegeE : Porcentaje de la población de 25 años o más con un título universitario de cuatro años
- pct_foreign_bornE: Porcentaje de la población que nació fuera de EE.UU
- pct_whiteE : Porcentaje de la población que se identifica como blanco no-hispano, se sigue la misma lógica con pct_blackE, pct_asian, pct_hispanic.
- percent_oohe : El porcentaje de unidades de vivienda en el tramo censal que están ocupadas por sus propietarios.

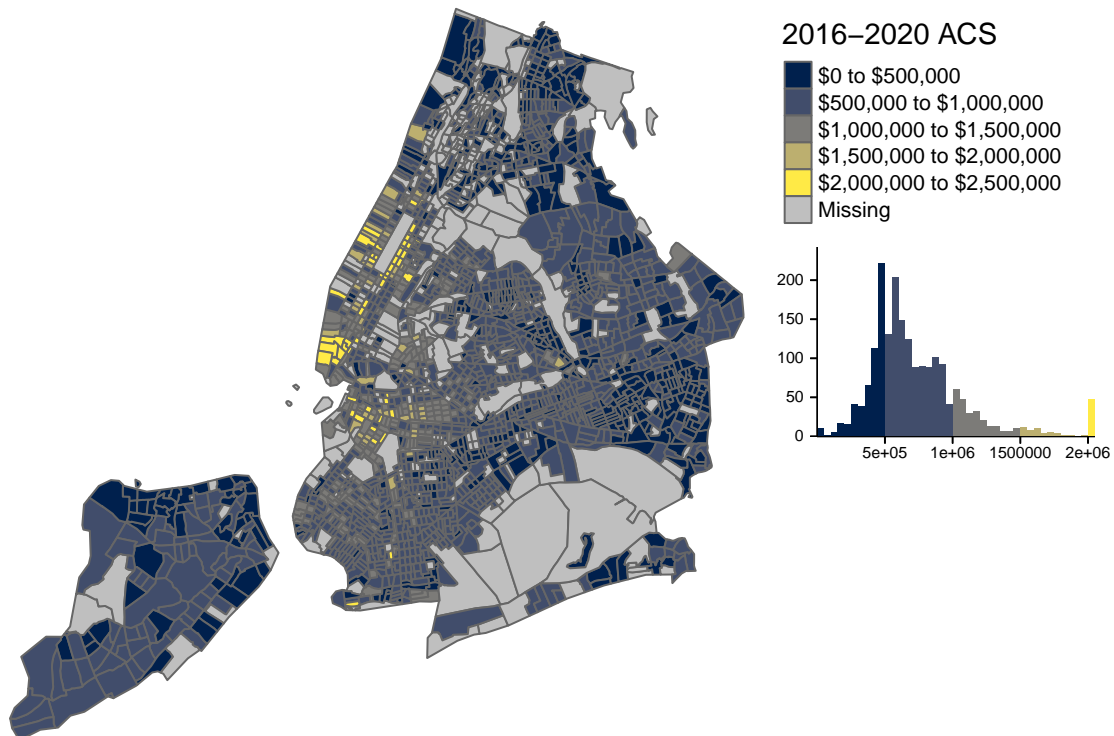
Más detalles sobre la estructura del código se encuentran en la sección 8.2.1 (Data setup and exploratory data analysis), en el libro se toma la misma variable “median home value”, pero que a diferencia de esta réplica, se toman los tramos censales en Dallas-Fort Worth metropolitan area, en consecuencia también cambian el código usando en st_transform, a un sistema de referencias apropiado para North Texas (code 32148), a diferencia del caso NYC donde usamos code 2263

Plot de la variable dependiente

Nuestra variable dependiente será el valor promedio de la vivienda en NYC.

```
# plot tmap
(nyc_median_value_hist_tm <- nyc_census_data[!st_is_empty(nyc_census_data),,drop=F] %>%
tm_shape() +
  tm_polygons(col = "median_valueE",
    palette = "cividis",
    title = "2016-2020 ACS",
    legend.hist = TRUE,
    legend.format = scales::dollar_format()) +
tm_layout(main.title = "NYC Median Home Value by Census Tract",
  frame = FALSE,
  legend.outside = TRUE,
  bg.color = "grey100",
  legend.hist.width = 5,
  ))
```

NYC Median Home Value by Census Tract



```
# save plot
tmap_save(nyc_median_value_hist_tm, "image/nyc_median_value.png", width=1920, height=1080, asp=
```

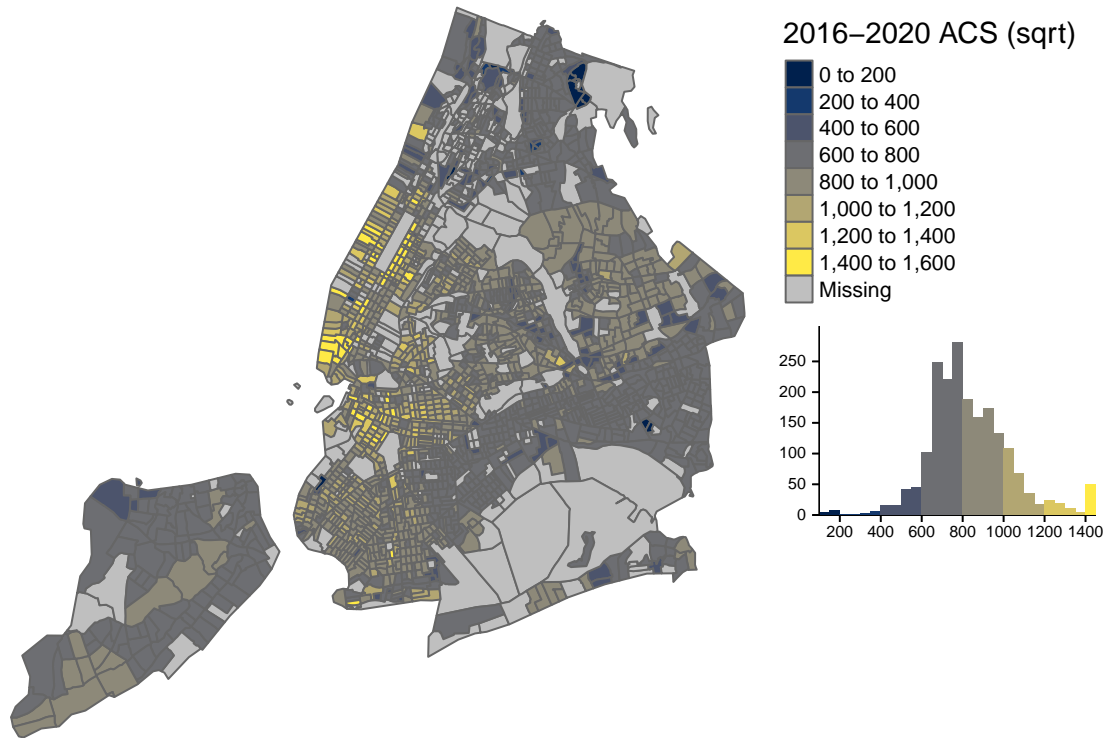
Notamos en el histograma que se presenta una asimetría a la derecha, existe una población no menor cuyos hogares perciben ingresos mucho mayores a la mediana, y vemos que existe data perdida, para el primer punto, podemos aplicar una transformación a la variable dependiente, tomando su raíz cuadrada para reducir esta asimetría a la derecha y tener los datos mejor distribuidos, acercándose más a una distribución normal, y en consecuencia, entregando resultados más precisos.

Transformación de los datos

Creamos la nueva variable tomando la raíz cuadrada de la variable dependiente median_valueE

```
# create plot
(nyc_median_value_sqrt_hist_tm <- nyc_census_data[!st_is_empty(nyc_census_data),,drop=F] %>% #
  mutate(sqrt_med_value = sqrt(median_valueE)) %>% #variable_sqrt
tm_shape() +
  tm_polygons(col = "sqrt_med_value",
    palette = "cividis",
    title = "2016-2020 ACS (sqrt)",
    legend.hist = TRUE) +
  tm_layout(main.title = "NYC Median Home Value by Census Tract",
    frame = FALSE,
    legend.outside = TRUE,
    bg.color = "grey100",
    legend.hist.width = 5,
  ))
```

NYC Median Home Value by Census Tract



```
# save plot
```

```
tmap_save(nyc_median_value_sqrt_hist_tm, "image/nyc_median_value_sqrt.png", width=1920, height=1080)
```

Preparando la data para modelar

Debemos borrar aquellas variables con data NA, y eliminar columnas con información del margen de error eliminando las columnas que terminan con “M”, se le quita la E final a las columnas como median_valueE y se añaden las variables pop_density y median_structure_age, esto pues buscamos transformar los predictores de manera que se represente mejor la relación que tiene con la variable de salida (en este caso, median_value)

- pop_density → mide densidad de población por metros cuadrados
- median_structure_age → resta 2020 de median_year_built

```
# load packages
#library(dplyr)
#library(stringr)
#library(units)
#library(stats)

# prep data for model
nyc_census_data_prepped <- nyc_census_data %>%
  mutate(pop_density = as.numeric(set_units(total_populationE / st_area(.),
    "1/km2")),
    median_structure_age = 2020 - median_year_builtE) %>%
  select(!ends_with("M")) %>% # drop margin of error cols
  rename_with(.fn = ~str_remove(.x, "E$")) %>% # remove E from col name
  na.omit() # omit NA

#Al parecer, entregan malos test debido a valor 0 en columna median_year_built

#Despues de realizar pruebas, se confirma que esta columna bugueada ensucia la data
#nyc_census_data_prepped <- filter(nyc_census_data_prepped, median_year_built != 0)
#Probamos manteniendo mas data pero eliminando las variables median_year_built
nyc_census_data_prepped <- nyc_census_data_prepped[ , !names(nyc_census_data_prepped) %in%
  c("median_year_built", "median_structure_age")]
```

Nota : La variable `median_year_built` posee alrededor de 800 filas en las que toma valor cero, esto ensucia la data `median_structure_age`, pues hace que permita que existan viviendas con tiempo promedio de construcción de 40 años, como viviendas con tiempo promedio de construcción de 2000 años, lo que es absurdo, se probó el modelo usando alrededor de 1100 datos con esta variable limpia, y por otro lado se probó con los 1967 datos sin hacer uso de esta variable. Los resultados que se adecuan más a la teoría expuesta en *Analyzing US Census* se dieron cuando no se toma en cuenta esta variable `median_year_built` y `median_structure_age`.

