Contexto General:

El código trata sobre operaciones unarias (operaciones que afectan a un solo objeto) sobre **GeoDataFrames (GDF)**. Un GeoDataFrame es un tipo de estructura de datos en la librería **GeoPandas** (una extensión de **Pandas** para datos espaciales) que permite almacenar datos geoespaciales como **puntos**, **líneas** y **polígonos**.

En este código, se trabajan con varios tipos de operaciones espaciales, como filtrado, combinación de geometrías, cálculo de áreas, y más.

Cargar los Datos:

```
import geopandas as gpd

# #world
world_rivers = gpd.read_file(linkWorldMap, layer='rivers')
# #brazil
brazil5880 = gpd.read_file(linkBrazil, layer='country')
airports_brazil5880 = gpd.read_file(linkBrazil, layer='airports')
states_brazil5880 = gpd.read_file(linkBrazil, layer='states')
municipalities_brazil5880 = gpd.read_file(linkBrazil,
layer='municipalities')
# #some indicators
indicators = gpd.read_file(linkIndicators)
```

- **`geopandas.read_file()`**: Lee archivos espaciales (como shapefiles, GeoPackage, etc.) y devuelve un **GeoDataFrame (GDF)**, que es una estructura similar a un DataFrame de **Pandas**, pero con una columna especial para las geometrías (puntos, líneas, polígonos).

Filtrado Usando Herencia de Pandas:

1. Filtrado con `iloc` y `loc`:

Estas funciones son de **Pandas** y también se aplican en **GeoPandas**.

`iloc` se utiliza para acceder a datos por **índices de posición**:

```
states_brazil5880.iloc[:5, 1:]
```

- Aquí se seleccionan las primeras 5 filas y todas las columnas a partir de la segunda (índice 1).

`loc` se utiliza para acceder a datos por **etiquetas (nombres)** de filas y columnas:

```
states_brazil5880.loc[:5, 'state_code':]
```

- Esto selecciona las primeras 5 filas y todas las columnas a partir de 'state_code'.

2. Filtrado con `query`:

Permite hacer consultas con condiciones como en SQL:

```
condition = 'elevation_ft > 5000 and airport_type=="small_airport"'
airports_brazil5880.query(condition)
```

Aquí, se filtran los aeropuertos cuya altitud sea mayor a 5000 pies y sean de tipo "small_airport".

3. Filtrado con `isin()`:

Filtra las filas según si los valores de una columna están en una lista dada:

```
choices = ['large_airport', 'seaplane_base']
airports_brazil5880[airports_brazil5880.airport_type.isin(choices)]
```

En este caso, se filtran los aeropuertos de tipo "large_airport" o "seaplane_base".

4. Filtrado de texto:

Filtra según condiciones sobre cadenas de texto, como **comienza con**, **termina con**, o **contiene**:

```
airports_brazil5880[airports_brazil5880.airport_name.str.startswith('Presi')]
```

Esto filtra los aeropuertos cuyo nombre comienza con "Presi".

5. Filtrado con valores faltantes:

Se pueden filtrar filas donde una columna tenga valores **faltantes** (`NaN`) o no:

```
airports_brazil5880[airports_brazil5880.elevation_ft.isna()]
```

Aquí se filtran los aeropuertos con valores faltantes en la columna **elevation_ft**.

Filtrado Espacial (Por Geometría):

1. Filtrado por Atributos de Geometría:

Puedes acceder a atributos geométricos como el **área**, **longitud** o **centroide** de las geometrías:

```
states_brazil5880[states_brazil5880.area > 1000000000000]
```

Este código filtra los estados cuyo área es mayor que 1,000,000 km².

2. Slicing usando `cx`:

La propiedad **`cx`** permite filtrar por coordenadas. En este ejemplo, se seleccionan aeropuertos que estén al **norte** de un punto central (como el centroide de Brasil):

```
airports_brazil5880.cx[:, mid_y:]
```

Esto selecciona todos los aeropuertos al norte del punto central de Brasil.

