Python Programming

GeoPandas

Dr. Chun-Hsiang Chan
Department of Geography
National Taiwan Normal University

Outlines

- Intro. to GeoPandas
- Differences between Geopandas and Pandas
- File I/O
- Coordinate Reference System
- Make a GeoDataFrame for Point\Polyline
- Basic Preprocessing in GeoPandas
- Map Visualization
- GeoProcessing :: Fundamentals

Intro. to GeoPandas

- **GeoPandas** is an open-source Python library that extends the popular **Pandas** library to handle geospatial data. It simplifies the process of working with geographic information by introducing spatial data structures and operations that integrate seamlessly with the familiar pandas workflow.
- GeoPandas builds upon pandas.DataFrame, adding a special geometry column that can store various geometric types (e.g., Points, Lines, and Polygons). This allows you to manipulate and analyze spatial data using the same intuitive operations you use on regular tabular data.
- **# Shapely #Fiona #Pyproj**

Intro. to GeoPandas

- **Simplified Spatial Analysis:**
- With GeoPandas, tasks like spatial joins, overlays, and clipping become straightforward, enabling you to quickly answer geographic questions and perform spatial analyses with minimal code.
- **Integrated Visualization:**
- GeoPandas provides easy-to-use plotting capabilities using Matplotlib, and it can also interface with interactive mapping libraries like Folium and Kepler.gl. This integration helps you visualize geospatial data in both static and dynamic formats.
- **Real-World Applications:**
- GeoPandas is widely used in diverse fields such as urban planning, environmental science, transportation, public health, and more—any domain where geographic data analysis is essential.

Differences between Geopandas and Pandas

Attributes	Pandas	GeoPandas
Data Structure	DataFrame, Series	GeoDataFrame & GeoSeries (Point, Polyline, and Polygon)
Support Formats	CSV, Excel, SQL, and JSON	Shapefile, GeoJSON, and KML
Operations	Table operations	Spatial analysis
Coordinate Reference System	Null	Define CRS & Transform CRS
Visualization	Matplotlib (bulit-in)	Folium or Kepler.gl
Library Ecosystem	Numpy	**Shapely** for geometric operations. **Fiona** for file I/O with geospatial formats. **pyproj** for handling projections and coordinate transformations.

File I/O

```
# import packages
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import geopandas as gpd
# load data
os.chdir('GeoPandas/')
town = gpd.read_file('TOWN_MOI_1120825.shp')
codebase = gpd.read_file('G97_63000_U0200_2015.shp')
pop = pd.read_csv('LOCATION_BASE_DATA.csv')
```

File I/O

geometry ➔ CRS?

town.head()

	TOWNID	TOWNCODE	COUNTYNAME	TOWNNAME	TOWNENG	COUNTYID	COUNTYCODE	geometry
0	V02	10014020	臺東縣	成功鎮	Chenggong Township	V	10014	POLYGON ((121.40981 23.21370, 121.40984 23.213...
1	T21	10013210	屏東縣	佳冬鄉	Jiadong Township	T	10013	POLYGON ((120.54845 22.46067, 120.54853 22.460...
2	P13	10009130	雲林縣	麥寮鄉	Mailiao Township	P	10009	POLYGON ((120.30198 23.81624, 120.30198 23.815...
3	V11	10014110	臺東縣	綠島鄉	Lüdao Township	V	10014	MULTIPOLYGON (((121.49154 22.67746, 121.49184 ...
4	V16	10014160	臺東縣	蘭嶼鄉	Lanyu Township	V	10014	MULTIPOLYGON (((121.61179 21.94290, 121.61133 ...

codebase.head()

	U_ID	CODEBASE	CODE1	CODE2	TOWN_ID	TOWN	COUNTY_ID	COUNTY	X	Y	AREA	geometry
0	1972	A6310-0024-00	A6310-01-006	A6310-01	63000100	內湖區	63000	臺北市	310951.84060	2.775913e+06	362112.90727	POLYGON Z ((310568.684 2775912.720 0.000, 3105...
1	1973	A6311-0791-00	A6311-61-005	A6311-61	63000110	士林區	63000	臺北市	300103.56005	2.776286e+06	17541.31869	POLYGON Z ((299986.213 2776330.686 0.000, 3000...
2	1974	A6311-0787-00	A6311-67-002	A6311-67	63000110	士林區	63000	臺北市	302742.35826	2.776312e+06	3627.65319	POLYGON Z ((302752.204 2776365.558 0.000, 3027...
3	1975	A6311-0793-00	A6311-62-004	A6311-62	63000110	士林區	63000	臺北市	302298.08171	2.776278e+06	15430.37397	POLYGON Z ((302336.695 2776362.841 0.000, 3023...
4	1976	A6311-0807-00	A6311-67-008	A6311-67	63000110	士林區	63000	臺北市	302823.86029	2.776199e+06	52775.40011	POLYGON Z ((302717.351 2776356.819 0.000, 3027...

pop.head()

	CODEBASE	DAY_NIGHT	CNT
0	A6813-0111-00	D	6
1	A6813-0110-00	D	32
2	A6813-0109-00	D	20
3	A6813-0108-00	D	22
4	A6813-0107-00	D	25

File I/O

- For data output, GeoPandas also provides several file formats for various usages, such as ESRI Shapefile, PostGIS, Feather, and Parquet.

```
# save ESRI Shapefile
gdf.to_file('test.shp')
```

Coordinate Reference System

In Taiwan ::

- **WGS84 (EPSG:4326)** is typically used as the geographic coordinate system for latitude and longitude.
- **TWD97TM2 (EPSG:3826)** is commonly used as the projected coordinate system for metre-based measurements.
- In general, we may face four situations in setting CRS: (1) file with the correct CRS; (2) with incorrect CRS; (3) with string-format geometry; (4) without CRS.