NAM <chr>	median_year_built <dbl>	geometry <S3: sfc_MULTIPOLYGON>
Census Tract 46, Bronx County, New York	1962	<S3: sfc_MULTIPOLYGON>
Census Tract 48, Bronx County, New York	0	<S3: sfc_MULTIPOLYGON>
Census Tract 50.01, Bronx County, New York	1944	<S3: sfc_MULTIPOLYGON>
Census Tract 50.02, Bronx County, New York	0	<S3: sfc_MULTIPOLYGON>
Census Tract 54, Bronx County, New York	0	<S3: sfc_MULTIPOLYGON>
Census Tract 56, Bronx County, New York	0	<S3: sfc_MULTIPOLYGON>
Census Tract 59.02, Bronx County, New York	0	<S3: sfc_MULTIPOLYGON>
Census Tract 60, Bronx County, New York	0	<S3: sfc_MULTIPOLYGON>
Census Tract 61, Bronx County, New York	1964	<S3: sfc_MULTIPOLYGON>
Census Tract 63.01, Bronx County, New York	0	<S3: sfc_MULTIPOLYGON>
Census Tract 64, Bronx County, New York	1960	<S3: sfc_MULTIPOLYGON>
Census Tract 68, Bronx County, New York	1964	<S3: sfc_MULTIPOLYGON>
Census Tract 70, Bronx County, New York	1951	<S3: sfc_MULTIPOLYGON>
Census Tract 71, Bronx County, New York	1999	<S3: sfc_MULTIPOLYGON>
Census Tract 72, Bronx County, New York	0	<S3: sfc_MULTIPOLYGON>
Census Tract 74, Bronx County, New York	1970	<S3: sfc_MULTIPOLYGON>

En las siguientes secciones se obtienen resultados distintos que en *Spatial Machine Learning - J. Morgan* debido a este cambio, pues trabajaremos con menos datos, pero con una variable que no hace ruido en el modelo.

En efecto sucede que para ciertos test, nos entrega mejores resultados respecto de la autocorrelación en los modelos `lagsar()`, cuando se ejecutó, daremos un comentario de este punto en dicha sección.

Modelo SLR y Spatial Autocorrelation

En esta sección pasaremos al desarrollo de modelos de predicción, para ello replicamos los modelos de predicción hecho en Spatial Machine Learning, Justin Morgan Williams y además, complementamos con el Capítulo 8 del libro Analyzing US Census Data - Modeling US Census Data (Walker K., 2023)[7] , capítulo donde se estudian conceptos durante la primera sección como índices de segregación y diversidad los cuales son usados en ciencias sociales para explicar patrones demográficos, en la segunda sección se estudian tópicos en modelamiento estadístico, incluyendo métodos de regresión con atributos espaciales, en donde tomamos en cuenta conceptos como la autocorrelación espacial que está inherente en la mayoría de las variables del censo; en la tercera sección del capítulo se estudian conceptos como clasificación, clusterización y regionalización, que son comunes en técnicas de Machine Learning. Nos enfocamos en particular en la segunda sección del capítulo.

Simple Linear Regression

Con la data preparada, podemos crear el primer modelo sencillo de regresión lineal, con la variable dependiente, la raíz cuadrada del valor medio de la vivienda (“NYC Median Home Value”)

```
# model formula
formula <- "sqrt(median_value) ~ median_rooms + median_income +
pct_college + pct_foreign_born + pct_white + pct_black + pct_hispanic +
pct_asian + median_age + percent_ooh + pop_density"

#Quitamos median_structure_age del modelo
# compute model
model1 <- lm(formula = formula, data = nyc_census_data_prepped)
summary(model1)
```

```
Call:
lm(formula = formula, data = nyc_census_data_prepped)

Residuals:
    Min       1Q   Median       3Q      Max
-657.80  -78.13    0.09   77.65  683.89

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.043e+02  3.983e+01  12.660 < 2e-16 ***
median_rooms  6.217e+01  6.476e+00   9.599 < 2e-16 ***
median_income  2.093e-03  1.706e-04  12.271 < 2e-16 ***
pct_college   1.693e+00  3.628e-01   4.667 3.27e-06 ***
pct_foreign_born -6.232e-01  3.249e-01  -1.918 0.055262 .
pct_white     2.424e+00  2.379e-01  10.190 < 2e-16 ***
pct_black     1.005e+00  2.076e-01   4.843 1.38e-06 ***
pct_hispanic  -1.397e-02  2.130e-03  -6.559 6.90e-11 ***
pct_asian     3.108e+00  3.002e-01  10.355 < 2e-16 ***
median_age    -2.170e+00  5.991e-01  -3.623 0.000298 ***
percent_ooh    -4.517e+00  2.810e-01 -16.071 < 2e-16 ***
pop_density    1.048e-03  3.404e-04   3.079 0.002109 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 146.9 on 1955 degrees of freedom
Multiple R-squared:  0.457,    Adjusted R-squared:  0.4539
F-statistic: 149.6 on 11 and 1955 DF, p-value: < 2.2e-16
```

Aquellas variables con mayor p-value son `pct_foreign_born`, `median_age`, `pop_density`, un p valor alto nos dice que la variable no tiene mucha significancia en el resultado (típicamente pedimos que sea menor a 0.05 para que la variable se considere significativa), notamos tambien que las primeras dos variables se correlacionan negativamente con `median_value`, es decir, a mayor cantidad de nacidos extranjeros y edad promedio, se tiene que el valor medio de la vivienda disminuye. Al contrario, si aumenta la densidad poblacional, notamos que el valor medio de la vivienda aumenta, podemos ver tambien el valor R^2 el cual nos dice que un 0.4124% de la varianza de `median_value` es explicado con las variables del modelo (el modelo es deficiente).

En la regresión lineal, los errores no son independientes en un modelo con componentes espaciales, esto es porque la autocorrelación espacial está presente en el error, lo que nos dice que el performance del modelo depende de la posición geográfica.

Haciendo uso de la librería **corrr** podemos calcular la matriz de correlaciones [7]

```
library(corrr)

nyc_estimates <- nyc_census_data_prepped %>%
  select(-GEOID, -median_value ) %>%
  st_drop_geometry()

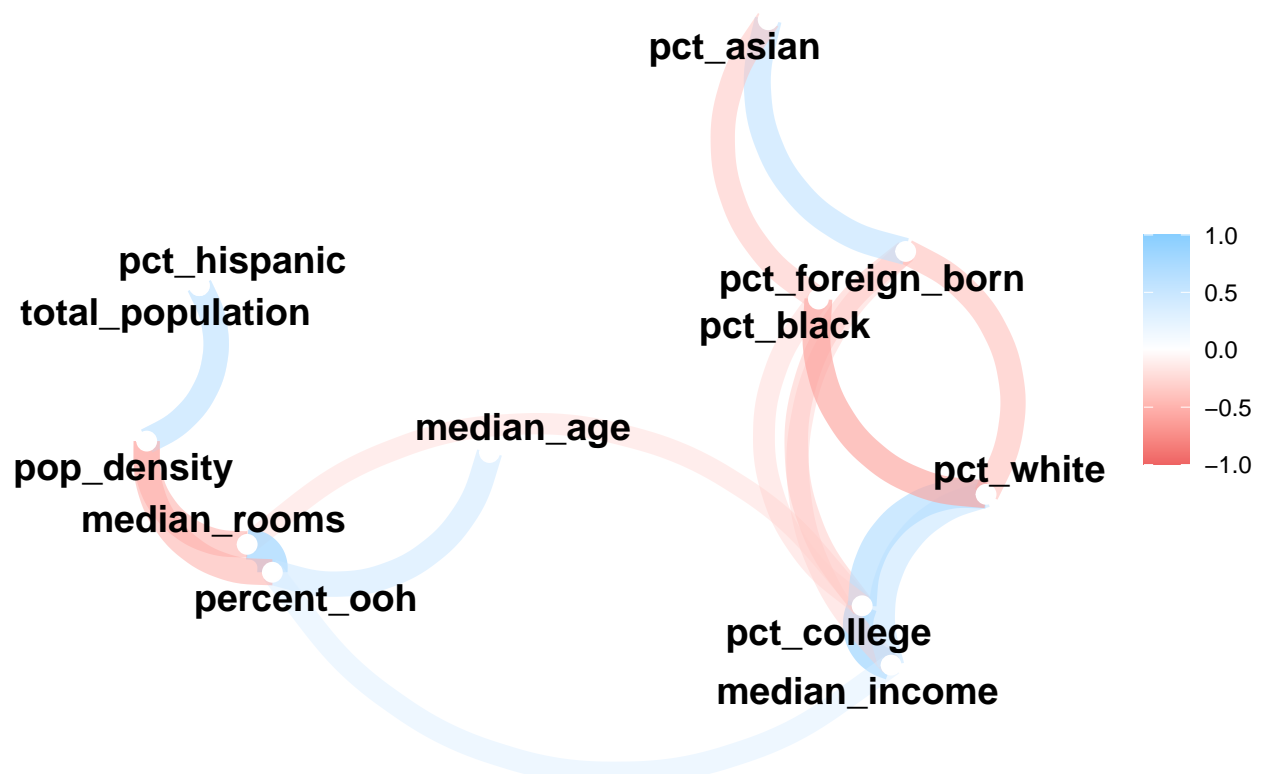
correlations <- correlate(nyc_estimates, method = "pearson")

correlations[is.na(correlations)] <- 0
```

	term	median_rooms	total_population	median_age	median_income	pct_college	pct_foreign_born
1	median_rooms	0.000000e+00	-2.755315e-01	1.862939e-01	8.841287e-02	-3.042464e-01	-3.492071e-02
2	total_population	-2.755315e-01	0.000000e+00	-1.053697e-01	-5.793610e-05	1.179050e-01	-1.067049e-01
3	median_age	1.862939e-01	-1.053697e-01	0.000000e+00	1.794448e-01	1.401042e-01	1.544935e-01
4	median_income	8.841287e-02	-5.793610e-05	1.794448e-01	0.000000e+00	7.502369e-01	-3.434698e-01
5	pct_college	-3.042464e-01	1.179050e-01	1.401042e-01	7.502369e-01	0.000000e+00	-3.935039e-01
6	pct_foreign_born	-3.492071e-02	-1.067049e-01	1.544935e-01	-3.434698e-01	-3.935039e-01	0.000000e+00
7	pct_white	-2.093989e-02	5.620708e-02	1.239277e-01	5.181632e-01	6.518286e-01	-4.723701e-01
8	pct_black	1.526903e-01	-1.021483e-01	-4.425499e-02	-2.146035e-01	-3.249169e-01	3.500025e-03
9	pct_hispanic	-2.755315e-01	1.000000e+00	-1.053697e-01	-5.793610e-05	1.179050e-01	-1.067049e-01
10	pct_asian	-3.019054e-02	-4.319723e-02	1.686201e-01	-4.314301e-02	-1.434416e-02	5.649252e-01
11	percent_ooh	7.479068e-01	-2.773636e-01	4.775084e-01	3.640753e-01	3.751176e-02	-3.817959e-02
12	pop_density	-5.257080e-01	4.827097e-01	-2.284217e-01	-6.815065e-02	1.562171e-01	6.102068e-04
13	residuals	-7.277834e-17	-6.226377e-17	-3.538226e-17	-2.886366e-17	-3.196077e-17	-4.313835e-17
14	lagged_residuals	-4.075953e-02	-3.706601e-02	-9.610488e-02	1.234156e-01	9.782962e-02	-1.749310e-02

