Agrupamiento, agrupamiento espacial y geodemografía (geodemographics)

Muchos tipos de análisis de datos espaciales son complejos e involucran varias dimensiones que son difíciles de resumir en una sola variable. El Clustering (agrupamiento) aborda estas preguntas reduciendo la dimensionalidad (el número de variables relevantes) convirtiéndolas en un conjunto de clases que se puede manejar de manera más intuitiva. Por esta razón, el método es utilizado en varios contextos como la política pública y la mercadotecnia. Debido a que estos métodos no requieren de suposiciones sobre la estructura de los datos, suelen ser utilizados como una herramienta exploratoria que puede dar pistas sobre la forma y el contenido de un conjunto de datos.

La idea básica del agrupamiento estadístico es la de resumir la información contenida en muchas variables y crear un conjunto reducido de categorías. Cada observación es asignada a una sola categoría. Cuando el clustering es realizado en observaciones que representan áreas, esta técnica sule ser llamada análisis geodemográfico.

Aunque exisen muchas técnicas para agrupar las observaciones de un conjunto de datos estadísticamente, todas ellas se basan en la premisa de utilizar un conjunto de atributos para definir clases o categorías de observaciones que son similares entre sí, pero difieren entre los grupos. La manera en que se definen estas similitudes o discimilitudes depende de las técnicas particulares. En la práctica utilizaremos el método K-Means, que es probablemente el más utilizado con estos fines y el método de agrupamiento jerárquico.

En el caso del análisis de datos espaciales, existe un subconjunto de métodos que son de particular interés en muchos casos comunes en las Ciencias de Datos Espaciales; estos métodos reciben el nombre de técnicas de regionalización. Las técnicas de regionalización involucran el agrupamiento estadístico de las observaciones con la limitante de que las observaciones requieren ser vecinos geográficaos para estar en la misma categoría. Debido a esto, más que categoría, utilizamos el término área para cada observación y región para cada categoría, de manera que la regionalización es la construcción de regiones constituidas por áreas pequeñas.

Instalación de librería en el entorno virtual:

conda install -c anaconda scikit-learn

```
%matplotlib inline

import seaborn as sns
import pandas as pd
import pysal as ps
import geopandas as gpd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import cluster
from pysal.lib import weights as ct
import pysal.lib.weights.set_operations as Wsets
```

```
C:\Users\ferna\.conda\envs\esda\lib\site-packages\pysal\explore\segregation\network\network.py:16: UserWarnin
You can install them with  `pip install urbanaccess pandana` or `conda install -c udst pandana urbanaccess`
  "You need pandana and urbanaccess to work with segregation's network module\n"
C:\Users\ferna\.conda\envs\esda\lib\site-packages\pysal\model\spvcm\abstracts.py:10: UserWarning: The `dill`
  from .sqlite import head_to_sql, start_sql
```

Datos

Los datos.

```
#Carga los datos
zmvm = gpd.read_file('zmvm_salud.gpkg')
```

```
zmvm
```

	OBJECTID	CVEGEO	NOM_ENT	NOMBRE	POB15_ANA	POB15_SPRI	OCUP_S_AGU	VIV_S_WC	VIV_S_DREN	VIV_HAC
0	2	09003	Distrito Federal	Coyoactn	1.843174	25.059767	1.145264	0.434767	0.434767	16.875285
1	3	09004	Distrito Federal	Cuajimalpa de Morelos	1.750268	20.305602	0.984172	0.645789	0.645789	13.313495
2	4	09005	Distrito Federal	Gustavo A. Madero	2.523498	26.444958	3.866662	0.702011	0.702011	22.686162
3	5	09006	Distrito Federal	Iztacalco	2.427785	27.955145	1.097958	0.368602	0.368602	21.298149
4	6	09007	Distrito Federal	Iztapalapa	2.010253	26.734174	0.704954	0.365089	0.365089	18.662848
...	...	...	...	...	...	...	...	...	...	..
71	73	15120	M5xico	Zumpango	13.158091	48.685680	10.542988	28.485085	28.485085	41.294599
72	74	15121	M5xico	Cuautitl	4.328795	33.274707	10.446731	1.915266	1.915266	35.550773
73	75	15122	M5xico	Valle de Chalco Solidaridad	1.702117	22.143360	2.941706	0.284294	0.284294	13.478453
74	76	15125	M5xico	Tonanitla	4.748747	38.156273	2.186875	0.263620	0.263620	43.874769
75	1	09002	Distrito Federal	Azcapotzalco	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

76 rows × 17 columns

```
# Fija el CRS manualmente.
zmvm.crs = {'init': u'epsg:27700'}
zmvm.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 76 entries, 0 to 75
Data columns (total 17 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   OBJECTID    76 non-null     int64
```

```
 1   CVEGEO      76 non-null     object
 2   NOM_ENT     76 non-null     object
 3   NOMBRE      76 non-null     object
 4   POB15_ANA   76 non-null     float64
 5   POB15_SPRI  76 non-null     float64
 6   OCUP_S_AGU  76 non-null     float64
 7   VIV_S_WC    76 non-null     float64
 8   VIV_S_DREN  76 non-null     float64
 9   VIV_HAC     76 non-null     float64
 10  VIV_S_REFR  76 non-null     float64
 11  PEAOCUPADA  76 non-null     float64
 12  POB_S_DERE  76 non-null     float64
 13  DISP_INFRA  76 non-null     float64
 14  DISP_RH     76 non-null     float64
 15  DISP_EQUIP  76 non-null     float64
 16  geometry    76 non-null     geometry
dtypes: float64(12), geometry(1), int64(1), object(3)
memory usage: 10.2+ KB
C:\Users\ferna\.conda\envs\esda\lib\site-packages\pyproj\crs\crs.py:53: FutureWarning: '+init=<authority>:<co
  return _prepare_from_string(" ".join(pjargs))
```

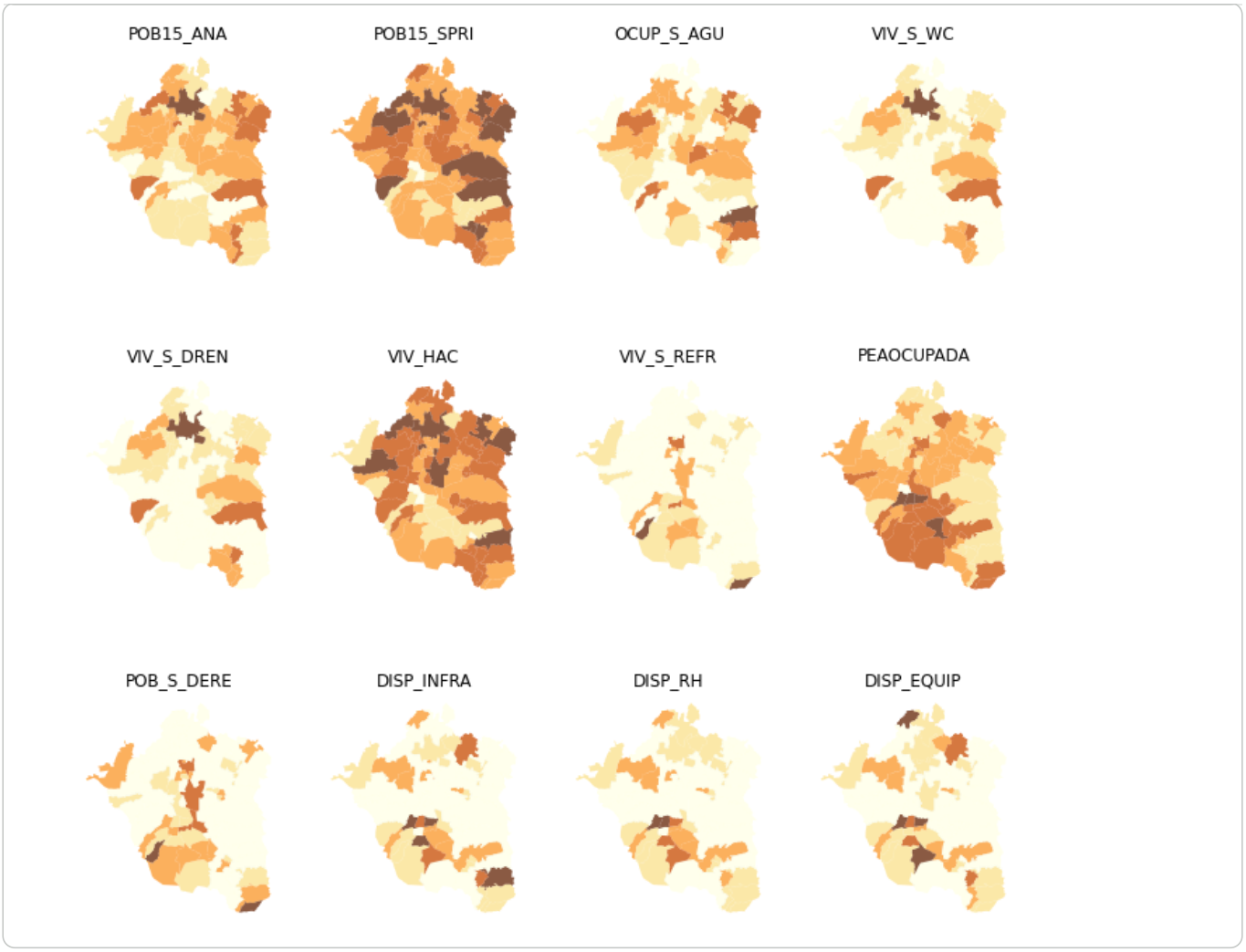
Antes de explorar los datos definimos las variables de interés para el agrupamiento.

```
variables = ['POB15_ANA', 'POB15_SPRI',
           'OCUP_S_AGU', 'VIV_S_WC',
           'VIV_S_DREN', 'VIV_HAC',
           'VIV_S_REFR', 'PEAOCUPADA', 'POB_S_DERE',
           'DISP_INFRA', 'DISP_RH',
           'DISP_EQUIP', ]
```