Coordinate Reference System

- **How to inspect the CRS of GeoDataFrame**

```
# show the CRS of GeoDataFrame
```

town.crs

```
<Geographic 2D CRS: GEOGCS["GCS_TWD97[2020]",DATUM["Taiwan_Datum_1997" ...>
Name: GCS_TWD97[2020]
Axis Info [ellipsoidal]:
- lon[east]: Longitude (Degree)
- lat[north]: Latitude (Degree)
Area of Use:
- undefined
Datum: Taiwan_Datum_1997
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
```

codebase.crs

```
<Projected CRS: EPSG:3826>
Name: TWD97 / TM2 zone 121
Axis Info [cartesian]:
- X[east]: Easting (metre)
- Y[north]: Northing (metre)
Area of Use:
- name: Taiwan, Republic of China - between 120°E and 122°E, onshore and offshore - Taiwan Island.
- bounds: (119.99, 20.41, 122.06, 26.72)
Coordinate Operation:
- name: Taiwan 2-degree TM zone 121
- method: Transverse Mercator
Datum: Taiwan Datum 1997
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
```

Coordinate Reference System

(1) File with the correct CRS.
- ⇜ no extra action should be done.

(2) File with incorrect CRSs.
- ⇜ by using **to_crs()** to transform the original CRS to the correct CRS.

(3) File with string-format geometry.
- ⇜ by using **wkt** package to translate the string information of the geometry column to usable geometry format.

(4) File without CRSs.
- ⇜ by using **set_crs()** to specify the matched CRS.

Coordinate Reference System

(1) File with the correct CRS.

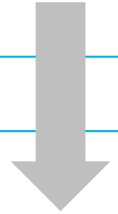
↬ no extra action should be done.

(2) File with incorrect CRSs.

↬ by using **to_crs()** to transform the original CRS to the correct CRS.

```
# from 4326 to 3826
town = town.to_crs('epsg:3826')
town.crs
```

```
<Geographic 2D CRS: GEOGCS["GCS_TWD97[2020]",DATUM["Taiw
an_Datum_1997" ...>
Name: GCS_TWD97[2020]
Axis Info [ellipsoidal]:
- lon[east]: Longitude (Degree)
- lat[north]: Latitude (Degree)
Area of Use:
- undefined
Datum: Taiwan_Datum_1997
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
```



```
<Projected CRS: EPSG:3826>
Name: TWD97 / TM2 zone 121
Axis Info [cartesian]:
- X[east]: Easting (metre)
- Y[north]: Northing (metre)
Area of Use:
- name: Taiwan, Republic of China - between 120°E and 12
2°E, onshore and offshore - Taiwan Island.
- bounds: (119.99, 20.41, 122.06, 26.72)
Coordinate Operation:
- name: Taiwan 2-degree TM zone 121
- method: Transverse Mercator
Datum: Taiwan Datum 1997
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
```

Coordinate Reference System

(3) File with string-format geometry.

- by using **wkt** package to translate the string information of the geometry column to usable geometry format.

```
# load camera data
cam2 = pd.read_csv('臺北市固定測速照相地點表v2.csv')
# preview data
cam2.head()
```

	編號	功能	設置路段	設置地點	緯度	經度	轄區	拍攝方向	速限-速度限制	geometry
0	1	測速	環河北路3段	葫蘆街	25.082459	121.507372	士林分隊	南向北	50	POINT (121.5073719 25.08245929)
1	2	測速	仰德大道2段	115巷口	25.108241	121.546019	士林分隊	下山方向	40	POINT (121.5460187 25.10824082)
2	3	測速	仰德大道4段75號	格致中學前	25.132062	121.546455	士林分隊	上下山雙向	40	POINT (121.5464555 25.13206152)
3	4	測速	承德路4段	百齡高中前	25.086516	121.522987	士林分隊	南向北	50	POINT (121.5229866 25.08651629)
4	5	闖紅燈	中正路	與基河路口	25.093230	121.519831	士林分隊	西向東	50	POINT (121.5198312 25.09323)

Coordinate Reference System

(3) File with string-format geometry.

- by using **wkt** package to translate the string information of the geometry column to usable geometry format.

```
# inspect the data type of geometry column
cam2.geometry[0]
```

```
cam2.geometry[0]
```

```
'POINT (121.5073719 25.08245929)'
```

```
0       POINT (121.5073719 25.08245929)
1       POINT (121.5460187 25.10824082)
2       POINT (121.5464555 25.13206152)
3       POINT (121.5229866 25.08651629)
4          POINT (121.5198312 25.09323)
                     ...
140     POINT (121.5573087 25.02906139)
141     POINT (121.5996961 25.06679101)
142      POINT (121.5277694 25.0961568)
143     POINT (121.4927469 25.02610889)
144     POINT (121.5684952 25.04768719)
Name: geometry, Length: 145, dtype: object
```

Coordinate Reference System

(3) File with string-format geometry.