Y lo visualizamos haciendo uso de `network_plot()`

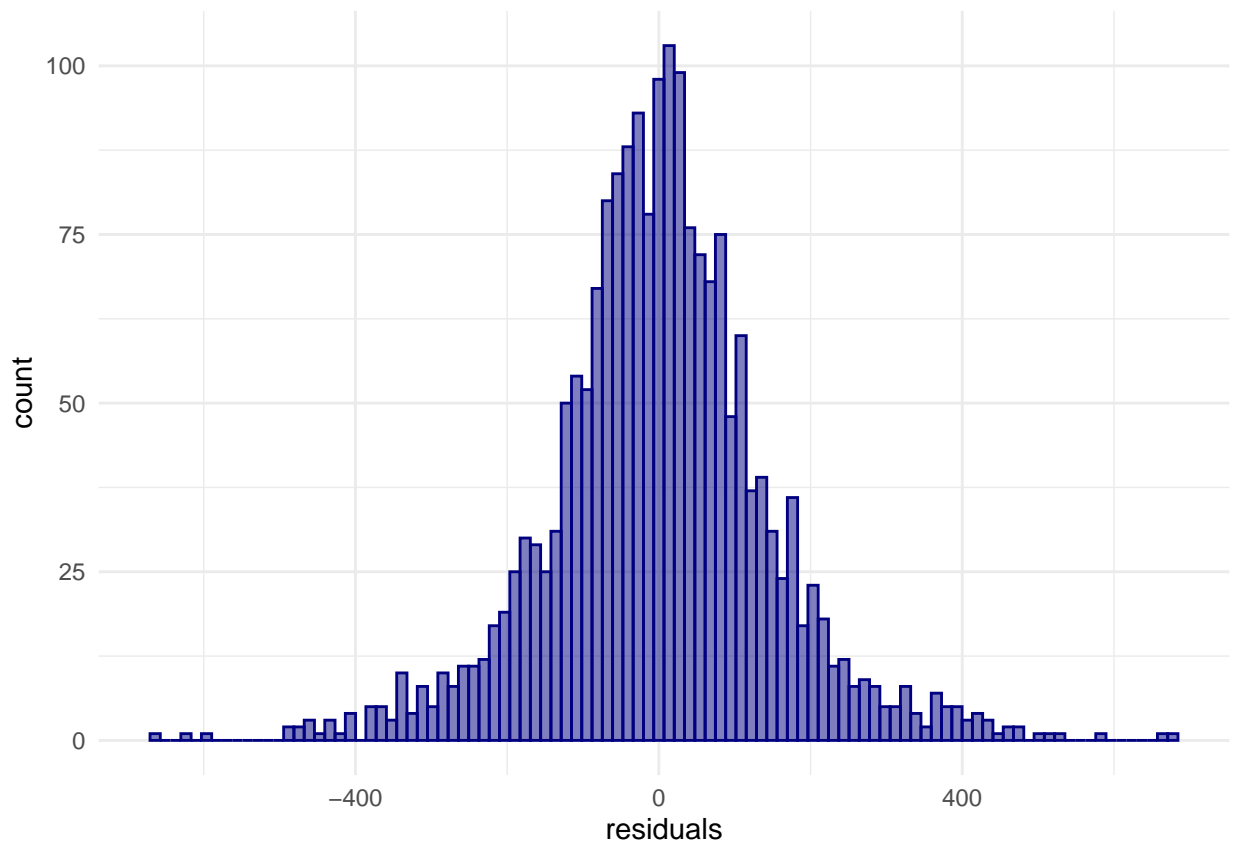
```
network_plot(correlations)
```



Nos interesa además los residuos del modelo, pues esperamos que estos sean explicados cuando tomamos en cuenta el fenómeno de spatial autocorrelation , para obtenerlos y añadirlos a nuestra data, seguimos como en la sección 8.3 de Analyzing US Census Data

```
#Añadimos los residuos a la data que estamos trabajando
nyc_census_data_prepped$residuals <- residuals(model1)

#Plot
ggplot(nyc_census_data_prepped, aes(x = residuals)) +
  geom_histogram(bins = 100, alpha = 0.5, color = "navy",
                fill = "navy") +
  theme_minimal()
```



Usaremos esta variable para realizar posteriormente un Test de Índice de Moran, para ello necesitamos hablar sobre lo que es Spatial data, el fenómeno de Spatial autocorrelation y el índice de Moran.

Spatial data

Definimos Spatial data como un conjunto de observaciones de data espacial (Jiang Z., 2019) [4]

$$\{(x(s_i), y(s_i)) \mid i \in \mathbb{N}, 1 \leq i \leq n\}$$

donde $n \in \mathbb{N}$ es la cantidad de muestras, $s_i \in \mathbb{R}^{2 \times 1}$ es un vector de coordenadas espaciales, $x(s_i) \in \mathbb{R}^m$ son las variables explicativas del modelo y $y(s_i) \in \mathbb{R}$ es la respuesta a dichas variables donde nuestra intención es predecir y .

Este mismo conjunto también puede ser descrito en formato matricial como $(X, Y) \in \mathbb{R}^{(m+1) \times n}$ donde $X = [x(s_1), x(s_2), \dots, x(s_n)]^T \in \mathbb{R}^{m \times n}$ (ya que $x(s_i) \in \mathbb{R}^m$) y $Y = [y(s_1), y(s_2), \dots, y(s_n)]^T \in \mathbb{R}^n$.

Por lo tanto, dado un conjunto de muestras de data espacial, buscamos modelar una función f tal que $Y = f(X)$, una vez que el modelo se entrena, puede usarse para predecir $y(s_i)$ dado $x(s_i)$.

La predicción espacial (Spatial prediction) tiene una cualidad única que la diferencia de la predicción usual en data mining en donde se asume que las muestras son independientes e idénticamente distribuidas (i.i.d) y por lo tanto dado un modelo reflejado en la función f podemos usarlo para predecir $y(s) = f(x(s))$ para todo s , sin embargo la suposición de que los datos son i.i.d no se cumple bajo la data espacial debido a la relación espacial implícita que existe entre posiciones cercanas en una región.

“Todo se relaciona con todo lo demás, pero las cosas más cercanas se relacionan más que las cosas distantes”, primera ley de la geografía (Tobler W.R, 1970) [6]

Spatial Autocorrelation

Usando spatial data recolectada, notamos que locaciones cercanas tienden a parecerse entre sí en lugar de ser estadísticamente independientes, llamamos a este fenomeno como efecto de autocorrelación, lo que hace que las muestras nos sean i.i.d, ignorar este factor puede producir modelos menos precisos, por ejemplo cuando aplicamos un modelo de regresión lineal, el error residual esta comunmente correlacionado.

La primera ley de geografía (Tobler W.R, 1970) [6] nos indica que en muestras con spatial data, estas no son estadísticamente independientes, si no que se encuentran correlacionadas, en particular sobre locaciones cercanas entre sí.

Índice de Moran

Para trabajar el problema de la autocorrelación espacial [4] [5] hacemos uso del índice de Moran el cual nos entrega una medida de como se comporta este fenómeno dado n observaciones de una variable, el índice de Moran es una auto-covarianza espacial estandarizada [2].

Definimos el índice de Moran como

$$I = \frac{\sum_{i=1}^n \sum_{j=1}^n w_{ij} (y(s_i) - \bar{y})(y(s_j) - \bar{y})}{(\sum_{i=1}^n \sum_{j=1}^n w_{ij}) \sum_{i=1}^n (y(s_i) - \bar{y})^2 / n}$$

O de manera equivalente

$$= \frac{n}{W} \frac{\sum_{i=1}^n \sum_{j=1}^n w_{ij} (y(s_i) - \bar{y})(y(s_j) - \bar{y})}{\sum_{i=1}^n (y(s_i) - \bar{y})^2}$$

donde $\bar{y} = \sum_{i=1}^n y(s_i)/n$ con n el número total de muestras, a partir de ahora, para simplificar la notación, $y(s_i) = y_i$, además $W = \sum_{i=1}^n \sum_{j=1}^n w_{ij}$, y abusando de la notación, $W = (w_{ij})_{i,j=1}^n$, que consideramos como una matriz de pesos.

Veamos que, bajo ciertas condiciones de la matriz W , se tiene que $I \in [-1, 1]$, en (Chen, Y. 2022)[1] tenemos una demostración de que el índice de Moran toma dichos valores cuando W clasifica como “globally normalized wight matrix”, pero no entra en detalles, además, se estudia que en la práctica muy típicamente para estas matrices, $I \in (-1, 1)$, en el manual de PQStat se habla de “Spatial weights matrix” como una matriz con coeficientes positivos y filas estandarizadas de manera las filas sumen uno, esto es $\sum_{j=1}^n w_{ij} = 1 \quad \forall i \in \{1, \dots, n\}$ (*row-normalized*).

En la discusión “Why is Moran’s I coming out greater than 1” - StackExchange se muestra un contraejemplo de matriz W cuya suma de las entradas es $\sum_{i,j} w_{ij} = 1$ y para muestras (X_1, \dots, X_4) sucede $I = 3$, por lo que si W no cumple la propiedad de ser *row-normalize*, no se puede saber a primeras los valores que puede tomar I , y por tanto, no tenemos una referencia clara de la significancia que pueda tener la magnitud de I en el modelo si no conocemos sus valores máximos y mínimos.

Veremos que se puede relajar la condición de que la suma de las filas sean todas iguales a 1 y pediremos que la suma de las filas sea un número α para cada fila, trabajaremos entonces con matrices de pesos estandarizadas que son las adecuadas para un modelo de autocorrelación

Presentamos una demostración en la que no usaremos técnicas de cálculo ni de optimización para probar que para W una matriz de pesos estandarizada, tenemos que $I \in [-1, 1]$

Entenderemos por matriz de pesos estandarizada aquella que cumple las siguientes propiedades

- Supondremos $w_{ij} \geq 0 \quad \forall i, j \in \{1, \dots, n\}$, pues w_{ij} es una magnitud de la influencia que hay entre los vecinos i y j
- Típicamente, $w_{ii} = 0 \quad \forall i \in \{1, \dots, n\}$, pues el hecho de tener que un punto tenga influencia con el mismo puede generar ruido en el modelo
- Suponemos que la influencia que tiene y_i sobre y_j es la misma que la de y_j sobre y_i , esto es, $w_{ij} = w_{ji} \quad \forall i, j \in \{1, \dots, n\}$, es decir, W es simétrica
- *Row-normalized*: Para cada $i \in \{1, \dots, n\}$ se tiene que $\sum_{j=1}^n w_{ij} = \alpha$ para algún $\alpha \in \mathbb{R}$, esto pues, entendemos la suma $\sum_{j=1}^n w_{ij}$ como la influencia total que existe entre el vecino i y todos sus vecinos j , por lo que pedimos que la influencia total sea la misma para cada vecino $i \in \{1, \dots, n\}$, de manera que el nivel de influencia esté estandarizado para cada vecino.