Conociendo los datos

La mejor forma de explorar la geografía de la accesibilidad a la salud es graficando cada variable en un mapa distinto. Esto nos dará una perspectiva de cada una de las vaiables de interés. Genramos un loop que genera cada mapa y lo pone en una subgráfica de la figura principal.

```
# Crea figura y ejes (en este caso es un arreglo de 3 x 4)
f, axs = plt.subplots(nrows=3, ncols=4, figsize=(12, 12))
# Hace los ejes accesibles con una sola indexación
axs = axs.flatten()
# Comienza el loop sobre las variables de interés
for i, col in enumerate(variables):
    # Selecciona el eje en el que el irá el mapa.
    ax = axs[i]
    # Grafica el mapa
    zmvm.plot(column=col, ax=ax, scheme='naturalbreaks', \
              linewidth=0, cmap='YlOrBr', alpha=0.75)
    # Remueve "basura" del eje
    ax.set_axis_off()
    # Le pone el nombre de la variable a cada eje
    ax.set_title(col)
# Despliega la figura
plt.show()
```



Antes de profundizar en la interpretación sustantiva del mapa, avancemos por el proceso de creación de la figura anterior, que involucra varias subgráficas dentro de la misma figura:

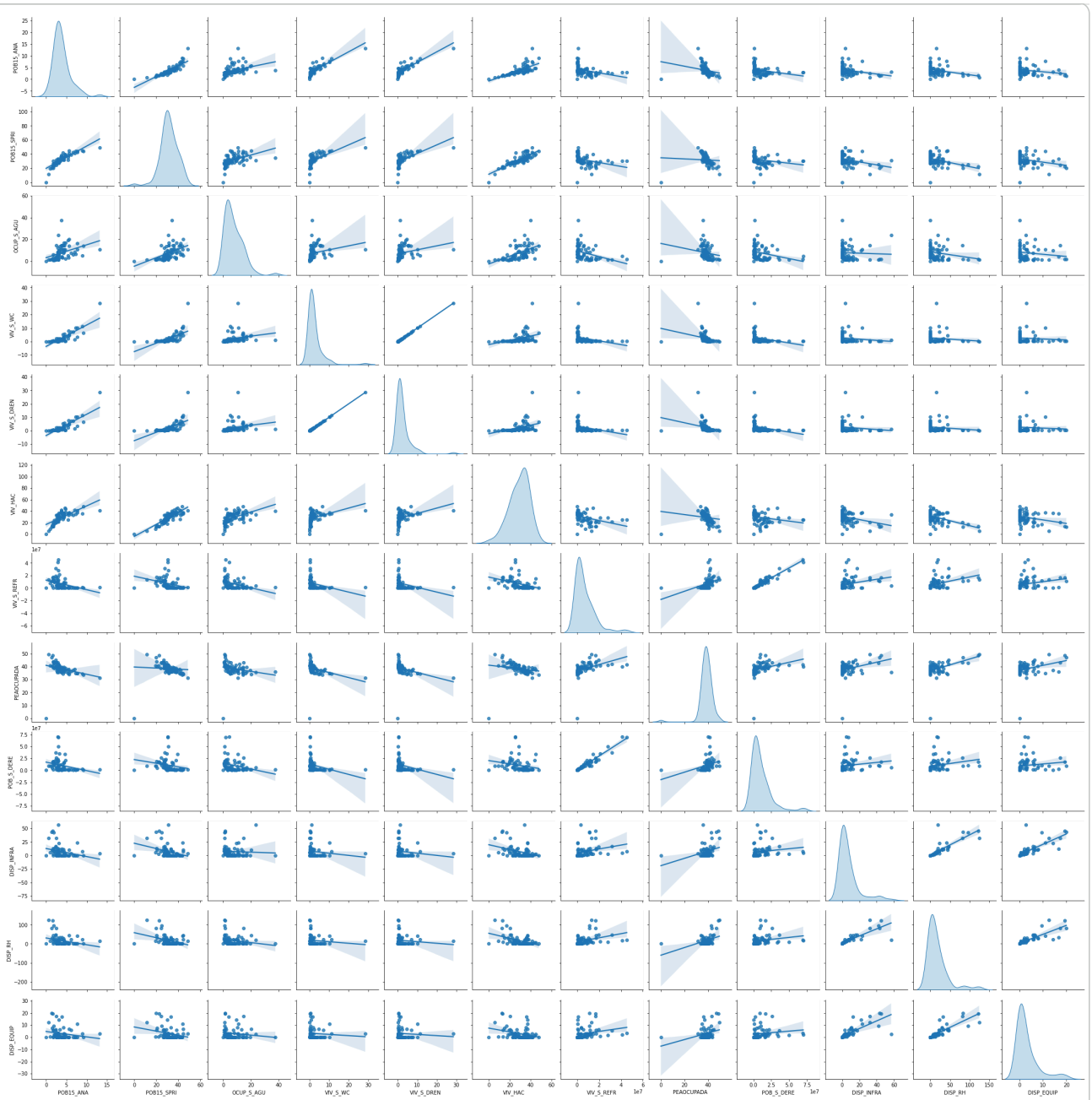
- Primero establecemos el número de filas y columnas que queremos para la cuadrícula de subplots. (L2)
- Luego desempaquetamos la cuadrícula en una lista plana (matriz) para los ejes de cada subtrama que podemos recorrer (L. 4).
- En este punto, configuramos un bucle for (L. 6) para trazar un mapa en cada una de las subgráficas.
- Dentro del ciclo (L. 6-14), extraemos el eje (L. 8), trazamos la coropleta (L. 10) y diseñamos el mapa (L. 11-14).
- Visualizar la figura (L. 16).

Como podemos ver, hay una variación sustancial en la forma en que las variables se distribuyen en el espacio.

Correlación entre variables. MATRIZ DE CORRELACIÓN

Se cuenta con 12 variables, pero "superponerlas" para obtener una evaluación general de la naturaleza de cada parte de la Zona Metropolitana. Para las correlaciones bivariadas, una herramienta útil es la gráfica de la matriz de correlación, disponible en seaborn:

```
_ = sns.pairplot(zmvm[variables], kind='reg', diag_kind='kde')
```