- by using **wkt** package to translate the string information of the geometry column to usable geometry format.

```
# translate string object to geometry object
cam2['geometry'] = cam2['geometry'].apply(wkt.loads)
```

```
cam2['geometry'][0]
```

```
0      POINT (121.5073719 25.08245929)
1      POINT (121.5460187 25.10824082)
2      POINT (121.5464555 25.13206152)
3      POINT (121.5229866 25.08651629)
4         POINT (121.5198312 25.09323)
                    ...
140    POINT (121.5573087 25.02906139)
141    POINT (121.5996961 25.06679101)
142     POINT (121.5277694 25.0961568)
143    POINT (121.4927469 25.02610889)
144    POINT (121.5684952 25.04768719)
Name: geometry, Length: 145, dtype: object
```

Coordinate Reference System

(3) File with string-format geometry.

- by using **wkt** package to translate the string information of the geometry column to usable geometry format.

```
# translate string object to geometry object
cam2['geometry'] = cam2['geometry'].apply(wkt.loads)
```

```
cam2['geometry'][0]
```

```
0      POINT (121.5073719 25.08245929)
1      POINT (121.5460187 25.10824082)
2      POINT (121.5464555 25.13206152)
3      POINT (121.5229866 25.08651629)
4         POINT (121.5198312 25.09323)
                    ...
140    POINT (121.5573087 25.02906139)
141    POINT (121.5996961 25.06679101)
142     POINT (121.5277694 25.0961568)
143    POINT (121.4927469 25.02610889)
144    POINT (121.5684952 25.04768719)
Name: geometry, Length: 145, dtype: object
```

Coordinate Reference System

(4) File without CRSs.

- by using **set_crs()** to specify the matched CRS.

```
# read tree data
gdf.set_crs = 'epsg:3826'
```

Make a GeoDataFrame for Point

- Practically, we cannot always obtain complete data for further analyses; additionally, governments and most online datasets adopt CSV and XLSX formats to release spatial information.
- Here, we demonstrate a simple example.

```
# read tree data
tree = gpd.read_file('TaipeiTree.csv')
# preview the dataset
tree.head()
```

	TreeID	Dist	Region	RegionRemark	TreeType	Diameter	TreeHeight	SurveyDate	TWD97X	TWD97Y	UpdDate	geometry
0	BT0614021096	士林區	文林路	東側人行道	榕樹	54	16	2022-11-18	302404.92	2777358.41	2025-02-14 16:23:46	None
1	BT0614021097	士林區	文林路	東側人行道	白千層	45	6.8	2022-11-18	302408.24	2777352.36	2025-02-14 16:23:46	None
2	BT0614021098	士林區	文林路	東側人行道	茄苳	6	8.7	2022-11-18	302413.58	2777342.69	2025-02-14 16:23:46	None
3	BT0614031095	士林區	文林路	西側人行道	白千層	22	7.7	2022-11-18	302369.5	2777349.52	2025-02-14 16:23:46	None
4	BT0614031096	士林區	文林路	西側人行道	茄苳	6	7.7	2022-11-18	302374.59	2777340.07	2025-02-14 16:23:46	None

Make a GeoDataFrame for Point

```
# transform Pandas.DataFrame to GeoPandas.GeoDataFrame
tree = gpd.GeoDataFrame(tree,
        geometry=gpd.points_from_xy(tree['TWD97X'],tree['TWD97Y']),
        crs='epsg:3826')
# preview the dataset
tree.head()
```

	TreeID	Dist	Region	RegionRemark	TreeType	Diameter	TreeHeight	SurveyDate	TWD97X	TWD97Y	UpdDate	geometry
0	BT0614021096	士林區	文林路	東側人行道	榕樹	54	16	2022-11-18	302404.92	2777358.41	2025-02-14 16:23:46	POINT (302404.920 2777358.410)
1	BT0614021097	士林區	文林路	東側人行道	白千層	45	6.8	2022-11-18	302408.24	2777352.36	2025-02-14 16:23:46	POINT (302408.240 2777352.360)
2	BT0614021098	士林區	文林路	東側人行道	茄苳	6	8.7	2022-11-18	302413.58	2777342.69	2025-02-14 16:23:46	POINT (302413.580 2777342.690)
3	BT0614031095	士林區	文林路	西側人行道	白千層	22	7.7	2022-11-18	302369.5	2777349.52	2025-02-14 16:23:46	POINT (302369.500 2777349.520)
4	BT0614031096	士林區	文林路	西側人行道	茄苳	6	7.7	2022-11-18	302374.59	2777340.07	2025-02-14 16:23:46	POINT (302374.590 2777340.070)

Make a GeoDataFrame for Polyline

In practice, we usually create a Polyline data from various sources. Here, we demonstrate an example to

```
# read data
od_csv = pd.read_csv('112年11月行政區電信信令日夜起訖矩陣(平日)人口統計資料_縣市.csv')
od_csv.head()
```