Haremos uso de los siguientes resultados conocidos

- **Teorema:** Sea A una matriz cuadrada simétrica a coeficientes reales, tenemos que A es ortogonalmente diagonalizable, es decir, podemos escribir $A = PDP^T$, con P matriz ortogonal cuyos vectores columna son los vectores propios de A y D es matriz diagonal con $(d_{ii})_{i=1}^n = (\lambda_i)_{i=1}^n$ valores propios de A
- **Lema:** Si P es ortogonal, entonces $P^T = P^{-1}$ y además, P es una isometría en $(\mathbb{R}^n, \|\cdot\|_2)$, es decir, $\|Pw\|_2 = \|w\|_2 \quad \forall w \in \mathbb{R}^n$

Con ello, podemos probar el siguiente teorema

- **Teorema:** Se tiene que el operador $\lambda_{max} : S^n \rightarrow \mathbb{R}$, que asocia a cada $A \in S^n$ matriz simétrica su mayor valor propio $\lambda_{max}(A)$, es convexa, más aún, $\lambda_{max}(A) = \sup_{\|v\|=1} v^T A v$ supremo de funcionales lineales sobre S^n .

Además haremos uso del siguiente lema

- **Lema:** El mayor valor propio de una matriz A con coeficientes no negativos está acotada por la mayor suma de las filas, esto es, $\lambda_{max}(A) \leq \max_i \sum_{j=1}^n a_{ij}$.
- **Demostración del lema:** Sea λ un valor propio de una matriz A no negativa asociada a un vector x , tenemos entonces que

$$\begin{aligned} Ax = \lambda x &\implies \lambda |x_i| = \left| \sum_{j=1}^n a_{ij} x_j \right| \quad \forall i \in \{1, \dots, n\} \\ \implies \lambda |x_i| &= \left| \sum_{j=1}^n a_{ij} x_j \right| \leq \left(\sum_{j=1}^n |a_{ij}| \right) \max_j |x_j| \quad \forall i \in \{1, \dots, n\} \end{aligned}$$

Usando que $a_{ij} \geq 0 \quad \forall i, j \in \{1, \dots, n\}$ y que la desigualdad se cumple para todo i tenemos

$$\begin{aligned} \lambda \max_i |x_i| &\leq \max_i \left(\sum_{j=1}^n a_{ij} \max_j |x_j| \right) = \max_i \left(\sum_{j=1}^n a_{ij} \right) \max_j |x_j| \\ \implies \lambda &\leq \max_i \left(\sum_{j=1}^n a_{ij} \right) \end{aligned}$$

Como λ era arbitrario deducimos que

$$\lambda_{max}(A) \leq \max_i \left(\sum_{j=1}^n a_{ij} \right)$$

Enunciamos entonces

- **Propiedad:** Para $I = \frac{n}{W} \frac{\sum_{i=1}^n \sum_{j=1}^n w_{ij} (y_i - \bar{y})(y_j - \bar{y})}{\sum_{i=1}^n (y_i - \bar{y})^2}$ donde $\bar{y} = \sum_{i=1}^n y_i / n$ con W matriz de pesos estandarizada e $y \in \mathbb{R}^n$, se cumple que $I \in [-1, 1]$.
- **Demostración:** Consideremos el operador $\Phi : \mathbb{R}^n \rightarrow \partial B(0, 1)$ definido por $\Phi(y) = z = (z_i)_{i=1}^n = (\frac{(y_i - \bar{y})}{\sqrt{(\sum_{i=1}^n (y_i - \bar{y})^2)}})_{i=1}^n$

Está bien definida pues $\forall y \in \mathbb{R}^n$

$$\begin{aligned} \|\Phi(y)\|_2 = \|z\|_2 &= \sum_{i=1}^n |z_i|^2 = \sum_{i=1}^n \frac{(y_i - \bar{y})^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \\ &= \frac{\sum_{i=1}^n (y_i - \bar{y})^2}{\sum_{i=1}^n (y_i - \bar{y})^2} = 1 \end{aligned}$$

Notamos que

$$\begin{aligned} I &= \frac{n \sum_{i=1}^n \sum_{j=1}^n w_{ij} (y_i - \bar{y})(y_j - \bar{y})}{W (\sum_{i=1}^n (y_i - \bar{y})^2)} \\ &= \frac{n}{W} \sum_{i=1}^n \sum_{j=1}^n w_{ij} \frac{(y_i - \bar{y})(y_j - \bar{y})}{\sqrt{\sum_{i=1}^n (y_i - \bar{y})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} \\ &= \frac{n}{W} \sum_{i=1}^n \sum_{j=1}^n w_{ij} \Phi(y)_i \Phi(y)_j = \frac{n}{W} \sum_{i=1}^n \sum_{j=1}^n w_{ij} z_i z_j \end{aligned}$$

Definimos $V = (\frac{w_{ij}}{W})_{i,j}^n$ matriz normalizada tal que $\sum_{i,j} v_{ij} = 1$, además, como W es matriz de pesos estandarizada, tenemos que V también lo es, por lo que es simétrica, no negativa, y row-normalized.

Entonces, recordando que $\|z\|_2 = 1$

$$\begin{aligned} I &= n(z^T V z) \leq n \sup_{\|z\|=1} z^T V z = n \lambda_{\max}(V) \\ \implies |I| &\leq n |\lambda_{\max}(V)| \leq n \max_i (\sum_{j=1}^n v_{ij}) \leq n \max_i (\sum_{j=1}^n |v_{ij}|) \end{aligned}$$

Por un lado, tenemos que $|v_{ij}| = v_{ij}$ y como $\sum_{i=1}^n \sum_{j=1}^n v_{ij} = 1$ y V es *row-normalized* tenemos que se cumple $\sum_{j=1}^n v_{ij} = \frac{1}{n} \quad \forall i \in \{1, \dots, n\}$, y por lo tanto

$$\implies |I| \leq n \max_i (\sum_{j=1}^n v_{ij}) = n * \frac{1}{n} = 1$$

$$\implies I \in [-1, 1]$$

En caso de que W no cumpla la propiedad *row-normalized* de igual manera podemos encontrar una cota para I y esta es $I \in [-n, n]$ donde n es la cantidad total de datos haciendo uso de la desigualdad de Schur [3] que nos dice

- **Teorema:** Sea A una matriz $n \times n$ con valores propios $(\lambda_1, \dots, \lambda_n)$, entonces se cumple

$$\sum_{i=1}^n |\lambda_i|^p \leq \sum_{i,j=1}^n |a_{ij}|^p, \quad 1 \leq p < 2$$

Luego, como vimos antes, tenemos ahora V una matriz normalizada, pero no necesariamente *row-normalized*, luego

$$I = n(z^T V z)$$

Y como V es simétrica, la escribimos como $V = P^T D P$ usando el teorema espectral, y como P es ortogonal, entonces $x = Pz$ tiene norma $\|Pz\|_2 = \|x\|_2 = 1$, en particular $|x_i|^2 \leq 1 \quad \forall i \in \{1, \dots, n\}$ luego

$$\begin{aligned} I &= n(z^T P^T D P z) = n(x^T D x) = n \sum_{i=1}^n \lambda_i(V) |x_i|^2 \\ \implies |I| &\leq n \sum_{i=1}^n |\lambda_i(V)| \leq n \sum_{i,j=1}^n |v_{ij}| = n \\ \implies I &\in [-n, n] \end{aligned}$$

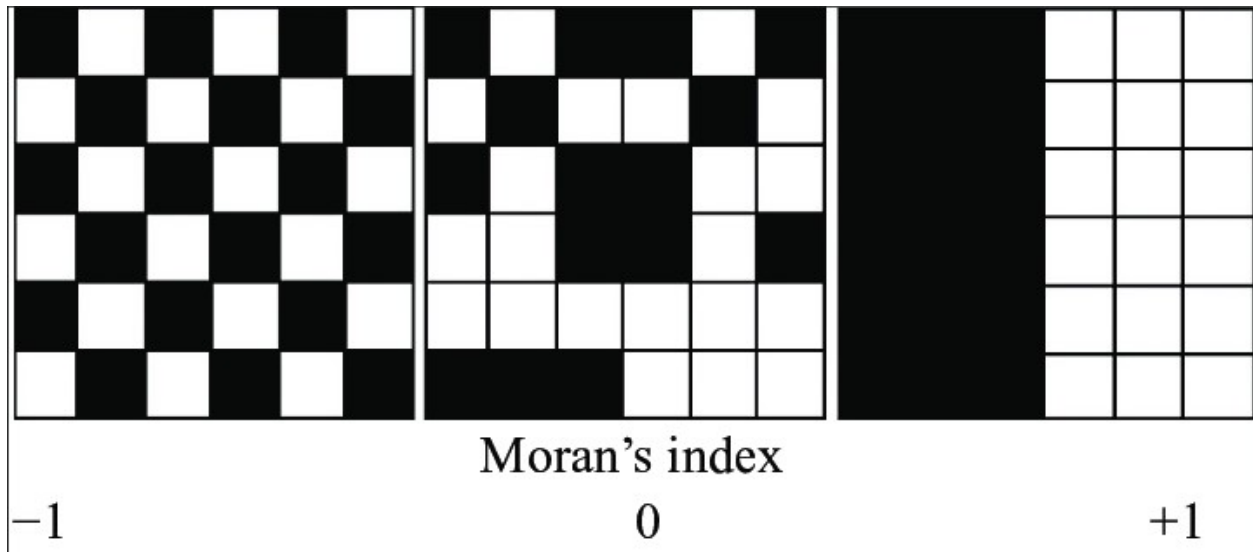
Sin embargo, usaremos matrices de peso estandarizadas pues son las que mejor se ajustan al modelo, y donde sabemos que el índice de Moran se mueve entre $[-1, 1]$.

Si sabemos el intervalo en el que se mueve I para nuestro modelo, podemos saber que tan significativo es el valor, y por lo tanto que tan presente esta el fenómeno de autocorrelación espacial.

Comentando sobre el significado de este valor, al igual que la correlación, en donde vemos la relación entre 2 variables de entrada cuando vemos el cambio que tienen respecto de una variable de respuesta, la autocorrelación es similar pero la variable de respuesta es la misma variable de entrada (Fuente).