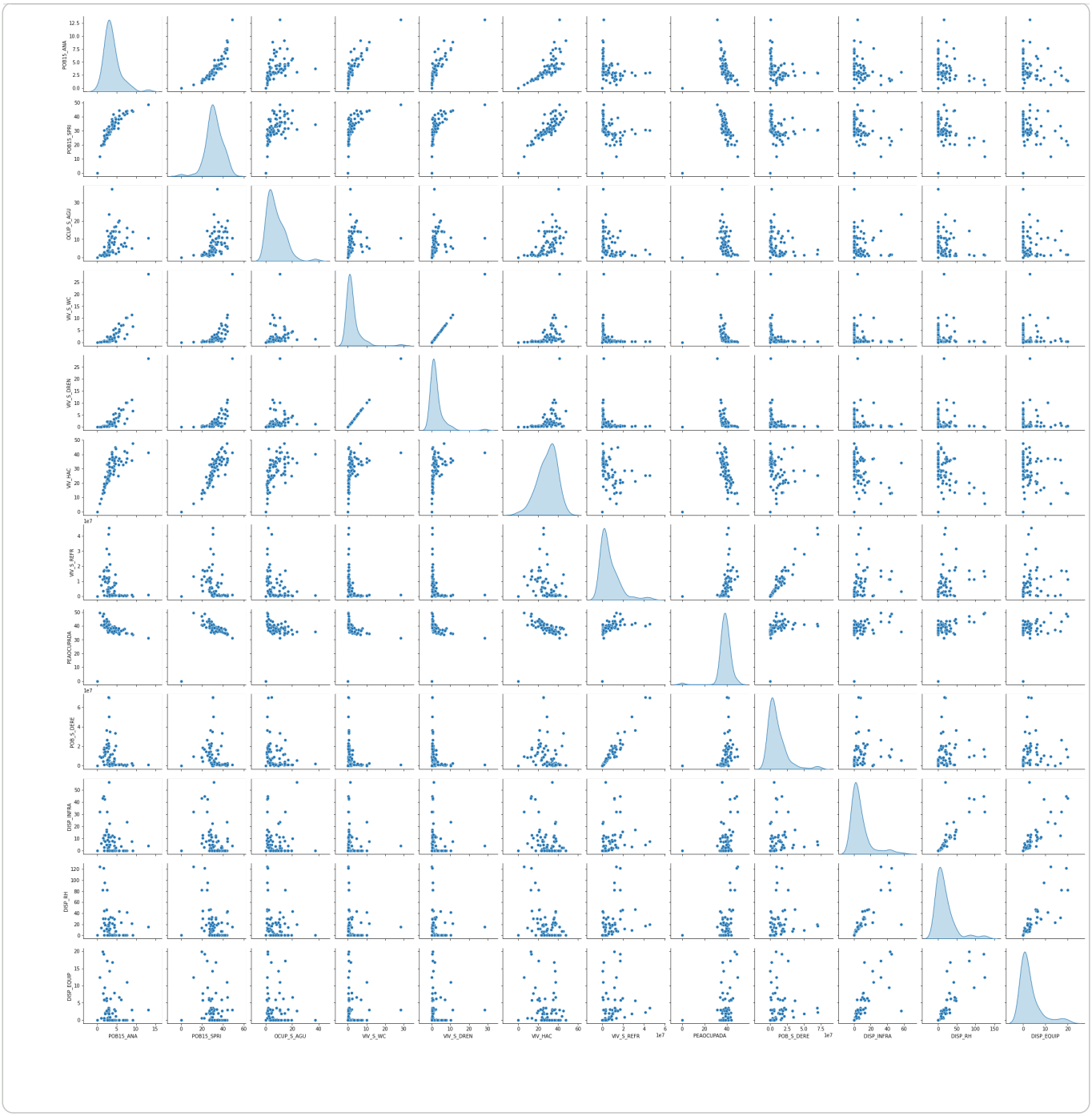



```
La mayor correlación positiva se da entre la población que no cuenta con drenaje y las que no disponen de WC.
```

```
  File "/tmp/ipython-input-2017779553.py", line 1
    La mayor correlación positiva se da entre la población que no cuenta con drenaje y las que no disponen
de WC.
       ^
SyntaxError: invalid syntax
```

Pasos siguientes: Explicar error

```
_ = sns.pairplot(zmvm[variables], kind='scatter', diag_kind='kde')
```



Una clasificación geodemográfica de la accesibilida de salud utilizando K-means

Un análisis geodemográfico implica la clasificación de las áreas que conforman un mapa greográfico en grupos o categorías de observaciones que son similares entre sí pero diferentes entre las categorías. La clasificación se lleva a cabo utilizando un algoritmo de agrupamiento estadístico que toma como entrada un conjunto de atributos y devuelve el grupo ("etiquetas" en la terminología) a la que pertenece cada observación. Para nuestra clasificación geodemográfica de las clasificaciones de la accesibilidad a la salud, usaremos uno de los algoritmos de agrupamiento más populares: K-means. Esta técnica solo requiere como entrada los atributos de observación y el número final de grupos en los que queremos agrupar las observaciones. En nuestro caso, usaremos cinco para comenzar, ya que esto nos permitirá ver más de cerca cada uno de ellos.

Aunque el algoritmo subyacente no es trivial, ejecutar K-means en Python se simplifica gracias a scikit-learn. Primero, debemos especificar los parámetros en el método KMeans (que es parte del submódulo de clúster de scikit-learn):

```
kmeans5 = cluster.KMeans(n_clusters=5)
```

Esto configura un objeto que contiene todos los parámetros necesarios para ejecutar el algoritmo. En nuestro caso, solo pasamos el número de clústeres, pero hay varios otros establecidos de forma predeterminada:

```
kmeans5
```

```
KMeans(n_clusters=5)
```

Para ejecutar realmente el algoritmo en los atributos, necesitamos llamar al método de ajuste:

```
# Esta línea se utiliza para obtener el mismo resultado siempre que se replique el análisis
np.random.seed(1234)
# Ejecuta el algoritmo de agrupamiento
k5cls = kmeans5.fit(zmvm[variables])
```

El objeto k5cls que acabamos de crear contiene varios componentes que pueden ser útiles para un análisis. Por ahora, usaremos las etiquetas, que representan las diferentes categorías en las que hemos agrupado los datos. Recuerde, en Python, la vida comienza en cero, por lo que las etiquetas de grupo van de cero a cuatro. Las etiquetas se pueden extraer de la siguiente manera:

```
k5cls.labels_
```

```
array([2, 4, 0, 1, 2, 3, 2, 0, 4, 2, 4, 4, 2, 4, 2, 2, 0, 0, 0, 0, 0, 4,
       0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 1, 3, 0, 0, 0, 2, 0, 4, 0, 0, 0,
       0, 1, 1, 0, 2, 0, 0, 0, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2,
       0, 0, 4, 0, 4, 0, 0, 4, 4, 0])
```

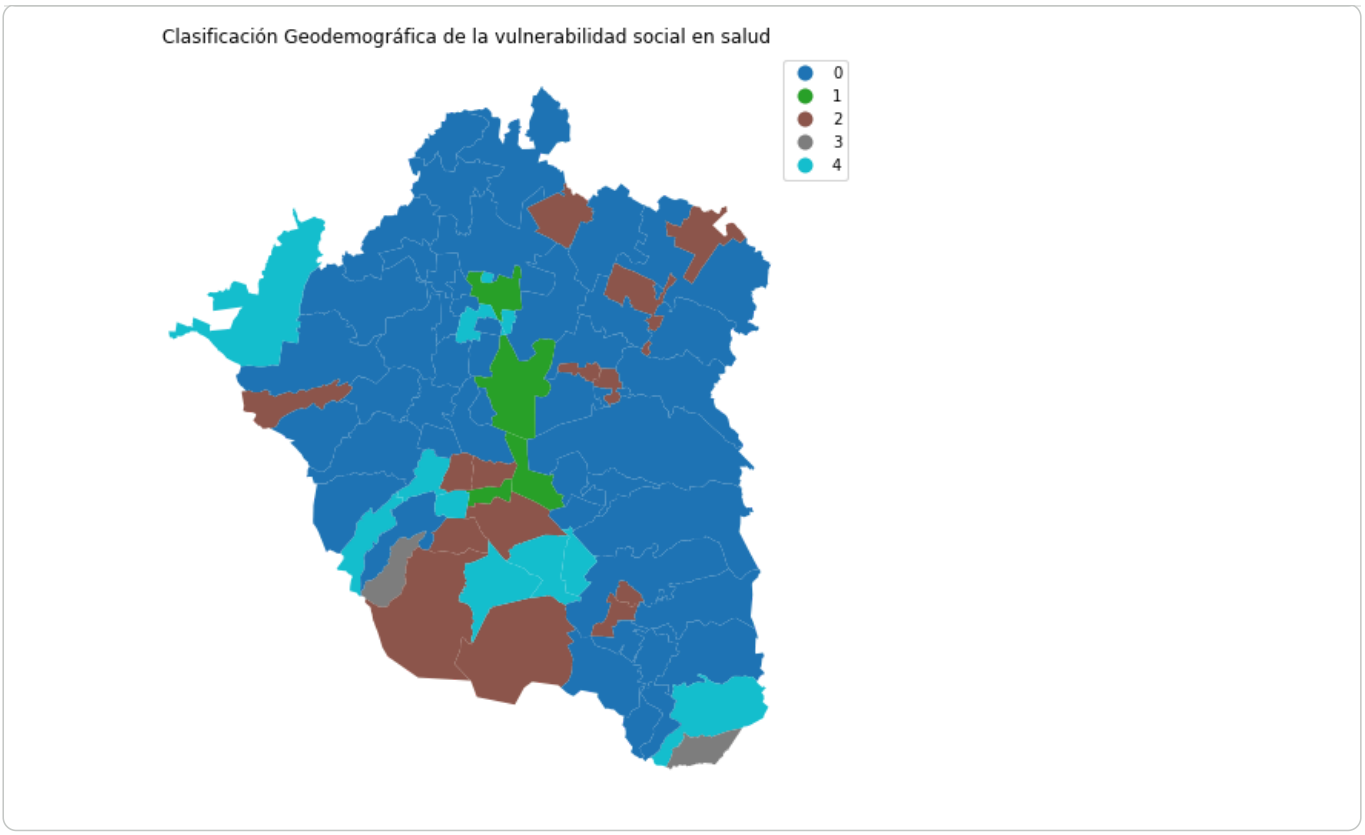
Cada número representa una categoría diferente, por lo que dos observaciones con el mismo número pertenecen al mismo grupo. Las etiquetas se devuelven en el mismo orden en que se pasaron los atributos de entrada, lo que significa que podemos agregarlas a la tabla de datos original como una columna adicional:

```
zmvm['k5cls'] = k5cls.labels_
```