	COUNTY_ID	COUNTY	TO_65000	TO_63000	TO_68000	TO_66000	TO_67000	TO_64000	TO_10002	TO_10004	...	TO_10013	TO_10014	TO_10015	TO_1
0	縣市代碼	縣市名稱	TO新北市	TO臺北市	TO桃園市	TO臺中市	TO臺南市	TO高雄市	TO宜蘭縣	TO新竹縣	...	TO屏東縣	TO臺東縣	TO花蓮縣	TO澎
1	65000	新北市	14.647	3.47	0.514	0.023	0.011	0.016	0.016	0.029	...	0.004	0.003	0.006	
2	63000	臺北市	0.728	9.652	0.122	0.017	0.009	0.012	0.01	0.013	...	0.003	0.002	0.004	
3	68000	桃園市	0.358	0.278	9.456	0.016	0.008	0.01	0.005	0.111	...	0.003	0.002	0.003	
4	66000	臺中市	0.018	0.026	0.023	12.742	0.015	0.017	0.003	0.014	...	0.005	0.002	0.002	

Make a GeoDataFrame for Polyline

```
# preprocessing dataset
od_csv = od_csv.iloc[1:,:].reset_index(drop=True)
del od_csv['COUNTY']
del od_csv['INFO_TIME']
od_csv.head()
```

	COUNTY_ID	TO_65000	TO_63000	TO_68000	TO_66000	TO_67000	TO_64000	TO_10002	TO_10004	TO_10005	...	TO_10010	TO_10013	TO_10014	TO_10
0	65000	14.647	3.47	0.514	0.023	0.011	0.016	0.016	0.029	0.009	...	0.005	0.004	0.003	0
1	63000	0.728	9.652	0.122	0.017	0.009	0.012	0.01	0.013	0.005	...	0.002	0.003	0.002	0
2	68000	0.358	0.278	9.456	0.016	0.008	0.01	0.005	0.111	0.012	...	0.003	0.003	0.002	0
3	66000	0.018	0.026	0.023	12.742	0.015	0.017	0.003	0.014	0.064	...	0.009	0.005	0.002	0
4	67000	0.009	0.01	0.008	0.013	7.495	0.13	NaN	0.003	0.002	...	0.029	0.009	0.002	

Make a GeoDataFrame for Polyline

```
# preprocessing dataset
od_arr = np.array(od_csv.iloc[:,1:]).astype(float)
od_arr
```

```
array([[1.4647e+01, 3.4700e+00, 5.1400e-01, 2.3000e-02, 1.1000e-02,
        1.6000e-02, 1.6000e-02, 2.9000e-02, 9.0000e-03, 6.0000e-03,
        5.0000e-03, 7.0000e-03, 5.0000e-03, 4.0000e-03, 3.0000e-03,
        6.0000e-03,        nan, 5.7000e-02, 1.8000e-02, 2.0000e-03,
        2.0000e-03,        nan],
       [7.2800e-01, 9.6520e+00, 1.2200e-01, 1.7000e-02, 9.0000e-03,
        1.2000e-02, 1.0000e-02, 1.3000e-02, 5.0000e-03, 4.0000e-03,
        3.0000e-03, 3.0000e-03, 2.0000e-03, 3.0000e-03, 2.0000e-03,
        4.0000e-03,        nan, 2.5000e-02, 1.0000e-02, 2.0000e-03,
               nan,        nan],
       [3.5800e-01, 2.7800e-01, 9.4560e+00, 1.6000e-02, 8.0000e-03,
        1.0000e-02, 5.0000e-03, 1.1100e-01, 1.2000e-02, 5.0000e-03,
        3.0000e-03, 3.0000e-03, 3.0000e-03, 3.0000e-03, 2.0000e-03,
        3.0000e-03,        nan, 5.0000e-03, 4.0000e-02,        nan,
               nan,        nan],
       [1.8000e-02, 2.6000e-02, 2.3000e-02, 1.2742e+01, 1.5000e-02,
```