El índice de Moran mide que tan similar es una variable respecto de las variables cercanas, por ejemplo, si este se acerca a -1 nos dice que cada variable tiende a tener valores distinto de sus alrededores, y caso contrario si se acerca a 1 , resumiendo

- $I \approx -1 \rightarrow$ Se clusterizan valores distintos entre sí (dispersión perfecta)
- $I \approx 0 \rightarrow$ No existe autocorrelación espacial (podemos pensar que los datos son independientes idénticamente distribuidos en el espacio)
- $I \approx 1 \rightarrow$ Se clusterizan valores similares entre sí



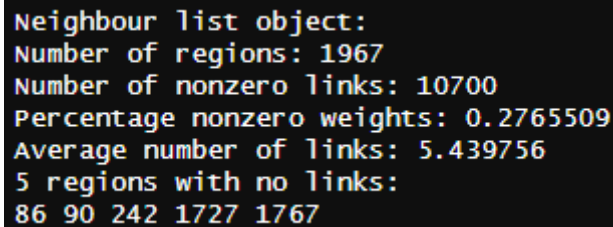
Spatial Weights Matrix

Seguimos el modelo, para esta parte nos guiamos con el capítulo 7 Analyzing US Census Data - Spatial analysis with US Census Data

Buscamos armar una matriz de pesos, para medir la interacción entre cada tramo censal, para ello generamos una “neighborhood list” en R con el paquete **spdep** usando la función `poly2nb()`, esta función nos presenta varias formas de catalogar a los vecinos, para el modelo usaremos

- *Contiguity-based neighbors* → se usa cuando la geometría es tipo polygon, las opciones incluyen neighbors tipo “Queen”, que se basa en que todos los polygons que compartan un vértice se consideren vecinos, y neighbors tipo “Rook”, en donde deben compartir al menos un segmento de línea para ser vecinos, podemos inferir que neighbors tipo “Queen” abre diagonales de manera que tiene menos entradas iguales a cero

```
# create neighbor list object
nyc_nb <- spdep::poly2nb(nyc_census_data_prepped, queen = TRUE)
nyc_nb
```



```
Neighbour list object:
Number of regions: 1967
Number of nonzero links: 10700
Percentage nonzero weights: 0.2765509
Average number of links: 5.439756
5 regions with no links:
86 90 242 1727 1767
```

Interpretamos la información de la siguiente manera

- Tenemos 1967 distritos censales en NYC (aquellos con valor NA se omiten)
- El porcentaje de las entradas $w_{i,j}$ tales que $w_{i,j} \neq 0$ es 0.2765%
- El número promedio de conexiones entre distritos censales, el promedio de cuantas conexiones posee cada distrito es 5.43
- Existen 5 distritos que no se conectan con nadie.

Podemos hacer un plot de la estructura que posee el objeto “neighborhood list”, para visualizarlo usamos funciones “st_” descritas en Analyzing US Census.

```
# filter for si
si <- nyc_census_data_prepped[str_detect(nyc_census_data_prepped$NAM, "Richmond"),]

# store geometry of polygons
si_geom <- st_geometry(si)

# centroid de cada tramo censal
si_centroids = st_centroid(si_geom)
si_coordinates = st_coordinates(si_geom)

# create neighbor list
si_nb_queen <- spdep::poly2nb(si)
si_nb_rook <- spdep::poly2nb(si, queen=FALSE)

# save plot as jpeg
jpeg("si_neighborhood_rook_queen.jpg", width = 1000, height = 800)

# set plot structure, 2 plots en la misma imagen
par(mfrow = c(1,2))

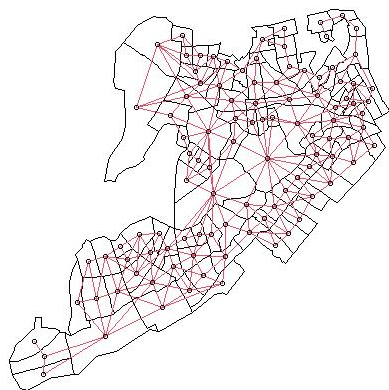
# plot queen
plot(si_geom,
      main = "Queen",
      reset = FALSE,
      cex.main = 3)
plot(si_nb_queen, si_centroids,
      add = TRUE,
      col = 2,
      lwd = 1.5)

# plot rook
plot(si_geom,
      main = "Rook",
      reset = FALSE,
      cex.main = 3)
plot(si_nb_rook, si_centroids,
      add = TRUE,
      col = 2,
      lwd = 1.5)

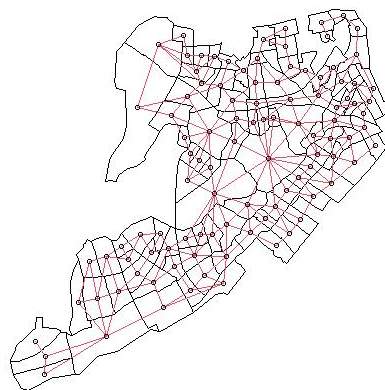
# close jpeg
dev.off()
```

Guardada la imagen con las conexiones entre cada distrito censal con los modelos “Queen” y “Rook” procedemos a mostrar los plots.

Queen



Rook



En efecto, observamos la presencia de conexiones diagonales para el modelo “Queen” a diferencia de la estructura de “Rook”, así como en ajedrez, la reina puede moverse en todas direcciones, mientras que la torre solo puede moverse en vertical y horizontal.

Test de Índice de Moran

Dadas la “neighborhood list” antes dada, podemos usarla para crear una matriz de pesos (spatial weights list) W usando la función de `spdep nb2listw()`, que nombramos `wts`, además para realizar el test de Índice de Moran, necesitamos ignorar aquellos datos que no posean vecinos, para ello usamos la librería `spatialreg` para verificar que se tiene activado la opción `zero.policy` que nos permite ignorar estos datos.

```
# create spatial weights matrix
library(spatialreg)
library(spdep)
set.ZeroPolicyOption(TRUE)
get.ZeroPolicyOption()
wts <- spdep::nb2listw(nyc_nb)
```

Visualizando a `wts`

<code>wts[["weights"]]</code>	<code>list [1967]</code>	List of length 1967
<code>[[1]]</code>	double [4]	0.25 0.25 0.25 0.25
<code>[[2]]</code>	double [7]	0.143 0.143 0.143 0.143 0.143 0.143 ...
<code>[[3]]</code>	double [6]	0.167 0.167 0.167 0.167 0.167 0.167
<code>[[4]]</code>	double [2]	0.5 0.5
<code>[[5]]</code>	double [4]	0.25 0.25 0.25 0.25
<code>[[6]]</code>	double [2]	0.5 0.5
<code>[[7]]</code>	double [3]	0.333 0.333 0.333
<code>[[8]]</code>	double [3]	0.333 0.333 0.333
<code>[[9]]</code>	double [5]	0.2 0.2 0.2 0.2 0.2
<code>[[10]]</code>	double [2]	0.5 0.5

Vemos que es una matriz cuyas filas tienen por suma total 1 (row normalized)

Con ella podemos realizar un test de Índice de Moran para observar autocorrelación espacial, para ello hacemos uso de la variable de residuos obtenida en secciones anteriores

```
# perform Moran's I test
spdep::moran.test(nyc_census_data_prepped$residuals, wts)
```

```
Moran I test under randomisation

data:  nyc_census_data_prepped$residuals
weights: wts  n reduced by no-neighbour observations

Moran I statistic standard deviate = 24.186, p-value < 2.2e-16
alternative hypothesis: greater
sample estimates:
Moran I statistic      Expectation      Variance
    0.3456912899      -0.0005099439      0.0002048902
```

Notamos el $p\text{-value} < 2.2e - 16$ lo que nos habla de la significancia que posee en el modelo

Además, vemos que el índice de Moran tiene un valor positivo, lo que nos dice que en el mapa, distritos censales con atributos similares tienden a agruparse (clusters).

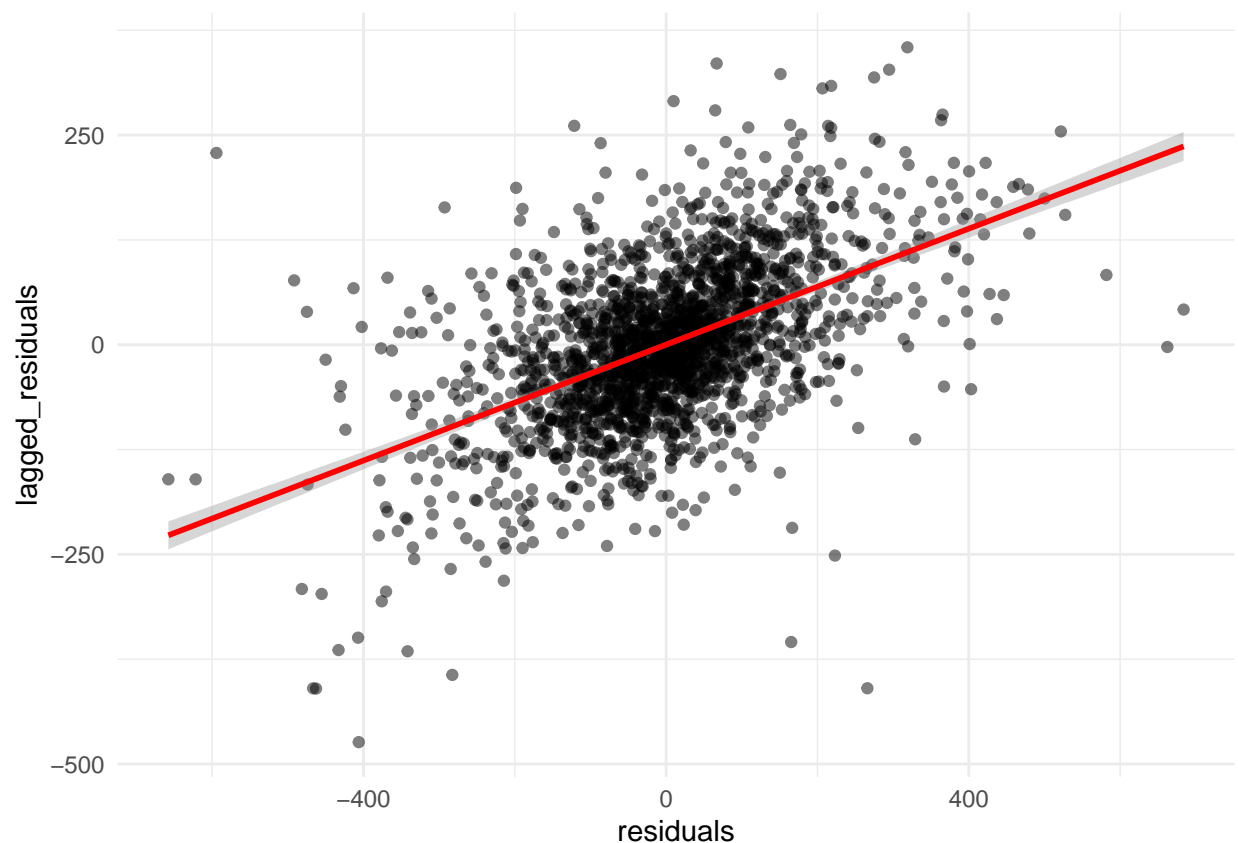
En efecto, podemos apreciar como en NYC, con la variable “Median Home Value” se forman agrupaciones de distritos en ciertos sectores del estado que comparten la cualidad de tener un alto valor de la vivienda, y a su vez, otros sectores con distritos que comparten tener un bajo nivel de la vivienda.

