**Clase 7: Geopandas**

**Sistemas de Información Geográfica (SIG):** Sistemas que permiten almacenar datos geoespaciales para su consulta, maniulación y representación. *Ciencia de la Información Geográfica*. El sistema permite separar la información en diferentes capas temáticas, las cuales pueden presentarse individualmente o todas juntas.

**Datos Geoespaciales:** Datos *georreferenciados.* La **georreferenciación** es la técnica de **posicionamiento espacial** de una **entidad** en una **localización geográfica única** y bien definida en un ***sistema de coordenadas***. Los **geodatos** tienen información *implícita* (cantidad de habitantes, referencia catastral, dirección postal, etc.) o *explícita* (coordenadas de ubicación a partir de GPS, etc.).

**Sistema de Coordenadas Geográficas:** Sistemas que permiten referenciar un objeto en una posición espacial determinada. El más usual es el **WGS84** que usa **latitud** (líneasparalelas al Ecuador) y **longitud** (líneas paralelas al Meridiano de Greenwich). La **latitud** toma valores entre 0 y 90 si están al norte del Ecuador, o entre 0 y -90 si están al sur del Ecuador. También pueden nomenclarse con el módulo de su valor y la aclaración de si son Norte o Sur. La **longitud** toma valores entre 0 y 180 si están al este de Greenwich, o entre 0 y -180 si están al oeste de Greenwich. También pueden nomenclarse con el módulo de su valor y la aclaración de si son Este u Oeste.

**Tipos de Datos Geoespaciales:** Hay 2. **Raster** y **Vectorial**.

**Raster:** Imágenes digitales o mapas. Se representan con mallas o cuadrículas. Divide el espacio en celdas regulares donde cada una representa un único valor. Similar a la fotografía digital donde cada pixel es la unidad de menor información de una imagen, y una combinación de pixeles crea la imagen. Se centran en las **propiedades del espacio**, tales como elevación del terreno, nivel de luz, variedad de cultivos. No tan precisos para la localización en el espacio.

**Vectorial:** Representan objetos que provienen de diversas fuentes, tales como negocios, localidades, barrios, rutas. Son datos ***discretos***, con límites definidos y con una **localización precisa** sobre el espacio. Se usan 3 formas para modelar objetos:

* **Punto:** Se usan en entidades geográficas que se pueden expresar mejor en un único punto de referencia, tales como una estación de subte o un edificio.
* **Líneas undimensionales o polilíneas:** rasgos lineales como ríos, rutas, ferrocarriles, etc. Se puede medir la *distancia.*
* **Polígono:** Se usan para representar elementos geográficos que cubren un área particular de la superficie de la tierra, tales como lagos, provincias, países, etc. Se puede medir el *perímetro* y el *área* de los mismos.

Los formatos de archivos con datos Geoespaciales se los denomina formato SIG.

Los **raster** se almacenan en tipos de archivo que sirven para almacenar imágenes (TIFF, JPEG, PNG, etc.)

Los **vectorial** más comunes son: **SHP (Shapefile)** Es el más común y el estándar de la industria; o bien **CSV / GeoCSV**, que representa datos en forma de tablas separados por comas. Suelen encontrarse en publicaciones de organismos. Ocupan poco espacio y es fácil compartirlos.

Hay otros tipos más tales como GeoJSON, KMZ/KML, OSM, PostGIS + PostgreSQL, entre otros.

**GeoPandas:** Biblioteca que permite trabajar con datos Geoespaciales en Python.

* Extiende la funcionalidad de Pandas: Todas las operaciones aplicables a Series y Dataframes son aplicables acá también.
* Tiene una interfaz con la librería **matplotlib** para generar mapas. Usando el método **plot()** pueden mapearse las figuras geométricas.
* Permite operaciones sobre los datos geométricos (puntos, líneas, polígonos), usando la librería **shapely**.
* Facilita el acceso a datos en múltiples formatos de archivos, extendiendo las operaciones de la librería **fiona**.
* Hace **proyecciones.** Permite usar distintos tipos de coordenadas para referenciar la posición geográfica de los datos geoespaciales.
* Los elementos con los que trabaja Geopandas son las **Geoseries** y los **GeoDataFrames.** La diferencia entre estos y las Series y DataFrames es que estos deben contener **al** **menos una columna con tipos de datos geoespaciales**, la cual normalmente se la llama por default **geometry.**