Make a GeoDataFrame for Polyline

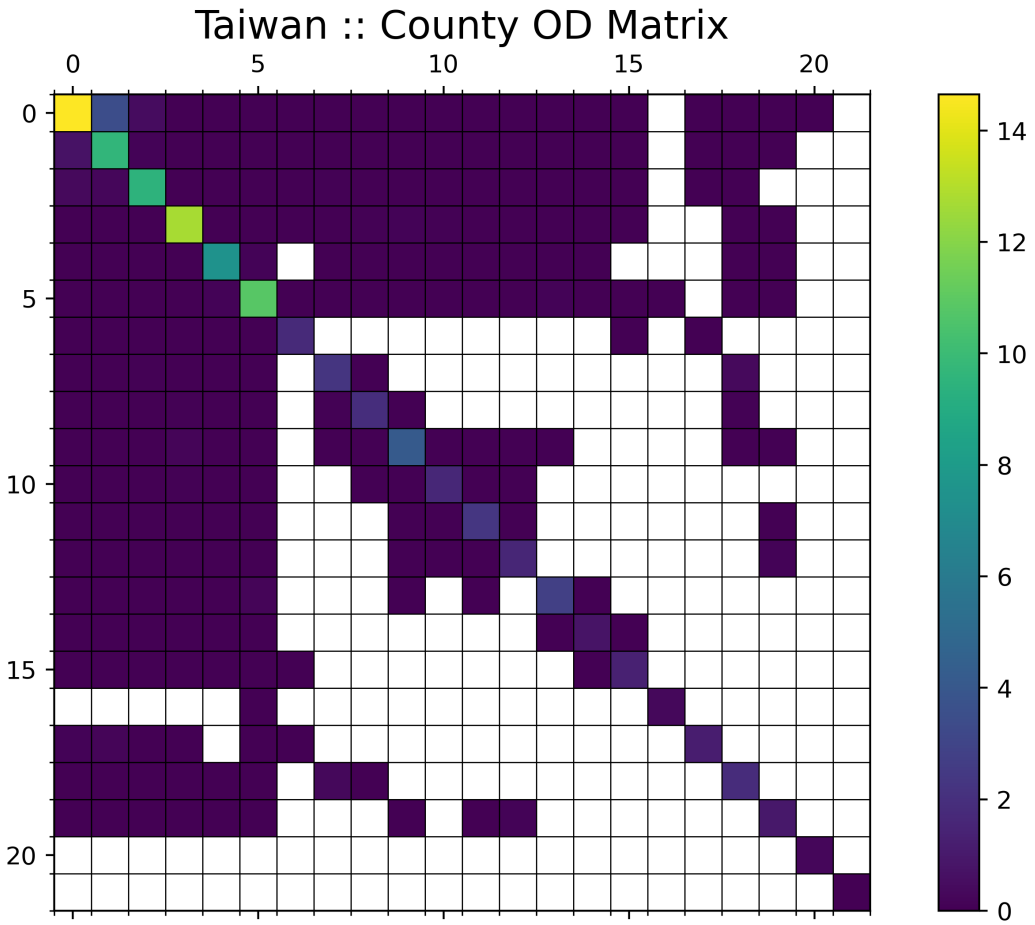
```
# visualize data
fig, ax = plt.subplots(figsize=[10, 6], dpi=400)
mappable = ax.matshow(od_arr, vmin=0)

# plot title
plt.title('Taiwan :: County OD Matrix', fontsize=16)

# plot colorbar
fig.colorbar(mappable)

# get the shape of the array
nrows, ncols = od_arr.shape

# set grid lines at cell boundaries
ax.set_xticks(np.arange(ncols+1)-0.5, minor=True)
ax.set_yticks(np.arange(nrows+1)-0.5, minor=True)
plt.grid(which="minor", color="black", linestyle='-', linewidth=0.5)
plt.show()
```



Make a GeoDataFrame for Polyline

```
# create empty list for transforming from adjacency matrix to edge list
edgelist = []
# fill in the list with row index, column index, and value
for r in range(len(od_arr)):
    for c in range(len(od_arr)):
        edgelist.append([r, c, od_arr[r,c]])
```

```
[[0, 0, 14.647],
 [0, 1, 3.47],
 [0, 2, 0.514],
 [0, 3, 0.023],
 [0, 4, 0.011],
 [0, 5, 0.016],
 [0, 6, 0.016],
 [0, 7, 0.029],
```

Make a GeoDataFrame for Polyline

```
# read country geometry centroid
countyPoint = pd.read_csv('縣市中心點位置.csv')
# preview the dataset
countyPoint.head()
```

	index	COUNTYID	COUNTYCODE	COUNTYNAME	COUNTYENG	x	y	geometry
0	0	F	65000	新北市	New Taipei City	310101.731666	2.764473e+06	POINT (310101.7316655754 2764473.4831959684)
1	1	A	63000	臺北市	Taipei City	305814.553919	2.775076e+06	POINT (305814.5539187362 2775076.2947102515)
2	2	H	68000	桃園市	Taoyuan City	276041.104446	2.755080e+06	POINT (276041.1044456995 2755080.378613185)
3	3	B	66000	臺中市	Taichung City	239057.970052	2.681426e+06	POINT (239057.97005161454 2681426.2470980636)
4	4	D	67000	臺南市	Tainan City	180850.987008	2.561134e+06	POINT (180850.9870075175 2561133.875381154)

Make a GeoDataFrame for Polyline

```
# merge centroid info into OD edgelist
OD_edgelist2 = OD_edgelist.merge(left_on=['O'], right=countyPoint, right_on=countyPoint['index'], how='left')
OD_edgelist2 = OD_edgelist2.merge(left_on=['D'], right=countyPoint, right_on=countyPoint['index'], how='left')
# select important columns
OD_edgelist3 = OD_edgelist2[['O', 'D', 'Value', 'COUNTYNAME_x', 'x_x', 'y_x', 'COUNTYNAME_y', 'x_y', 'y_y']]
# rename columns
OD_edgelist3.columns = ['O', 'D', 'Value', 'COUNTYNAME_O', 'x_O', 'y_O', 'COUNTYNAME_D', 'x_D', 'y_D']
# preview the dataset
OD_edgelist3.head()
```