Un índice de Moran de $I = 0.345$ es estadísticamente significativo (Walker K., 1970)[7]

Observamos los residuos, hacemos uso de la función `lag.listw()` la cual nos toma en cuenta los residuos del modelo junto a la matriz de peso que estamos añadiendo.

```
# get lagged version of residuals for plot
nyc_census_data_prepped$lagged_residuals <-
lag.listw(wts, nyc_census_data_prepped$residuals)

# plot
(morans_i_res <- nyc_census_data_prepped %>%
ggplot(aes(x = residuals, y = lagged_residuals)) +
  theme_minimal() +
  geom_point(alpha = 0.5) +
  geom_smooth(method = "lm", color = "red"))
```



```
# save image
ggsave("image/morans_i_res.png",
  plot = morans_i_res)
```

El plot muestra una correlación positiva entre el fenómeno de autocorrelación espacial y los residuos, lo que sugiere rechazar la independencia en los términos del error pues existe autocorrelación espacial en los residuos.

Para trabajar este problema, tenemos métodos para tratar con regresión espacial

Spatial Regression Model

En esta sección se aplica el dataset junto a 5 modelos distintos de regresión espacial con los diferentes paquetes que proporciona R.

Cada uno de ellos son adaptaciones de distintas técnicas de Machine Learning para hacer frente a los problemas asociados a data espacial.

Simultaneous Autoregressive (SAR) models

Los siguientes dos modelos SAR que vienen incluidos en el paquete **spatialreg** se enfocan en añadir términos para abordar la autocorrelación espacial

- `spatialreg::lagsarlm()` → da cuenta de la dependencia espacial al incluir un “spatial lag” de la variable dependiente (outcome) del modelo. Esto explica los efectos indirectos espaciales de los valores de los vecindarios (neighboring areas) que influyen en los valores dentro de una ubicación determinada.
- `spatialreg::errorsarlm()` → incluye “spatial lag” en la medida del error del modelo. Esto está diseñado para capturar procesos espaciales latentes que no se tienen en cuenta en la estimación del modelo y, a su vez, aparecen en los residuos del modelo.

A pesar de que estos modelos dan cuenta de la autocorrelación espacial, asumen que todos los coeficientes son iguales y, por lo tanto, no abordan la heterogeneidad espacial (spatial heterogeneity) que es otra problemática asociada a la data espacial como se vio en secciones anteriores. Por el contrario, el método Geographically Wighted Regression (GWR) toma en cuenta la heterogeneidad espacial.

Geographically Weighted Regression (GWR)

Los siguientes dos modelos abordan la heterogeneidad espacial mediante GWR, aprendiendo un conjunto de parámetros del modelo en cada ubicación. Los inconvenientes son el coste computacional y la suposición de “spatial isotropy”. La isotropía espacial es la suposición de que, independientemente de la dirección en que nos movamos, las propiedades son las mismas.

- `GWModel::gwr.basic()` → modelo simple de regresión ponderada geográficamente (geographically weighted regression) diseñado para evaluar las variaciones locales en los resultados de los modelos de regresión dada una función de pesos. Típicamente funciones que decaen con la distancia.
- `spgwr::gwr()` → implementa el enfoque básico de regresión ponderada geográficamente (GWR) para explorar la “no estacionariedad espacial” para un ancho de banda global dado y un esquema de ponderación elegido (weighting scheme).

Geographically Weighted Random Forest (GRF)

El siguiente modelo aborda la heterogeneidad espacial o, más específicamente, la “no estacionariedad” espacial de los datos, mediante el uso de una versión local de algoritmos de Machine Learning. Según la documentación del paquete, la diferencia entre un GWR lineal y un GRF es que podemos modelar la “no estacionariedad” junto con un modelo no lineal, lo que vuelve menos restrictiva la función que queremos modelar.

- `SpatialML::grf()` → implementa una extensión espacial del algoritmo de Random Forest (RF) que incluye un “bosque aleatorio ponderado geográficamente” en un conjunto de modelos locales.

Aplicación de modelos espaciales

SAR models

Con el paquete **spatialreg** hacemos uso de `lagsarlm()`, el formato para crear el modelo es similar a los usados en SLR, usamos como parámetro `listw` (spatial weights list) nuestra variable `wts` creada para el test de índice de Moran.

```
# lag model
lag_model <- spatialreg::lagsarlm(
  formula = formula,
  data = nyc_census_data_prepped,
  listw = wts,
  zero.policy=TRUE #Importante omitir valores NA
)

summary(lag_model, Nagelkerke = TRUE)
```

```
Call:spatialreg::lagsarlm(formula = formula, data = nyc_census_data_prepped,
  listw = wts, zero.policy = TRUE)

Residuals:
    Min       1Q   Median       3Q      Max
-634.2803  -63.0357  -1.2368   60.1508  659.3921

Type: lag
Regions with no neighbours included:
 86 90 242 1727 1767
Coefficients: (asymptotic standard errors)
              Estimate Std. Error z value Pr(>|z|)
(Intercept)   1.3963e+02  3.5889e+01  3.8905  0.000100
median_rooms   5.3824e+01  5.3865e+00  9.9924 < 2.2e-16
median_income  1.3199e-03  1.4287e-04  9.2387 < 2.2e-16
pct_college    8.6689e-01  3.0351e-01  2.8562  0.004288
pct_foreign_born -1.4501e-01  2.7003e-01 -0.5370  0.591251
pct_white      1.0520e+00  2.0021e-01  5.2547  1.483e-07
pct_black      3.4974e-01  1.7213e-01  2.0318  0.042170
pct_hispanic   -8.7347e-03  1.7695e-03 -4.9363  7.961e-07
pct_asian      1.4128e+00  2.5225e-01  5.6006  2.136e-08
median_age     -9.5120e-01  4.9750e-01 -1.9120  0.055882
percent_ooh    -3.1195e+00  2.3748e-01 -13.1358 < 2.2e-16
pop_density    2.0708e-04  2.8214e-04  0.7340  0.462970

Rho: 0.54755, LR test value: 589.09, p-value: < 2.22e-16
Asymptotic standard error: 0.020512
z-value: 26.694, p-value: < 2.22e-16
Wald statistic: 712.58, p-value: < 2.22e-16

Log likelihood: -12305.05 for lag model
ML residual variance (sigma squared): 14813, (sigma: 121.71)
Nagelkerke pseudo-R-squared: 0.5975
Number of observations: 1967
Number of parameters estimated: 14
AIC: 24638, (AIC for lm: 25225)
LM test for residual autocorrelation
test value: 1.4586, p-value: 0.22716
```

Por otra parte, usando `errorsarlm()`

```
# error model
error_model <- spatialreg::errorsarlm(
  formula = formula,
  data = nyc_census_data_prepped,
  listw = wts,
  zero.policy=TRUE
)

summary(error_model, Nagelkerke = TRUE)
```

```
Call:spatialreg::errorsarlm(formula = formula, data = nyc_census_data_prepped,
  listw = wts, zero.policy = TRUE)

Residuals:
    Min       1Q   Median       3Q      Max
-615.5586  -63.6194   -3.2227   59.4714  647.4568

Type: error
Regions with no neighbours included:
 86 90 242 1727 1767
Coefficients: (asymptotic standard errors)
              Estimate Std. Error z value Pr(>|z|)
(Intercept)  5.9749e+02  4.0931e+01  14.5974 < 2.2e-16
median_rooms  5.9771e+01  6.1166e+00   9.7720 < 2.2e-16
median_income 1.1222e-03  1.6138e-04   6.9540 3.551e-12
pct_college   1.5110e+00  3.7415e-01   4.0384 5.382e-05
pct_foreign_born -4.8325e-01  3.7538e-01  -1.2874 0.197969
pct_white     9.5876e-01  3.1428e-01   3.0506 0.002284
pct_black    -4.9192e-01  3.0915e-01  -1.5912 0.111567
pct_hispanic  -6.0655e-03  2.0010e-03  -3.0312 0.002436
pct_asian     1.2737e+00  3.9175e-01   3.2515 0.001148
median_age   -8.8719e-01  5.6784e-01  -1.5624 0.118196
percent_ooh  -3.1395e+00  2.5721e-01 -12.2061 < 2.2e-16
pop_density  -5.3785e-04  3.5265e-04  -1.5252 0.127221

Lambda: 0.66672, LR test value: 551.41, p-value: < 2.22e-16
Asymptotic standard error: 0.019944
z-value: 33.43, p-value: < 2.22e-16
Wald statistic: 1117.6, p-value: < 2.22e-16

Log likelihood: -12323.88 for error model
ML residual variance (sigma squared): 14469, (sigma: 120.29)
Nagelkerke pseudo-R-squared: 0.58972
Number of observations: 1967
Number of parameters estimated: 14
AIC: 24676, (AIC for lm: 25225)
```

Podemos hacer una comparación entre el primer modelo SLR y estos dos nuevos modelos usando el paquete **jtools** y **huxtable** usando la función `export_summs()`

```
#compare simple linear model with SAG models
library(jtools)
library(huxtable)
jtools::export_summs(model1, lag_model, error_model)
```

	Model 1	Model 2	Model 3
(Intercept)	504.29 *** (39.83)	139.63 *** (35.89)	597.49 *** (40.93)
median_rooms	62.17 *** (6.48)	53.82 *** (5.39)	59.77 *** (6.12)
median_income	0.00 *** (0.00)	0.00 *** (0.00)	0.00 *** (0.00)
pct_college	1.69 *** (0.36)	0.87 ** (0.30)	1.51 *** (0.37)
pct_foreign_born	-0.62 (0.32)	-0.15 (0.27)	-0.48 (0.38)
pct_white	2.42 *** (0.24)	1.05 *** (0.20)	0.96 ** (0.31)
pct_black	1.01 *** (0.21)	0.35 * (0.17)	-0.49 (0.31)
pct_hispanic	-0.01 *** (0.00)	-0.01 *** (0.00)	-0.01 ** (0.00)
pct_asian	3.11 *** (0.30)	1.41 *** (0.25)	1.27 ** (0.39)
median_age	-2.17 *** (0.60)	-0.95 (0.50)	-0.89 (0.57)
percent_ooH	-4.52 *** (0.28)	-3.12 *** (0.24)	-3.14 *** (0.26)
pop_density	0.00 ** (0.00)	0.00 (0.00)	-0.00 (0.00)
rho		0.55 *** (0.02)	
lambda			0.67 *** (0.02)
N	1967	1967	1967
R2	0.46	0.63	0.65