**Formas Geométricas:**

* **POINT.**
* **LINESTRING.**
* **POLYGON**

También se puede representar colecciones de formas:

* **MULTIPOINT**
* **MULTILINESTRING**
* **MULTIPOLYGON**

**En Python:**

**Trabajando con Puntos:** Se tiene un DataFrame con dos columnas X e Y que representan la Longitud y Latitud de unos lugares determinados. Estos datos son numéricos; no formas geométrias. Para convertirlos, se usa el método gpd.points\_from\_xy(nombreDataFrame.X, nombreDataFrame.Y). En **importante notar que primero debe darse como input la Longitud y luego la Latitud (LoLa).**

geo\_locales = gpd.GeoDataFrame(df\_locales, geometry=gpd.points\_from\_xy(df\_locales.X, df\_locales.Y))

Con este Código, está creando un GeoDataFrame a partir de un DataFrame, creando al mismo tiempo la columna geometry, que en este caso está compuesta por puntos.

**Trabajando con Polígonos:** Se tiene un archivo GeoCSV con una columna de datos Geoespaciales WKT, donde se contienen polígonos. En forma análoga al caso anterior, al importar el archivo CSV, la columna WKT no es interpretada como un tipo de datos geoespaciales. Usando la librería **shapely** se lo transforma:

barrios["WKT"] = barrios["WKT"].apply(shapely.wkt.loads)

Le pide a la librería que trabaje sobre WKT para convertirlo en un tipo de datos geoespacial.

geo\_barrios = gpd.GeoDataFrame(barrios, geometry='WKT')

Crea un nuevo GeoDataFrame a partir del DataFrame original y definiendo la columna Geometry usando el ahora tipo de datos geoespacial ‘WKT’.

geo\_barrios.plot()

Ahora que Geometry contiene un conjunto de polígonos, le pide que lo grafique.

**Trabajando con Líneas:** Las líneas se generan usando dos puntos. Con la longitud y latitud de cada punto, puede generarse un tipo de datos Point usando la librería **shapely**:

from shapely.geometry import Point

Punto\_determinado1 = Point (#Longitud1, #Latitud1)

Punto\_determinado2 = Point (#Longitud2, #Latitud2)

Con el **método Linestring** de la librería shapely es posible crear la línea

from shapely.geometry import Linestring

linea = LineString([Punto\_determinado1, Punto\_determinado2])

Ahora que tenemos los 2 puntos y la línea que los une, podemos graficar la línea sobre un mapa. Necesitamos crear un tipo de Datos Geoseries que contenga ambos puntos y la línea:

linea\_a\_geo = gpd.GeoSeries([Punto\_determinado1, Punto\_determinado2, linea])

Ahora pasa a graficar:

fig, ax = plt.subplots()

ax.set\_aspect('equal')

geo\_barrios.plot(ax=ax, color='white', edgecolor='black') #grafica el mapa de buenos aires.

linea\_a\_geo.plot(ax=ax, color='red') #grafica la línea.

plt.show();

**Determinar si un punto se encuentra dentro de un polígono**: Se puede ver con el método **GeoDataFrame.contains**

Se generan dos DataFrames: Uno con el polígono del barrio y otro con el punto.

Grafica ambos Dataframes y verifica visualmente si el punto se encuentra o no efectivamente dentro del polígono:

fig, ax = plt.subplots()

ax.set\_aspect('equal')

geo\_monserrat.plot(ax=ax, color='white', edgecolor='black')

geo\_rey\_castro.plot(ax=ax, marker='o', color='red', markersize=40)

plt.show();

Crea un nuevo punto a partir de la Longitud y Latitud (dice que si se usa el geodataframe que ya tenemos daría error). Lo genera usando **Point**.

Por último, usa el método contains en el polígono, poniendo el punto a verificar entre paréntesis:

Polígono\_donde\_buscar.contains(punto\_a\_ver\_si\_está\_adentro)