	O	D	Value	COUNTYNAME_O	x_O	y_O	COUNTYNAME_D	x_D	y_D
0	0	0	14.647	新北市	310101.731666	2.764473e+06	新北市	310101.731666	2.764473e+06
1	0	1	3.470	新北市	310101.731666	2.764473e+06	臺北市	305814.553919	2.775076e+06
2	0	2	0.514	新北市	310101.731666	2.764473e+06	桃園市	276041.104446	2.755080e+06
3	0	3	0.023	新北市	310101.731666	2.764473e+06	臺中市	239057.970052	2.681426e+06
4	0	4	0.011	新北市	310101.731666	2.764473e+06	臺南市	180850.987008	2.561134e+06

Make a GeoDataFrame for Polyline

```
# create an index for binding points to a linestring
OD_edgelist4 = OD_edgelist3.reset_index()
# preview the dataset
OD_edgelist4.head(3)
```

	index	O	D	Value	COUNTYNAME_O	x_O	y_O	COUNTYNAME_D	x_D	y_D
0	0	0	0	14.647	新北市	310101.731666	2.764473e+06	新北市	310101.731666	2.764473e+06
1	1	0	1	3.470	新北市	310101.731666	2.764473e+06	臺北市	305814.553919	2.775076e+06
2	2	0	2	0.514	新北市	310101.731666	2.764473e+06	桃園市	276041.104446	2.755080e+06

Make a GeoDataFrame for Polyline

```
# seperate Origin and Destination information into two DataFrames
O_info = OD_edgelist4[['index', 'O', 'Value', 'COUNTYNAME_O', 'x_O', 'y_O']]
D_info = OD_edgelist4[['index', 'D', 'Value', 'COUNTYNAME_D', 'x_D', 'y_D']]
# rename columns
O_info.columns = ['index', 'location', 'Value', 'COUNTYNAME', 'x', 'y']
D_info.columns = ['index', 'location', 'Value', 'COUNTYNAME', 'x', 'y']
# preview the dataset
O_info.head()
```

Origin

	index	location	Value	COUNTYNAME	x	y
0	0	0	14.647	新北市	310101.731666	2.764473e+06
1	1	0	3.470	新北市	310101.731666	2.764473e+06
2	2	0	0.514	新北市	310101.731666	2.764473e+06

Destination

	index	location	Value	COUNTYNAME	x	y
0	0	0	14.647	新北市	310101.731666	2.764473e+06
1	1	1	3.470	臺北市	305814.553919	2.775076e+06
2	2	2	0.514	桃園市	276041.104446	2.755080e+06

Make a GeoDataFrame for Polyline

```
# concatenate Origin and Destination DataFrame into a single DataFrame
OD_info = pd.concat([O_info, D_info], axis=0).sort_index()
# preview the dataset
OD_info.head(4)
```

	index	location	Value	COUNTYNAME	x	y
0	0	0	14.647	新北市	310101.731666	2.764473e+06
0	0	0	14.647	新北市	310101.731666	2.764473e+06
1	1	0	3.470	新北市	310101.731666	2.764473e+06
1	1	1	3.470	臺北市	305814.553919	2.775076e+06

Make a GeoDataFrame for Polyline

```
# import package
from shapely.geometry import Point, LineString, shape
# zip the coordinates into a point object and convert to a GeoDataFrame
geometry = [Point(xy) for xy in zip(OD_info.x, OD_info.y)]
# convert to GeoDataFrame
gOD = gpd.GeoDataFrame(OD_info, geometry=geometry)
# combine POINT to POLYLINE
gOD2 = gOD.groupby(['index', 'Value'])['geometry'].apply(lambda x: LineString(x.tolist()))
# convert to GeoDataFrame with polyline-format geometry
gOD2 = gpd.GeoDataFrame(gOD2, geometry='geometry').reset_index()
# preview the dataset
gOD2.head(2)
```

	index	Value	geometry
0	0	14.647	LINESTRING (310101.732 2764473.483, 310101.732...
1	1	3.470	LINESTRING (310101.732 2764473.483, 305814.554...

Basic Preprocessing in GeoPandas

- GeoPandas extends the functionality of Pandas, offering most of its powerful data manipulation features while adding specialized support for spatial data. This means you can directly preprocess, filter, aggregate, and transform geographic datasets using familiar Pandas operations on **GeoDataFrame** and **GeoSeries** objects.
- With GeoPandas, you can seamlessly handle both attribute and spatial data, making tasks such as coordinate transformations, spatial joins, and geometric operations more intuitive. Whether you're working with shapefiles, GeoJSON, or spatial databases, GeoPandas allows for efficient preprocessing, enabling smoother workflows for visualization, spatial analysis, and machine learning applications.

Basic Preprocessing in GeoPandas

Merge, Concatenation, and Filtration

- **Task: Merge population into CODEBASE zones**

```
# read population data
pop = pd.read_csv('112年12月臺北市統計區人口統計_最小統計區.csv',
                  skiprows=1)
# rename columns
pop.columns = ['CODE2', 'CODE1', 'CODEBASE', 'Households', 'POP',
               'M_POP', 'F_POP', 'Date']
# preview the dataset
pop.head(2)
```

	CODE2	CODE1	CODEBASE	Households	POP	M_POP	F_POP	Date
0	A6301-01	A6301-01-001	A6301-0003-00	13	22	9	13	112Y12M
1	A6301-01	A6301-01-001	A6301-0004-00	0	0	0	0	112Y12M