Mapeando las categorias

Para comprender mejor la clasificación que acabamos de realizar, es útil mostrar las categorías creadas en un mapa. Para esto, usaremos un valor único de coropletas, que asignará automáticamente un color diferente a cada categoría:

```
# Ajusta figura y ax
f, ax = plt.subplots(1, figsize=(9, 9))
# Graficar valores únicos coropletas incluyendo una leyenda y sin líneas de límite
zmvm.plot(column='k5cls', categorical=True, legend=True, linewidth=0, ax=ax)
# Remueve ejes
ax.set_axis_off()
# Mentiene los ejes proporcionados
plt.axis('equal')
# Agrega título
plt.title('Clasificación Geodemográfica de la vulnerabilidad social en salud ')
# Despliega el mapa
plt.show()
```



Se observa que no están realizados por contigüidad y tienen distintos números de polígonos por cluster. Hay algunos con pocos polígonos. Los coloca en otra categoría.

El mapa de arriba representa la distribución geográfica de las cinco categorías creadas por el algoritmo K-means. Una mirada rápida muestra una estructura espacial fuerte en la distribución de los colores: hay grupos que se encuentra principalmente en el centro de la ciudad y apenas en la periferia, mientras que otros se comportan de manera opuesta. También hay diferencias en el número de elementos que se encuentran en cada categoría.

Explorando la naturaleza de las categorías.

Una vez que tenemos una idea de dónde y cómo se distribuyen las categorías en el espacio, también es útil explorarlas estadísticamente. Esto nos permitirá caracterizarlas, dándonos una idea del tipo de observaciones contenidas en cada uno de ellos. Como primer paso, veamos cuántas observaciones hay en cada categoría. Para hacer eso, haremos uso del operador groupby introducido anteriormente, combinado con el tamaño de la función, que devuelve el número de elementos en un subgrupo:

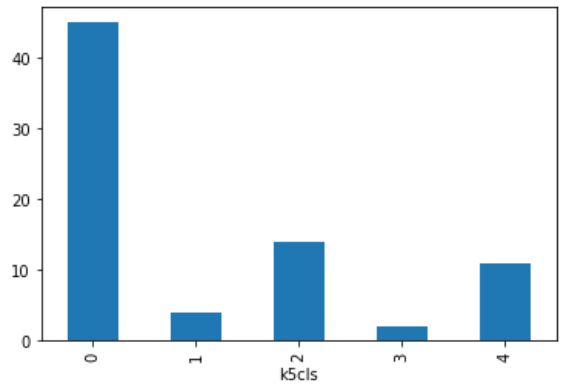
```
k5sizes = zmvm.groupby('k5cls').size()
k5sizes
```

```
k5cls
0    45
1     4
2    14
3     2
4    11
dtype: int64
```

El operador groupby agrupa una tabla (DataFrame) usando los valores en la columna provista (k5cls) y los pasa a la función proporcionada de antemano, que en este caso es el tamaño. Efectivamente, lo que esto hace es agrupar por las

observaciones por las categorías creadas y contar cuántas de ellas contiene cada una. Para una representación más visual de la salida, una gráfica de barras es una buena alternativa:

```
#Es lo mismo que el anterior pero en gráfica de barras.
_ = k5sizes.plot.bar()
```



Como sospechamos en el mapa, los grupos varían en tamaño, con los grupos cero, tres y cuatro más de 60 observaciones cada uno, y uno y dos menores de veinte.

Para describir la naturaleza de cada categoría, podemos observar los valores de cada uno de los atributos que hemos utilizado para crearlos en primer lugar. Recuerde que utilizamos las calificaciones promedio en muchos aspectos (limpieza, comunicación del anfitrión, etc.) para crear la clasificación, por lo que podemos comenzar por verificar el valor promedio de cada uno. Para hacer eso en Python, volveremos a usar el operador groupby pero, en este caso, lo combinaremos con la función media:

```
k5means = zmvm.groupby('k5cls')[variables].mean()
# Muestra la tabla transpuesta
k5means.T
```

k5cls	0	1	2	3	4
POB15_ANA	4.511728e+00	3.355853e+00	2.532027e+00	2.955491e+00	2.606059e+00
POB15_SPRI	3.426426e+01	3.196274e+01	2.646302e+01	3.037147e+01	2.670235e+01
OCUP_S_AGU	9.578362e+00	4.065304e+00	5.061432e+00	3.061712e+00	4.590154e+00
VIV_S_WC	3.621163e+00	3.673086e-01	7.939769e-01	3.186136e-01	4.090049e-01
VIV_S_DREN	3.621163e+00	3.673086e-01	7.939769e-01	3.186136e-01	4.090049e-01
VIV_HAC	3.268139e+01	3.100359e+01	2.220971e+01	2.530997e+01	2.200262e+01
VIV_S_REFR	1.243277e+06	2.388949e+07	8.878119e+06	4.330756e+07	1.425386e+07
PEAOCUPADA	3.616967e+01	4.047419e+01	4.160699e+01	4.069369e+01	4.207423e+01
POB_S_DERE	2.200779e+06	3.869794e+07	1.181957e+07	6.976631e+07	2.010701e+07
DISP_INFRA	3.754861e+00	6.884073e+00	1.343590e+01	6.105165e+00	1.321699e+01
DISP_RH	6.913427e+00	1.884440e+01	3.500606e+01	1.795460e+01	3.531518e+01
DISP_EQUIP	1.871002e+00	2.734020e+00	4.835315e+00	2.914036e+00	5.842745e+00

O podemos intentar obtener una descripción más completa e incluir también los cuartiles y la desviación estándar llamando a la función describe en lugar de simplemente la media:

```
# Calcular el summary
# Si deseas desplegar todas las columnas descomenta la siguiente línea
```

```
#pd.set_option("display.max_columns", None)
k5desc = zmvm.groupby('k5cls')[variables].describe()
# Muestra la tabla
k5desc
```

	POB15_ANA								POB15_SPRI		...	DISP_I
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%
k5cls												
0	45.0	4.511728	2.337345	0.000000	2.998296	3.949238	5.546521	13.158091	45.0	34.264262	...	9.597
1	4.0	3.355853	0.966959	2.427785	2.865223	3.145523	3.636154	4.704581	4.0	31.962742	...	20.880
2	14.0	2.532027	1.201799	0.648355	1.800793	2.330830	3.337916	4.433839	14.0	26.463022	...	32.456
3	2.0	2.955491	0.095309	2.888097	2.921794	2.955491	2.989188	3.022885	2.0	30.371472	...	18.623
4	11.0	2.606059	0.874979	1.607624	2.054014	2.448336	2.931080	4.748747	11.0	26.702353	...	45.040