*** p < 0.001; ** p < 0.01; * p < 0.05.

Notamos que en cuando al test de R^2 , estadística que mide la proporción de varianza de la variable dependiente explicada por las variables independientes que usamos en el modelo de regresión, se tiene acorde al desempeño

- Model 1 - `lm()` → $R^2 = 0.46$
- Model 2 - `spatialreg::lagsarlm()` → $R^2 = 0.63$
- Model 3 - `spatialreg::errorsarlm()` → $R^2 = 0.65$

Comentario: Notamos los mismos valores de R^2 (a excepción del Modelo 1 por muy poco) que aquellos replicados en Spatial Machine Learning - J. Morgan cuando lo trabajamos sin tomar en cuenta la variable `median_year_built`

Test de índice de Moran con modelos auto-regresivos (SAR)

Dada las propiedades de estos modelos, esperamos una disminución del efecto de autocorrelación espacial en los residuos del modelo, los resultados son los siguientes

```
# lag model Moran's I test
spdep::moran.test(lag_model$residuals, wts)
```

Lag model

```

      Moran I test under randomisation

data:  lag_model$residuals
weights: wts  n reduced by no-neighbour observations

Moran I statistic standard deviate = -0.65869, p-value = 0.745
alternative hypothesis: greater
sample estimates:
Moran I statistic      Expectation      Variance
-0.0099353756      -0.0005099439      0.0002047554
```

```
# error model Moran's I test
spdep::moran.test(error_model$residuals, wts)
```

Error model

```

      Moran I test under randomisation

data:  error_model$residuals
weights: wts  n reduced by no-neighbour observations

Moran I statistic standard deviate = -3.3466, p-value = 0.9996
alternative hypothesis: greater
sample estimates:
Moran I statistic      Expectation      Variance
-0.0483965461      -0.0005099439      0.0002047533
```

Notamos que en ambos modelos se minimiza el valor que toma el índice de Moran, el p-valor aumenta considerablemente sobre $p > 0.05$, lo que disminuye la significancia de la autocorrelación del modelo. En particular `error_model` tiene mejor desempeño haciendo frente a la autocorrelación espacial en comparación a `lag_model`.

GWR models

Estudiamos dos modelos de regresión ponderada geográficamente (GWR). Estos modelos abordan el problema de la heterogeneidad espacial aprendiendo un modelo distinto en cada ubicación de punto de datos, creando así parámetros de modelo dependientes de la ubicación.

GWModel Primero tenemos el modelo `gwr.basic()` del paquete `GWmodel`. El preprocesamiento consiste en convertir el conjunto de datos en un objeto `SpatialPolygons` `sp` (no admite objetos `sf`) y en seleccionar un “bandwidth” (ancho de banda). El concepto de “kernel bandwidth” se utiliza en GWR para calcular un modelo de regresión local para cada tramo censal. Se basa en el tipo de kernel, fijo o adaptativo, y en una función que decae con la distancia. El kernel adaptativo utiliza los vecinos más próximos de cada sección censal, mientras que el kernel fijo utiliza una distancia de corte.

La función de reducción de la distancia determina la ponderación de las observaciones con respecto a cada dato. Dado que el tamaño de las secciones censales varía, es preferible utilizar un núcleo adaptativo, ya que garantiza la coherencia de los barrios de toda la región Spatial Machine Learning - J. Morgan.

```
library(GWmodel)
library(sf)
library(tidyverse)

# convert to sp object
nyc_census_data_prepped_sp <- nyc_census_data_prepped %>%
  as_Spatial()

# choose bandwidth
bw <- bw.gwr(
  formula = formula,
  data = nyc_census_data_prepped_sp,
  kernel = "bisquare",
  adaptive = TRUE
)

# run model
gw_model <- gwr.basic(
  formula = formula,
  data = nyc_census_data_prepped_sp,
  bw = bw,
  kernel = "bisquare",
  adaptive = TRUE
)

# save results
gw_model_results <- gw_model$SDF %>%
  st_as_sf()
```

Para ver el R^2 del modelo, escribimos

```
# global R2 of gwr.basic from GWModel
gw_model$GW.diagnostic$gw.R2
```

```
## [1] 0.7214728
```

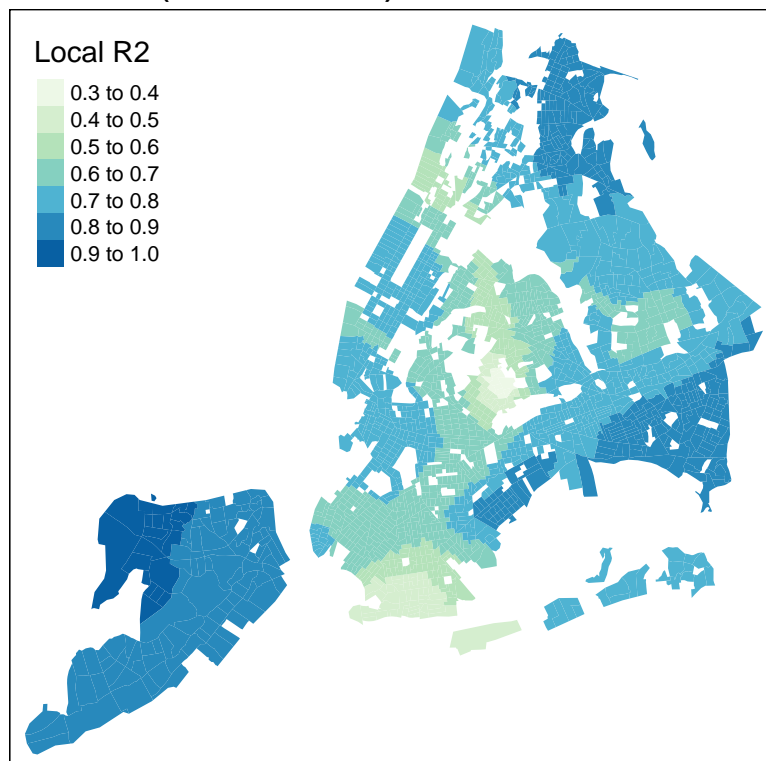
Podemos observar una medida local del R^2 para visualizar en que lugares el modelo posee un mejor desempeño.

Primero tenemos que añadir los datos R^2 locales al objeto sp. Después de esto podemos utilizar el paquete tmap para crear un mapa para el modelo GWRmodel de salida.

```
# gwmodel add to sp object
nyc_census_data_prepped_sp$gwmodel <- gw_model$SDF$Local_R2

# gwmodels
nyc_gw_model <- tm_shape(nyc_census_data_prepped_sp) +
  tm_fill("gwmodel",
    palette = "GnBu",
    title = "Local R2") +
  tm_layout(main.title = "GWR (GWModel) Local R2")
nyc_gw_model
```

GWR (GWModel) Local R2



GWR spgwr Utilizaremos la función `spgwr gwr()` para comparar una GWR con la versión del GWmodel. Aquí, si pasamos a la función un objeto `sp`, no necesitamos pasarle coordenadas.

```
library(spgwr)

# choose bandwidth
bw2 <- gwr.sel(formula = formula,
               data = nyc_census_data_prepped_sp,
               adapt = TRUE,
               gweight = gwr.bisquare,
               method = "cv",
               verbose = TRUE)

# run model
spgwr_model <- gwr(formula = formula,
                  data = nyc_census_data_prepped_sp,
                  adapt = bw2,
                  gweight = gwr.bisquare,
                  hatmatrix = TRUE)
```

En este caso, el modelo arroja un R^2 “cuasiglobal”, que debemos calcular. Basta con restar 1 a la división de las sumas de cuadrados residuales (RSS) entre las sumas de cuadrados totales (TSS):

```
# calculate quasi-global R2
spgwr_model$LocalR2 <- (1 - (spgwr_model$results$rss/spgwr_model$gTSS))
```

Hacemos lo mismo que antes con la visualización GWModel

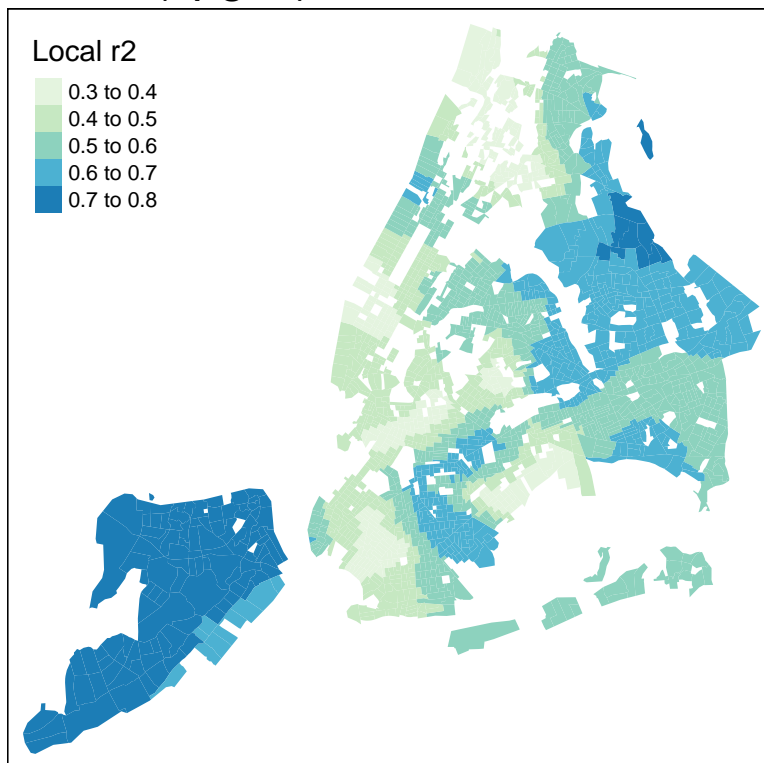
```
# load packages
library(tmap)

# spgwr add to sp object
nyc_census_data_prepped_sp$spgwr <- spgwr_model$SDF$localR2

# spgwr
nyc_spgwr <- tm_shape(nyc_census_data_prepped_sp) +
  tm_fill("spgwr",
    palette = "GnBu",
    title = "Local r2") +
  tm_layout(main.title = "GWR (spgwr) Local R2")

nyc_spgwr
```

GWR (spgwr) Local R2



Geographically Weighted Random Forest (GRF)

SpatialML Por último, utilizamos el paquete SpatialML para aplicar el modelo de bosque aleatorio (random forest) ponderado geográficamente grf() al conjunto de datos. Este modelo necesita que se le pase como argumento un objeto de coordenadas.