Basic Preprocessing in GeoPandas

Merge, Concatenation, and Filtration

```
# merge population into CODEBASE data
cb = codebase.merge(left_on=['CODEBASE', 'CODE1', 'CODE2'],
                    right=pop, right_on=['CODEBASE', 'CODE1', 'CODE2'],
                    how='left')
# select important columns
cb = cb[['U_ID', 'CODEBASE', 'CODE1', 'CODE2', 'TOWN_ID', 'TOWN',
         'COUNTY_ID', 'COUNTY', 'Households', 'POP', 'X', 'Y',
         'M_POP', 'F_POP', 'AREA', 'geometry']]
# preview the dataset
cb.head(2)
```

	U_ID	CODEBASE	CODE1	CODE2	TOWN_ID	TOWN	COUNTY_ID	COUNTY	Households	POP	X	Y	M_POP	F_POP	AREA	geometry
0	1972	A6310-0024-00	A6310-01-006	A6310-01	63000100	內湖區	63000	臺北市	11	33	310951.84060	2.775913e+06	15	18	362112.90727	POLYGON Z ((310568.684 2775912.720 0.000, 3105...
1	1973	A6311-0791-00	A6311-61-005	A6311-61	63000110	士林區	63000	臺北市	10	19	300103.56005	2.776286e+06	8	11	17541.31869	POLYGON Z ((299986.213 2776330.686 0.000, 3000...

Map Visualization

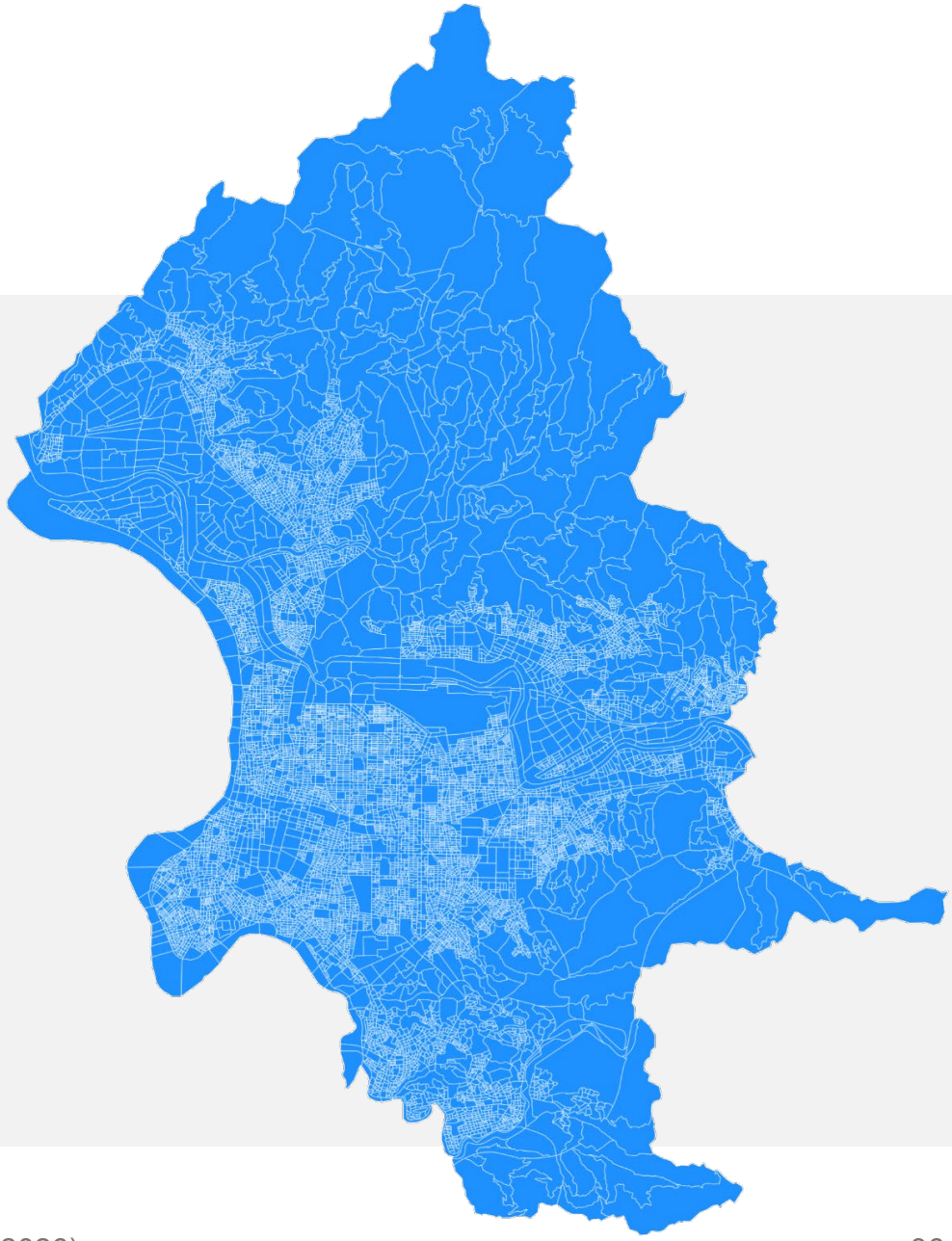
- Map visualization is a fundamental aspect of geospatial data science. A well-crafted map not only reveals spatial patterns and relationships but also strengthens visual evidence and enhances the storytelling power of your analysis. Effective map visualizations can help convey complex geographic information in an intuitive and accessible way, making them essential in fields such as urban planning, environmental analysis, transportation modeling, and epidemiology.