5 rows × 96 columns

```
#Otra forma de representar los resultados de la tabla anterior
for clus in k5desc.T:
    print('\n\t---------\n\tCluster %i'%clus)
    print(k5desc.T[clus].unstack())
```

```
        ---------
        Cluster 0
             count          mean           std  min           25%  \
POB15_ANA     45.0  4.511728e+00  2.337345e+00  0.0  2.998296e+00
POB15_SPRI    45.0  3.426426e+01  7.941907e+00  0.0  3.061901e+01
OCUP_S_AGU    45.0  9.578362e+00  6.987574e+00  0.0  4.026351e+00
VIV_S_WC      45.0  3.621163e+00  4.757087e+00  0.0  1.201732e+00
VIV_S_DREN    45.0  3.621163e+00  4.757087e+00  0.0  1.201732e+00
VIV_HAC       45.0  3.268139e+01  7.967805e+00  0.0  3.090486e+01
VIV_S_REFR    45.0  1.243277e+06  1.220317e+06  0.0  4.576213e+05
PEAOCUPADA    45.0  3.616967e+01  5.966475e+00  0.0  3.543294e+01
POB_S_DERE    45.0  2.200779e+06  1.932253e+06  0.0  1.022367e+06
DISP_INFRA    45.0  3.754861e+00  9.766028e+00  0.0  0.000000e+00
DISP_RH       45.0  6.913427e+00  1.243535e+01  0.0  0.000000e+00
DISP_EQUIP    45.0  1.871002e+00  3.944411e+00  0.0  0.000000e+00

                    50%           75%           max
POB15_ANA    3.949238e+00  5.546521e+00  1.315809e+01
POB15_SPRI   3.329566e+01  3.866917e+01  4.868568e+01
OCUP_S_AGU   8.107531e+00  1.414791e+01  3.748773e+01
VIV_S_WC     1.876955e+00  4.647361e+00  2.848508e+01
VIV_S_DREN   1.876955e+00  4.647361e+00  2.848508e+01
VIV_HAC      3.396098e+01  3.683636e+01  4.776699e+01
VIV_S_REFR   7.800315e+05  1.247619e+06  4.643713e+06
PEAOCUPADA   3.676369e+01  3.854868e+01  4.372207e+01
POB_S_DERE   1.480116e+06  3.053715e+06  7.562911e+06
DISP_INFRA   0.000000e+00  2.462523e+00  5.653276e+01
DISP_RH      0.000000e+00  9.597793e+00  4.365144e+01
DISP_EQUIP   0.000000e+00  1.899563e+00  1.672828e+01

        ---------
        Cluster 1
             count          mean           std           min           25%  \
POB15_ANA      4.0  3.355853e+00  9.669589e-01  2.427785e+00  2.865223e+00
POB15_SPRI     4.0  3.196274e+01  5.033008e+00  2.795514e+01  2.924645e+01
OCUP_S_AGU     4.0  4.065304e+00  4.577457e+00  1.097958e+00  1.210594e+00
VIV_S_WC       4.0  3.673086e-01  1.045136e-01  2.235191e-01  3.323311e-01
VIV_S_DREN     4.0  3.673086e-01  1.045136e-01  2.235191e-01  3.323311e-01
VIV_HAC        4.0  3.100359e+01  1.004047e+01  2.129815e+01  2.690509e+01
VIV_S_REFR     4.0  2.388949e+07  7.498074e+06  1.464763e+07  1.962222e+07
PEAOCUPADA     4.0  4.047419e+01  1.857670e+00  3.776139e+01  4.016944e+01
```

```
POB_S_DERE    4.0  3.869794e+07  7.821671e+06  3.336192e+07  3.435738e+07
DISP_INFRA    4.0  6.884073e+00  6.938791e+00  2.441196e+00  2.865908e+00
DISP_RH       4.0  1.884440e+01  1.902175e+01  7.323587e+00  8.381616e+00
DISP_EQUIP    4.0  2.734020e+00  2.113837e+00  8.035703e-01  1.511036e+00

                     50%           75%           max
POB15_ANA   3.145523e+00  3.636154e+00  4.704581e+00
POB15_SPRI  3.030441e+01  3.302069e+01  3.928701e+01
OCUP_S_AGU  2.186227e+00  5.040937e+00  1.079080e+01
VIV_S_WC    3.880085e-01  4.229860e-01  4.696983e-01
VIV_S_DREN  3.880085e-01  4.229860e-01  4.696983e-01
VIV_HAC     2.880986e+01  3.290837e+01  4.509650e+01
VIV_S_REFR  2.466581e+07  2.893309e+07  3.157871e+07
PEAOCUPADA  4.108558e+01  4.139033e+01  4.196422e+01
POB_S_DERE  3.557776e+07  3.991831e+07  5.027431e+07
DISP_INFRA  3.962424e+00  7.980590e+00  1.717025e+01
```

Sin embargo, este enfoque se expande rápidamente y las tablas se vuelven muy grandes para comunicar fácilmente cualquier patrón. Una buena alternativa es visualizar la distribución de valores por categoría. La siguiente extensión opcional muestra cómo transformar los datos para que pueda crear un gráfico bastante sofisticado que resuma la tabla anterior.

Para hacer esto convenientemente, necesitamos "ordenar" la tabla de valores. La tabla que queremos trazar para reemplazar el resumen anterior contiene los siguientes datos:

```
# Nombrar (index) los renglones por la categoría a la que pertenecen
to_plot = zmvm.set_index('k5cls')
# Subconjutno que solo mantiene las variables utilizadas por el K-Means
to_plot = to_plot[variables]
# Muestra la tabla
to_plot.head()
```

k5cls	POB15_ANA	POB15_SPRI	OCUP_S_AGU	VIV_S_WC	VIV_S_DREN	VIV_HAC	VIV_S_REFR	PEAOCUPADA	POB_S_DE
2	1.843174	25.059767	1.145264	0.434767	0.434767	16.875285	1.140831e+07	42.750011	1.031191e+
4	1.750268	20.305602	0.984172	0.645789	0.645789	13.313495	1.737401e+07	43.831075	1.841841e+
0	2.523498	26.444958	3.866662	0.702011	0.702011	22.686162	4.643713e+06	43.722068	5.782912e+
1	2.427785	27.955145	1.097958	0.368602	0.368602	21.298149	3.157871e+07	41.964222	3.646631e+
2	2.010253	26.734174	0.704954	0.365089	0.365089	18.662848	1.016181e+07	43.352258	1.245141e+

Apilando los datos

```
to_plot = to_plot.stack()
to_plot.head()
```

```
k5cls
2      POB15_ANA      1.843174
       POB15_SPRI    25.059767
       OCUP_S_AGU     1.145264
       VIV_S_WC       0.434767
       VIV_S_DREN     0.434767
dtype: float64
```

Esto devuelve un objeto de indexación múltiple, que podemos convertirlo en un DataFrame al tratar el índice como columnas adicionales:

```
to_plot = to_plot.reset_index()
to_plot.head()
```

	k5cls	level_1	0
0	2	POB15_ANA	1.843174
1	2	POB15_SPRI	25.059767
2	2	OCUP_S_AGU	1.145264
3	2	VIV_S_WC	0.434767
4	2	VIV_S_DREN	0.434767

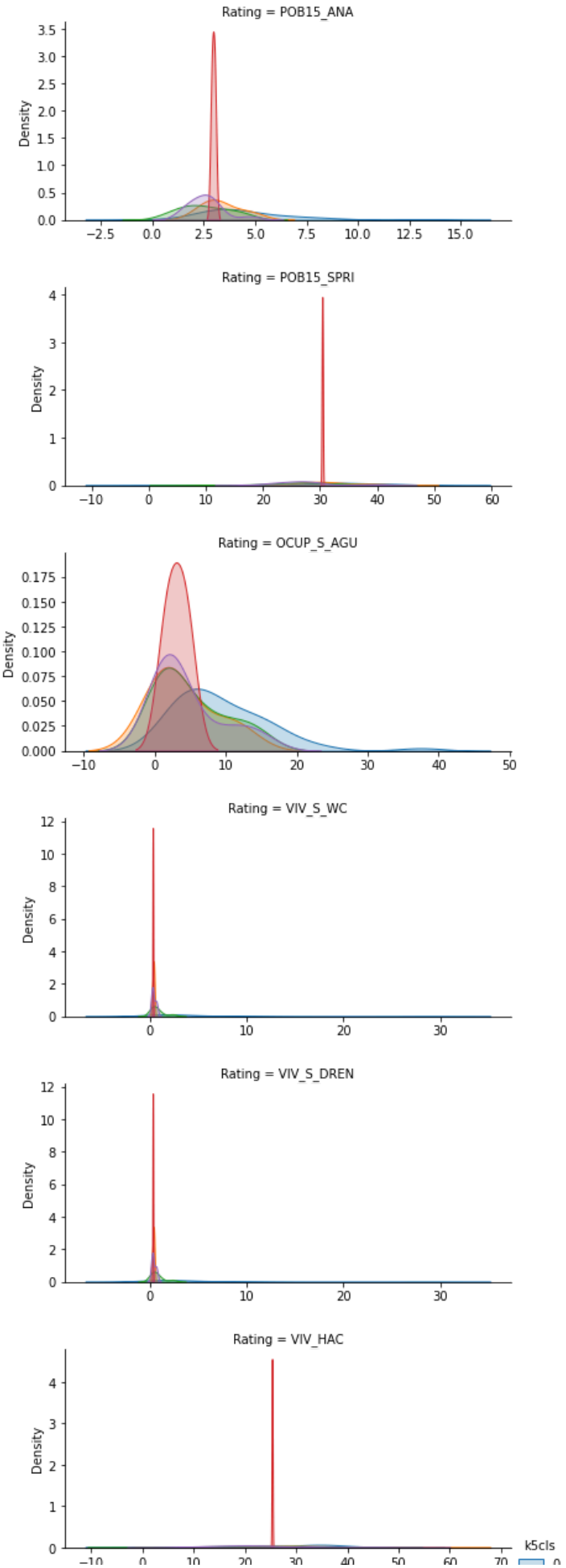
Le damos a las columnas nombres más significativos:

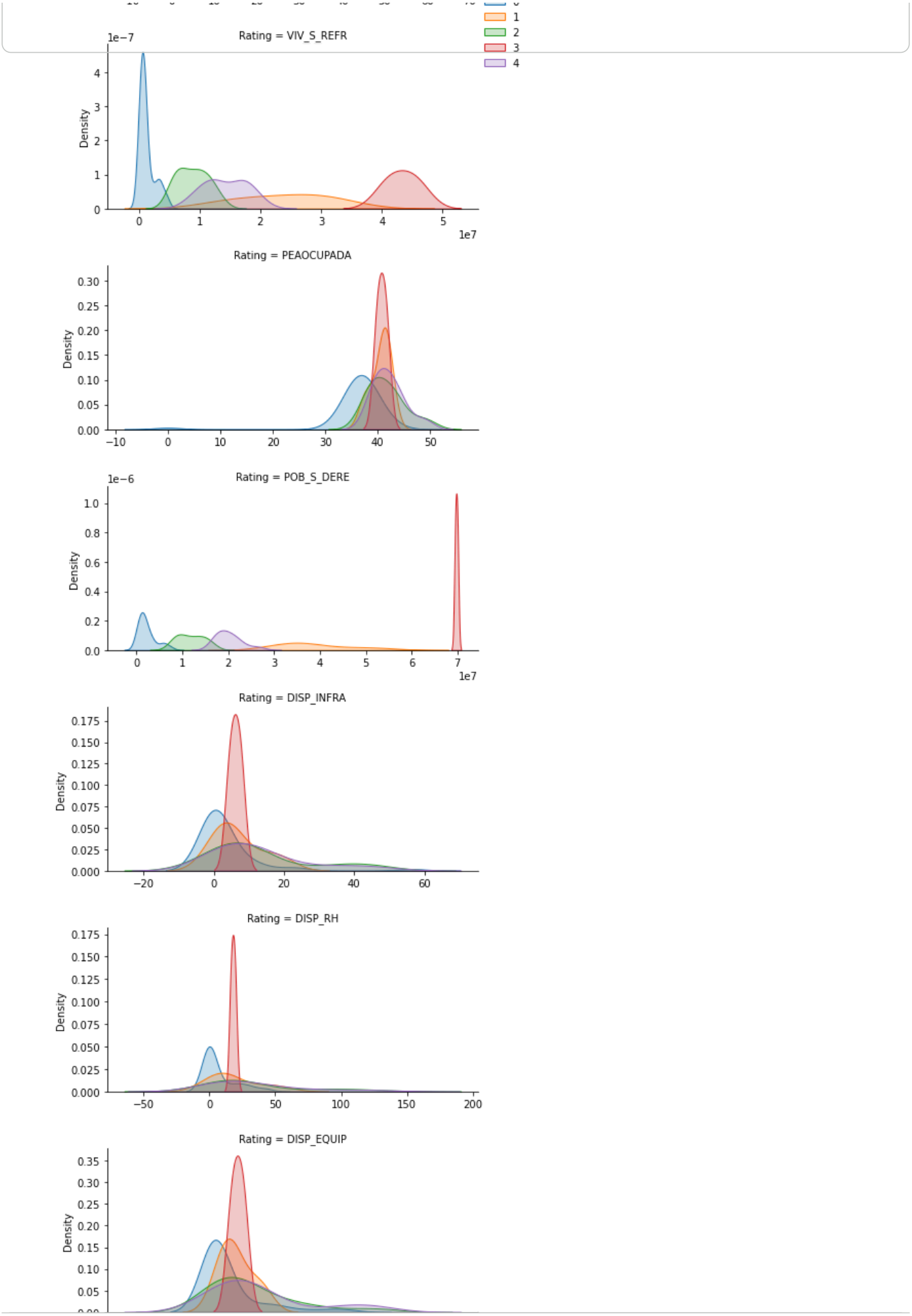
```
to_plot = to_plot.rename(columns={'level_1': 'variables', 0: 'Values'})
to_plot.head()
```

	k5cls	Rating	Values
0	2	POB15_ANA	1.843174
1	2	POB15_SPRI	25.059767
2	2	OCUP_S_AGU	1.145264
3	2	VIV_S_WC	0.434767
4	2	VIV_S_DREN	0.434767

En este punto, estamos listos para visualizar la distribución de valores por tipo de calificación por categoría. Esto se hace en dos pasos:
1.-Configuración del eje ("facet") para graficar por variable.
2.-Construcción de la gráfica

```
# Setup the facets
facets = sns.FacetGrid(data=to_plot, row='Rating', hue='k5cls', \
                  sharey=False, sharex=False, aspect=2)
# Build the plot as a `sns.kdeplot`
_ = facets.map(sns.kdeplot, 'Values', shade=True).add_legend()
```


Rating = POB15_ANA
Density
Rating = POB15_SPRI
Density
Rating = OCUP_S_AGU
Density
Rating = VIV_S_WC
Density
Rating = VIV_S_DREN
Density
Rating = VIV_HAC
Density
k5cls


1
2
3
4
Rating = VIV_S_REFR
1e−7
Density
1e7
Rating = PEAOCUPADA
Density
Rating = POB_S_DERE
1e−6
Density
1e7
Rating = DISP_INFRA
Density
Rating = DISP_RH
Density
Rating = DISP_EQUIP
Density

```
Empieza a programar o a crear código con IA.
```

Con esto concluye la sección de geodemografía. Como hemos visto, la esencia de este enfoque es agrupar áreas basándose únicamente en una base estadística: donde se ubica cada área es irrelevante para la etiqueta que recibe del algoritmo de agrupación. En muchos contextos, esto no solo es permisible sino incluso deseable, ya que el interés es ver si determinadas combinaciones de valores se distribuyen en el espacio de alguna manera discernible. Sin embargo, en otro contexto, es posible que nos interesen los grupos creados de observaciones que siguen ciertas restricciones espaciales. Para eso, ahora nos convertimos en técnicas de regionalización.

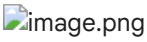
Agrupamiento Jerárquico (Hierarchical Clustering)

Como se mencionó previamente, el k-medias no es el único algritmo de agrupamiento. Hay muchos más. Ahora utilizaremos el mismo conjunto de datos utilizando un método aglomerativo jerárquico "agglomerative hierarchical clustering" (AHC). EL algoritmo aglomerativo trabaja construyendo una jerarquía de agrupamiento que comienza con cada observación perteneciendo a un cúmulo distinto y termina con todas las observaciones asignadas al mismo cúmulo. Estos extremos no son muy útiles por si mismos, pero, en el punto medio de la jerarquía podemos encontrar muchas alternaivas de agrupamiento con distintos niveles de detalle.

Algoritmo:

1. Comienza con cada elemento como parte de su propio cúmulo;
2. Encuentra las dos observaciones más cercanas basado en una métrica de distancia (por ejemplo, la distancia euclideana);
3. Une las observaciones en un nuevo cúmulo;
4. Repite los pasos 2) y 3) hasta alcanzar el grado de agregación deseado.

EL algoritmo se llama "aglomerativo" debido a que empieza con cúmulos individuales y los aglomera cada vez en menos y menos cúmulos que contienen más observaciones cada vez. El método AHC no requiere que el usuario especifique un número de cúmulos ya que el algoritmo puede producir muchos cúmulos (empezando con tantos cúmulos como observaciones) o un solo cúmulo (con todas las observaciones asignadas a él).



```
# Siembra semilla para que el resultado sea reproducible
np.random.seed(0)
# Inicia el algoritmo, la opción ward minimiza la varianza de los cúmulos que se fusionan
model = cluster.AgglomerativeClustering(linkage='ward', n_clusters=5)
# Ejecuta el agrupamiento
model.fit(zmvm[variables])
# Asigna etiquetas a la tabla
zmvm['clus_ahc'] =model.labels_
```

```
ahc_size = zmvm.groupby('clus_ahc').size()
ahc_size
```

```
clus_ahc
0    17
1     4
2     2
3    15
4    38
dtype: int64
```