Este modelo es diferente del clásico random forest, ya que se construye un submodelo para cada ubicación de datos teniendo en cuenta sólo las observaciones cercanas. Estas observaciones cercanas se deciden por la selección del “bandwidth” óptimo. Sin embargo, este proceso consume mucho tiempo, por lo que se escoge aleatoriamente un número para el bandwidth (98).

Escribimos el código para implementar grf()

```
# load packages
library(SpatialML)
library(sf)
library(tidyverse)

# get centroids from each geom
nyc_census_data_prepped <- nyc_census_data_prepped %>%
  mutate(lon = map_dbl(geometry, ~st_centroid(.x)[[1]]),
         lat = map_dbl(geometry, ~st_centroid(.x)[[2]]))

# create coords column
coords <- nyc_census_data_prepped %>%
  st_drop_geometry() %>%
  select(lat, lon)

# prep for grf
nyc_census_grf_prepped <- nyc_census_data_prepped %>%
  st_drop_geometry() %>%
  mutate(sqrt_med_value = sqrt(median_value)) %>%
  select(!c(GEOID, NAM, residuals, lagged_residuals, lat, lon))

# define formula for grf, quitamos median_structure_age
formula_grf <- "sqrt_med_value ~ median_rooms + median_income +
pct_college + pct_foreign_born + pct_white + pct_black +
pct_hispanic + pct_asian + median_age + percent_ooh +
pop_density"

# get optimal bandwidth
bwgrf <- grf.bw(formula = formula_grf,
               dataset = nyc_census_grf_prepped,
               kernel = "adaptive",
               coords = coords,
               bw.min = 98,
               bw.max = 99,
               step = 1,
```

```

        trees = 500,
        mtry = NULL,
        importance = "impurity",
        forests = FALSE,
        weighted = TRUE,
        verbose = TRUE)

# model
grf_model <- grf(formula = formula_grf,
  dframe = nyc_census_grf_prepped,
  bw=bwgrf$Best.BW,
  ntree = 500,
  mtry = 2,
  kernel = "adaptive",
  forests = TRUE,
  coords = coords)

```

Podemos ver el R^2 global del modelo, el cual es peor que los modelos anteriores

```

# global R2 from grf
grf_model$Global.Model$r.squared

```

```
## [1] 0.54788
```

Visualizamos el desempeño con el R2 local

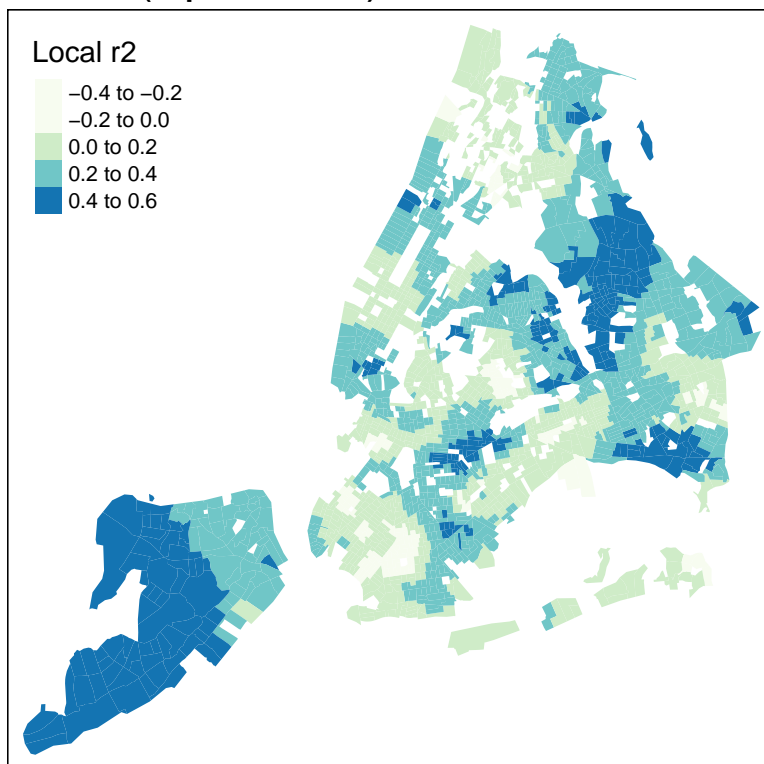
```
# grf

# add grf to sp object
nyc_census_data_prepped_sp$grf <- grf_model$LGofFit$LM_Rsq100 #local r2 grf

nyc_grf <- tm_shape(nyc_census_data_prepped_sp) +
  tm_fill("grf",
    palette = "GnBu",
    title = "Local r2") +
  tm_layout(main.title = "GRF (SpatialML) Local R2")

nyc_grf
```

GRF (SpatialML) Local R2

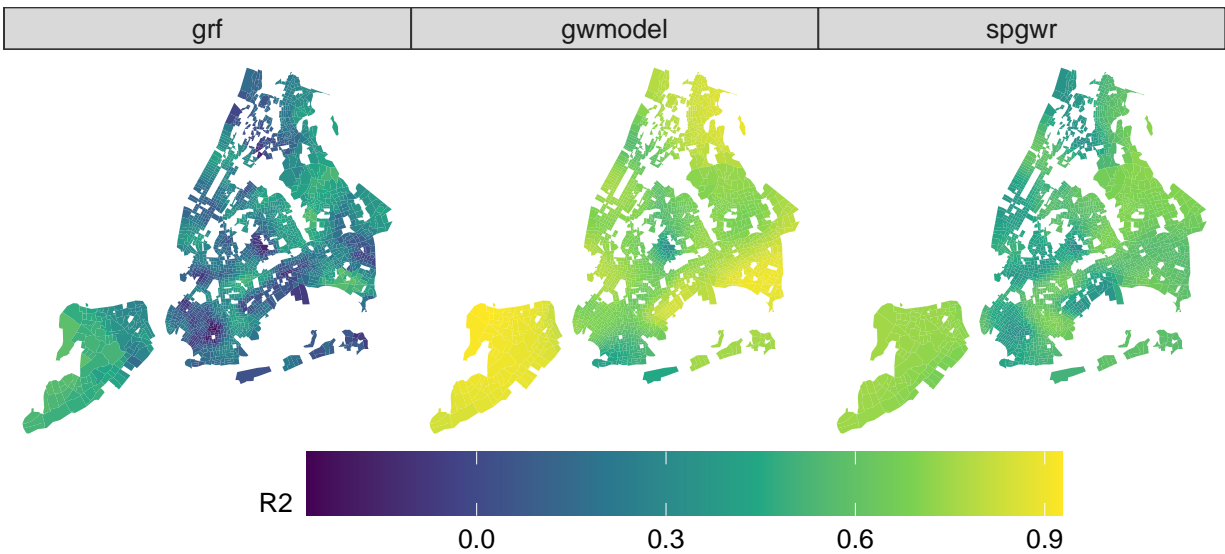


Comparación de R2 Local en modelos GWR y GRF

Usamos ggplot para visualizar los 3 modelos sobre el mismo mapa y bajo la misma escala de colores

```
library(ggthemes)
# pivot then plot
(ggplot_comp_loc_r2 <- nyc_census_data_prepped_sp %>%
  st_as_sf() %>%
  pivot_longer(cols = c("spgwr", "gwmodel", "grf")) %>%
  ggplot(aes(fill = value)) +
  geom_sf(color = NA) +
  scale_fill_viridis_c() +
  ggthemes::theme_map(base_size = 8) +
  facet_wrap(~name) +
  labs(title = "Local R2 of Geographically Weighted Models",
       fill = "R2") +
  theme(plot.title.position = 'plot',
        plot.title = element_text(hjust = 0.5,
                                   vjust = 3,
                                   size = 20),
        legend.position = c(0.20,-0.28),
        legend.key.height = unit(0.85, 'cm'),
        legend.key.width = unit(2, "cm"),
        legend.direction = "horizontal",
        legend.text = element_text(size=10),
        legend.title = element_text(size = 10),
        strip.text = element_text(size = 10)))
```

Local R2 of Geographically Weighted Models



```
png('image/localR2GWM.png')  
print(ggplot_comp_loc_r2)  
dev.off()
```

```
## pdf  
## 2
```

Podemos ver que la versión GWR del paquete GWmodel tiene, con diferencia, las puntuaciones R2 locales más altas las alcanza el paquete GWmodel seguido por el paquete spgwr (GWR) y finalmente SpatialML (GRF)

Conclusiones

Respecto de las aplicaciones de los distintos modelos para la predicción espacial, los modelos basados en SAR tienen mejores puntuaciones R^2 que el modelo de regresión lineal simple. En este caso, los erroresarlm() obtuvieron la puntuación más alta. Los modelos basados en GWR, que ajustan un modelo lineal a cada ubicación de datos, tienen valores R^2 globales más altos que los modelos basados en SAR. El único modelo basado en GRF (SpatialML) obtuvo una puntuación menor que los modelos basados en GWR, pero estuvo a la par con los modelos basados en SAR.

Se acompañan el experimento con documentación relacionada a la predicción usando data espacial, y los problemas que conlleva y las herramientas matemáticas ocupadas en el modelo.

Referencias

- [1] Chen, Y. 2022. Deriving two sets of bounds of moran's index by conditional extremum method. (2022). DOI:<https://doi.org/10.48550/ARXIV.2209.08562>.
- [2] Chen, Y. 2013. New Approaches for Calculating Moran's Index of Spatial Autocorrelation. *PLoS ONE*. 8, 7 (Jul. 2013), e68336. DOI:<https://doi.org/10.1371/journal.pone.0068336>.
- [3] Ikramov, K.D. 1994. A simple proof of the generalized Schur inequality. *Linear Algebra and its Applications*. 199, (Mar. 1994), 143–149. DOI:[https://doi.org/10.1016/0024-3795\(94\)90346-8](https://doi.org/10.1016/0024-3795(94)90346-8).
- [4] Jiang, Z. 2019. A survey on spatial prediction methods. *IEEE Transactions on Knowledge and Data Engineering*. 31, 9 (2019), 1645–1664. DOI:<https://doi.org/10.1109/TKDE.2018.2866809>.
- [5] Nikparvar, B. and Thill, J.-C. 2021. Machine Learning of Spatial Data. *ISPRS International Journal of Geo-Information*. 10, 9 (Sep. 2021), 600. DOI:<https://doi.org/10.3390/ijgi10090600>.
- [6] Tobler, W.R. 1970. A computer movie simulating urban growth in the detroit region. *Economic Geography*. 46, (Jun. 1970), 234. DOI:<https://doi.org/10.2307/143141>.
- [7] Walker, K. 2023. Analyzing US census data. (Jan. 2023). DOI:<https://doi.org/10.1201/9780203711415>.