3. Uso de la propiedad `centroid`:

Un **centroide** es el punto medio de una geometría. Puedes obtener el centro de un país o región y usarlo para filtrar objetos espaciales en función de su posición relativa a ese centro.

Operaciones de Combinación de Geometrías:

1. Unión de Geometrías (`union_all()`):

El método **`union_all()`** combina todas las geometrías de un **GeoDataFrame** en una sola:

```
Rondonia_union = muniRondonia.union_all()
```

Esto une todos los polígonos de los municipios de Rondônia en un solo objeto.

2. Dissolve:

La función **`dissolve()`** agrupa geometrías basadas en una columna específica (como el nombre del estado) y las combina en una sola geometría:

```
Rondonia_dissolved = muniRondonia.dissolve(by='state_name')
```

En este caso, todos los municipios de un estado se combinan en una única geometría.

3. Dissolve con Agregación:

El método **`dissolve()`** también puede combinarse con una función de agregación. Por ejemplo, se puede calcular el promedio de una variable como la **fragilidad** de los países:

```
indicatorsByRegion = indicators.dissolve(by="region",
aggfunc={"fragility": "mean"})
```

Aquí, se agrupa por **región** y se calcula el **promedio** de la variable **fragilidad**.

Operaciones con Convex Hull:

El **convex hull** es un polígono que envuelve un conjunto de puntos de forma mínima. Se usa para crear un polígono que cubra un conjunto de puntos:

```
large_airports.convex_hull.plot()
```

Primero se obtiene el **convex hull** de todos los aeropuertos grandes. Si no se combinan las geometrías, el **convex hull** no se mostrará correctamente.

Buffering Geometrías:

El **buffer** es una operación que crea una zona alrededor de una geometría (punto, línea o polígono). Por ejemplo, si tenemos un río, podemos crear un buffer alrededor del mismo:

```
AmazonSystem_5880.buffer(50000).plot()
```

Aquí, se crea un buffer de **50,000 metros** alrededor del sistema fluvial del Amazonas.

Validación de Geometrías:

Las geometrías pueden ser inválidas debido a errores de digitación o resultados de operaciones como la **unión** o **dissolve**. Para detectar estas geometrías inválidas, se utiliza el método **`is_valid`**:

```
S_brazil[~S_brazil.is_valid]
```

Aquí, se verifican las geometrías inválidas en los estados del sur de Brasil.

- **`make_valid()`** es una función que corrige las geometrías inválidas, y **`buffer(0)`** también se usa a menudo para arreglar geometrías sencillas que tienen pequeños errores.

Resumen de Funciones Clave:

- `gpd.read_file()`: Lee un archivo espacial y lo convierte en un GeoDataFrame.
- `iloc`, `loc`: Se utilizan para seleccionar datos en un DataFrame por índice o nombre de columna, respectivamente.
- `query()`: Permite realizar consultas en los datos como si fuera SQL.
- `isin()`: Filtra datos basados en una lista de valores.
- `geometry`: Una columna especial de un GeoDataFrame que almacena las geometrías espaciales (puntos, líneas, polígonos).
- `union_all()`: Combina varias geometrías en una sola.
- `dissolve()`: Combina geometrías basadas en un criterio de agrupación (por ejemplo, estado o región).
- `convex_hull`: Crea un polígono envolvente de un conjunto de puntos.
- `buffer()`: Crea un buffer alrededor de una geometría.
- `is_valid` y `make_valid()`: Detectan y corrigen geometrías inválidas.

Este tipo de operaciones son esenciales cuando trabajas con datos espaciales y te permiten realizar análisis espaciales complejos como calcular áreas, distancias, agrupar regiones, etc.

Te sugiero que sigas practicando y experimentando con estos comandos en tu propio código, para que puedas dominar las operaciones espaciales y entender cómo se manipulan y analizan los datos geoespaciales. ¡Ánimo!