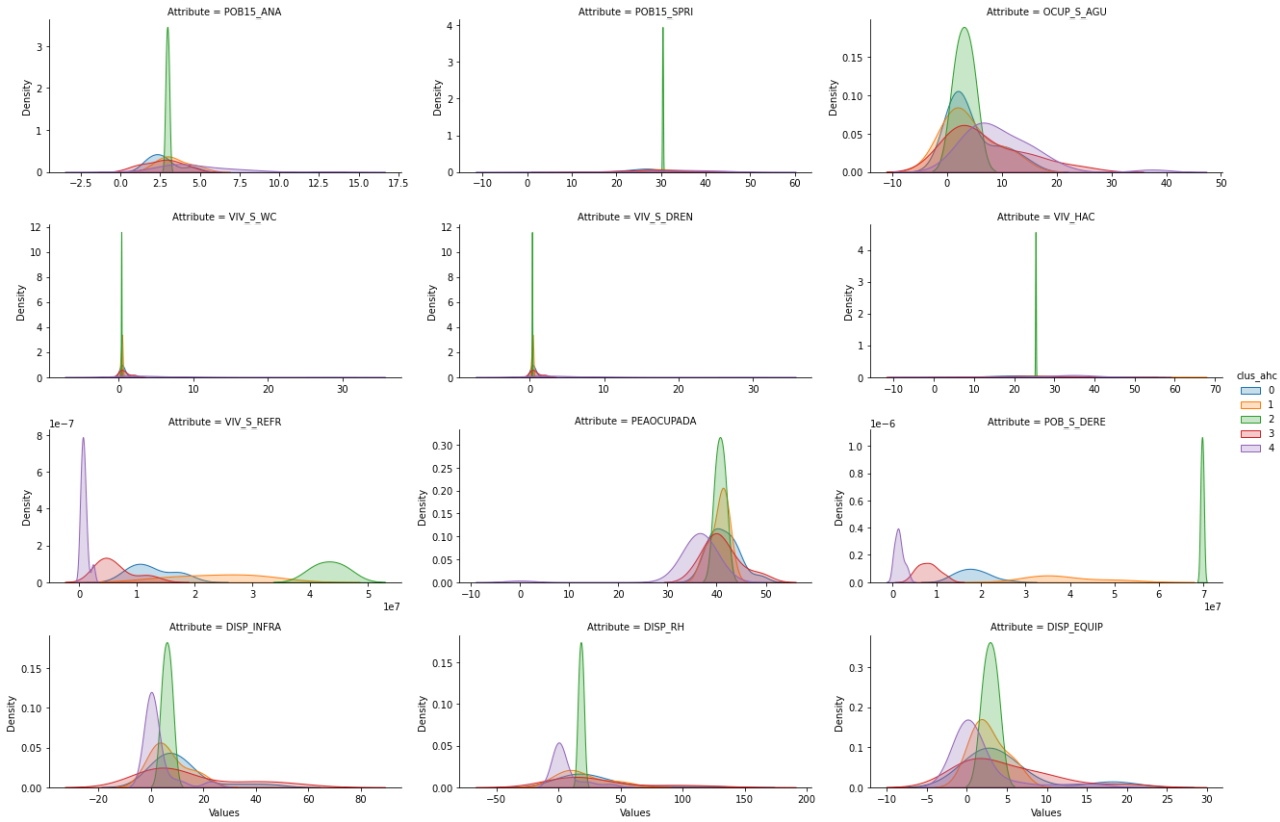
```
ahc_means = zmvm.groupby('clus_ahc')[variables].mean()
ahc_means.T
```

clus_ahc	0	1	2	3	4
POB15_ANA	2.700206e+00	3.355853e+00	2.955491e+00	2.623911e+00	4.786332e+00
POB15_SPRI	2.749690e+01	3.196274e+01	3.037147e+01	2.626775e+01	3.538517e+01
OCUP_S_AGU	4.532601e+00	4.065304e+00	3.061712e+00	7.159974e+00	9.682216e+00
VIV_S_WC	5.850118e-01	3.673086e-01	3.186136e-01	8.125786e-01	4.116662e+00
VIV_S_DREN	5.850118e-01	3.673086e-01	3.186136e-01	8.125786e-01	4.116662e+00
VIV_HAC	2.323211e+01	3.100359e+01	2.530997e+01	2.368940e+01	3.350896e+01
VIV_S_REFR	1.257884e+07	2.388949e+07	4.330756e+07	6.221673e+06	7.860062e+05
PEAOCUPADA	4.140091e+01	4.047419e+01	4.069369e+01	4.100058e+01	3.563489e+01
POB_S_DERE	1.815896e+07	3.869794e+07	6.976631e+07	8.018723e+06	1.492181e+06
DISP_INFRA	1.116085e+01	6.884073e+00	6.105165e+00	1.502817e+01	2.297402e+00
DISP_RH	2.931657e+01	1.884440e+01	1.795460e+01	3.156071e+01	5.733254e+00
DISP_EQUIP	4.733311e+00	2.734020e+00	2.914036e+00	4.868161e+00	1.649236e+00

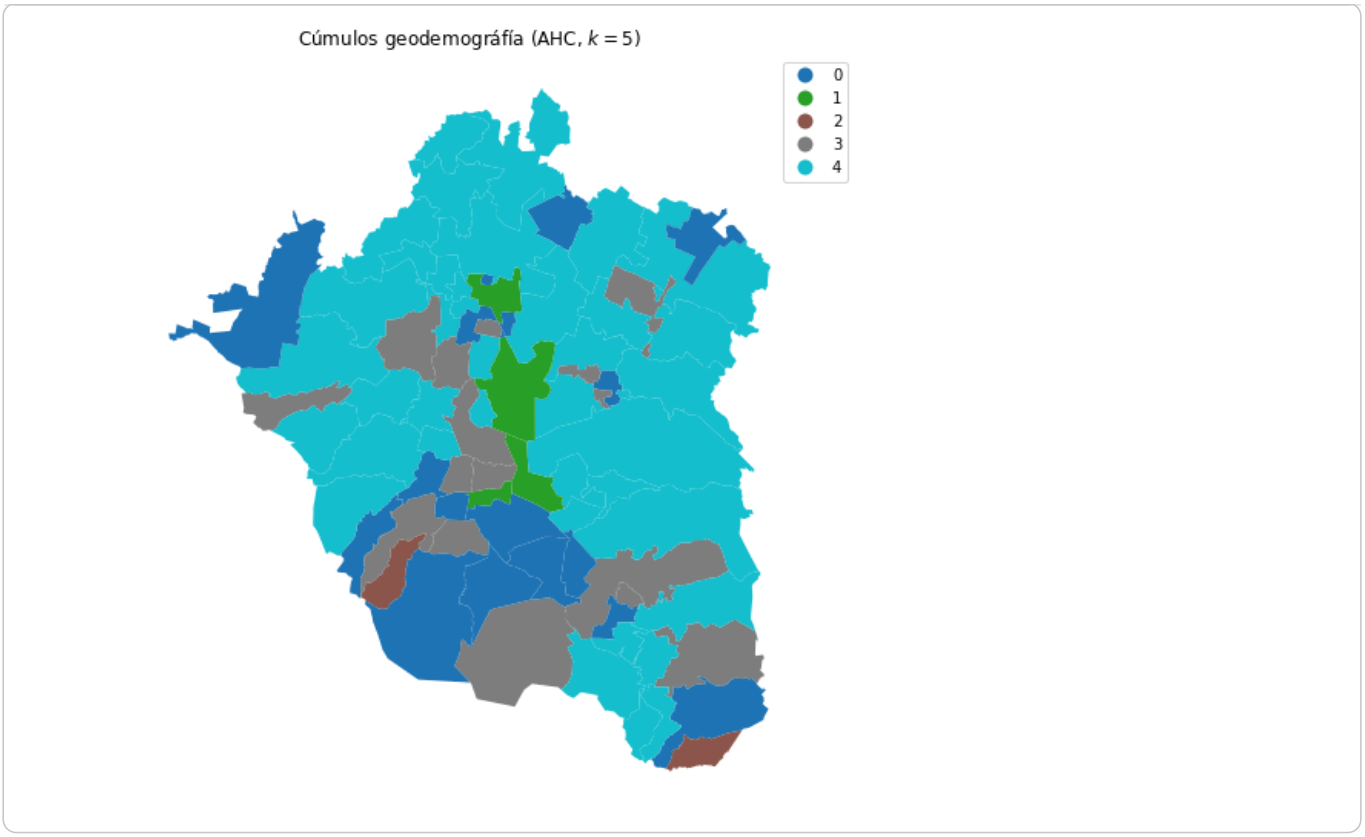
```
# Index db on cluster ID
tidy_db = zmvm.set_index('clus_ahc')
# Keep only variables used for clustering
tidy_db = tidy_db[variables]
# Stack column names into a column, obtaining
# a "long" version of the dataset
tidy_db = tidy_db.stack()
# Take indices into proper columns
tidy_db = tidy_db.reset_index()
# Rename column names
tidy_db = tidy_db.rename(columns={
                        'level_1': 'Attribute',
                        0: 'Values'})
# Check out result
tidy_db.head()
```

	clus_ahc	Attribute	Values
0	3	POB15_ANA	1.843174
1	3	POB15_SPRI	25.059767
2	3	OCUP_S_AGU	1.145264
3	3	VIV_S_WC	0.434767
4	3	VIV_S_DREN	0.434767

```
# Setup the facets
facets = sns.FacetGrid(data=tidy_db, col='Attribute', hue='clus_ahc', \
                  sharey=False, sharex=False, aspect=2, col_wrap=3)
# Build the plot as a `sns.kdeplot`
_ = facets.map(sns.kdeplot, 'Values', shade=True).add_legend()
```




```
abb['clus_ahc'] =model.labels_
# Setup figure and ax
f, ax = plt.subplots(1, figsize=(9, 9))
# Plot unique values choropleth including a legend and with no boundary lines
abb.plot(column='clus_ahc', categorical=True, legend=True, linewidth=0, ax=ax)
# Remove axis
ax.set_axis_off()
# Keep axes proportionate
plt.axis('equal')
# Add title
plt.title('Cúmulos geodemográfía (AHC, $k=5$)')
# Display the map
plt.show()
```



Algoritmos de regionalización

La regionalización es el subconjunto de técnicas de agrupamiento que imponen una restricción espacial en la clasificación. En otras palabras, el resultado de un algoritmo de regionalización contiene áreas que son espacialmente contiguas. Efectivamente, lo que esto significa es que estas técnicas agregan áreas en un conjunto más pequeño de las más grandes, llamadas regiones. En este contexto, entonces, las áreas están anidadas dentro de las regiones. Los ejemplos de este fenómeno en el mundo real incluyen los condados dentro de los estados o, en el Reino Unido, las áreas de superproducción local (LSOA) en áreas de superproducción media (MSOA). La diferencia entre esos ejemplos y la salida de un algoritmo de regionalización es que mientras que los primeros se agregan en base a principios administrativos, los últimos siguen una técnica estadística que, muy parecida a la agrupación estadística estándar, agrupa áreas que son similares en La base de un conjunto de atributos. Solo que ahora, tal agrupamiento estadístico está espacialmente restringido.

Para ilustrar estos conceptos, ejecutaremos un algoritmo de regionalización en los datos de AirBnb que hemos estado usando. En este caso, el objetivo será volver a delinear las líneas de límite de los condados de Inner London siguiendo una justificación basada en las diferentes calificaciones promedio de las propiedades de AirBnb, en lugar de las razones administrativas detrás de las líneas de límite existentes. De esta manera, las regiones resultantes representarán un conjunto consistente de áreas que son similares entre sí en términos de las calificaciones recibidas.

Definir formalmente el espacio De manera muy similar a la técnica ESDA, los métodos de regionalización requieren una representación formal del espacio que sea amigable con las estadísticas. En la práctica, esto significa que tendremos que crear una matriz de ponderaciones espaciales para las áreas que se agregarán.

Técnicamente hablando, este es el mismo proceso que hemos visto antes usando PySAL:

```
# Se crea la matriz de pesos por contiguidad de reina
w_queen = ct.Queen.from_dataframe(zmvm)
w_queen.islands
# La matriz tiene al menos un elemento sin contiguidad
#La isla tiene el ID 29
```

```
[]
```

```
#Para solucionar el problema se crea la matriz de vecinos más cercanos de orden 1
w_k1 = ct.KNN.from_dataframe(zmvm, k=1)
w_k1.islands
```

```
[]
```

```
# Y se mezclan ambas matrices para asegurar que todos los elementos tengan al menos una interacción
w = Wsets.w_union(w_queen, w_k1)
```

```
w.islands
```

```
[]
```

Creación de regiones a partir de áreas

En este punto, tenemos todas las piezas necesarias para ejecutar un algoritmo de regionalización. Para este ejemplo, usaremos una versión espacialmente restringida del algoritmo de aglomeración. Este es un enfoque similar al utilizado anteriormente (sin embargo, el funcionamiento interno del algoritmo es diferente) con la diferencia de que, en este caso, las observaciones solo se pueden etiquetar en el mismo grupo si son vecinas espaciales, según lo define la variable de retardo espacio espacial. matriz de pesos w. La forma de interactuar con el algoritmo es muy similar a la anterior. Primero establecemos los parámetros:

```
sagg13 = cluster.AgglomerativeClustering(n_clusters=5, connectivity=w.sparse)
sagg13
```

```
AgglomerativeClustering(connectivity=<76x76 sparse matrix of type '<class 'numpy.float64'>'
        with 401 stored elements in Compressed Sparse Row format>,
                        n_clusters=5)
```

Y podemos ejecutar el algoritmo llamando a la función fit:

```
# This line is required to obtain the same results always
np.random.seed(1234)
# Run the clustering algorithm
sagg13cls = sagg13.fit(zmvm[variables])
```

And then we append the labels to the table:

```
zmvm['sagg13cls'] = sagg13cls.labels_
```

```
k5hrf = zmvm.groupby('sagg13cls')[variables].mean()
# Muestra la tabla transpuesta
k5hrf.T
```

sagg13cls	0	1	2	3	4
POB15_ANA	3.418970e+00	2.069091e+00	2.888097e+00	3.022885e+00	4.382901e+00
POB15_SPRI	3.186144e+01	2.369995e+01	3.043039e+01	3.031255e+01	3.395533e+01
OCUP_S_AGU	4.258384e+00	4.341090e+00	4.292387e+00	1.831037e+00	9.203395e+00
VIV_S_WC	3.997815e-01	5.011847e-01	2.984959e-01	3.387313e-01	3.318017e+00
VIV_S_DREN	3.997815e-01	5.011847e-01	2.984959e-01	3.387313e-01	3.318017e+00
VIV_HAC	3.111113e+01	1.819298e+01	2.536100e+01	2.525895e+01	3.213286e+01
VIV_S_REFR	1.905360e+07	1.119207e+07	4.124001e+07	4.537511e+07	2.355748e+06
PEAOCUPADA	3.996854e+01	4.305174e+01	3.995805e+01	4.142933e+01	3.659292e+01
POB_S_DERE	3.126782e+07	1.350350e+07	6.998481e+07	6.954782e+07	4.030334e+06
DISP_INFRA	7.711675e+00	1.731728e+01	4.830606e+00	7.379724e+00	3.900711e+00
DISP_RH	1.891574e+01	4.634890e+01	1.661728e+01	1.929192e+01	7.667474e+00
DISP_EQUIP	2.669507e+00	6.998537e+00	2.270385e+00	3.557688e+00	1.890836e+00

```
# Index db on cluster ID
tidy_db = zmvm.set_index('sagg13cls')
# Keep only variables used for clustering
tidy_db = tidy_db[variables]
# Stack column names into a column, obtaining
# a "long" version of the dataset
tidy_db = tidy_db.stack()
# Take indices into proper columns
tidy_db = tidy_db.reset_index()
# Rename column names
tidy_db = tidy_db.rename(columns={
                        'level_1': 'Attribute',
                        0: 'Values'})
# Check out result
tidy_db.head()
```

	sagg13cls	Attribute	Values
0	1	POB15_ANA	1.843174
1	1	POB15_SPRI	25.059767
2	1	OCUP_S_AGU	1.145264
3	1	VIV_S_WC	0.434767
4	1	VIV_S_DREN	0.434767

```
# Setup the facets
facets = sns.FacetGrid(data=tidy_db, col='Attribute', hue='sagg13cls', \
                  sharey=False, sharex=False, aspect=2, col_wrap=3)
# Build the plot as a `sns.kdeplot`
_ = facets.map(sns.kdeplot, 'Values', shade=True).add_legend()
```

```
C:\Users\ferna\.conda\envs\esda\lib\site-packages\seaborn\distributions.py:316: UserWarning: Dataset has 0 va
  warnings.warn(msg, UserWarning)
C:\Users\ferna\.conda\envs\esda\lib\site-packages\seaborn\distributions.py:316: UserWarning: Dataset has 0 va
  warnings.warn(msg, UserWarning)
C:\Users\ferna\.conda\envs\esda\lib\site-packages\seaborn\distributions.py:316: UserWarning: Dataset has 0 va
  warnings.warn(msg, UserWarning)
C:\Users\ferna\.conda\envs\esda\lib\site-packages\seaborn\distributions.py:316: UserWarning: Dataset has 0 va
  warnings.warn(msg, UserWarning)
```

Mapeo de las regiones resultantes

En este punto, la columna sagg13cls no es diferente de k5cls: una variable categórica que se puede mapear en un único valor choropleth. De hecho, el siguiente fragmento de código es exactamente el mismo que antes, sólo reemplazando el nombre de la variable que se asignará y el título:

```
Empieza a programar o a crear código con IA.
```

```
C:\Users\ferna\.conda\envs\esda\lib\site-packages\seaborn\distributions.py:316: UserWarning: Dataset has 0 va
```

```
# Ajusta figura y ejes
f, ax = plt.subplots(1, figsize=(9, 9))
# Grafica por coro´pletas
zmvm.plot(column='sagg13cls', categorical=True, legend=True, linewidth=0, ax=ax)
# Remueve ejes
ax.set_axis_off()
# mantiene los ejes proporcionados
plt.axis('equal')
#AGrega título
plt.title('Accesibilidad a la salud en la ZMVM')
# Muestra el mapo
plt.show()
```

```
  warnings.warn(msg, UserWarning)
ions.py:316: UserWarning: Dataset has 0 va
ions.py:316: UserWarning: Dataset has 0 va
ions.py:316: UserWarning: Dataset has 0 va
ions.py:316: UserWarning: Dataset has 0 va
ions.py:316: UserWarning: Dataset has 0 va
```