Map Visualization

- GeoPandas, offers intuitive tools for creating static map visualizations. Similar to popular GIS software like ArcGIS and QGIS, GeoPandas provides built-in functions for rendering spatial data, allowing users to explore, analyze, and present geographic patterns effectively.
- In addition to its native plotting capabilities, GeoPandas seamlessly integrates with Matplotlib, enabling users to customize maps with annotations, color schemes, legends, and overlays. This flexibility makes GeoPandas an excellent choice for producing high-quality static maps for reports, presentations, or exploratory data analysis.
- In this guide, we will explore how to visualize spatial data using GeoPandas, including techniques to enhance your maps with Matplotlib to improve clarity, readability, and impact.

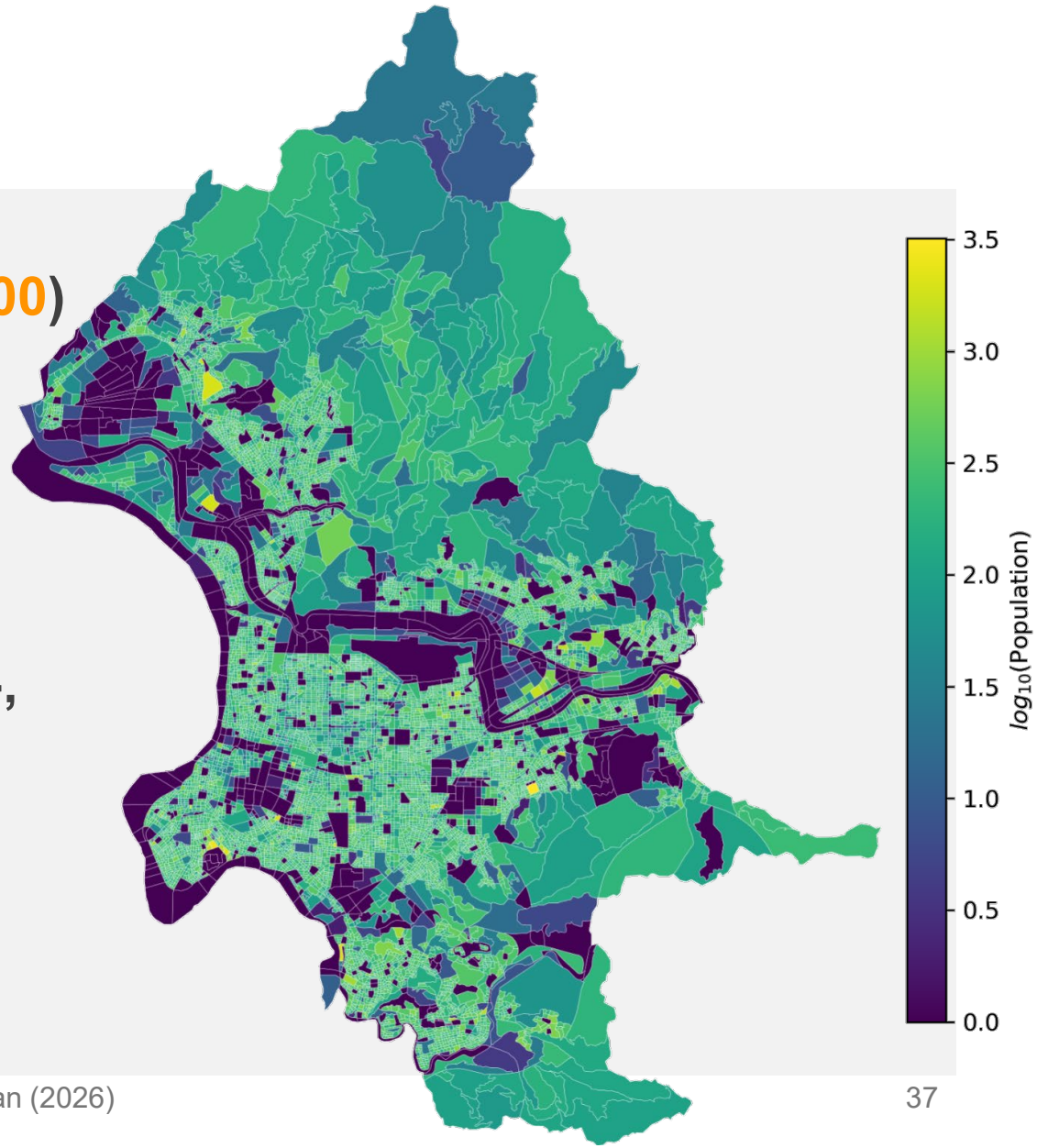
Map Visualization

```
# figure settings
fig, ax = plt.subplots(figsize=[8,12],
                       dpi=300)
# plot CODEBASE map
cb.plot(fc='dodgerblue', ec='w',
        linewidth=0.2, ax=ax)
# off axis
plt.axis('off')
# show map
plt.show()
```



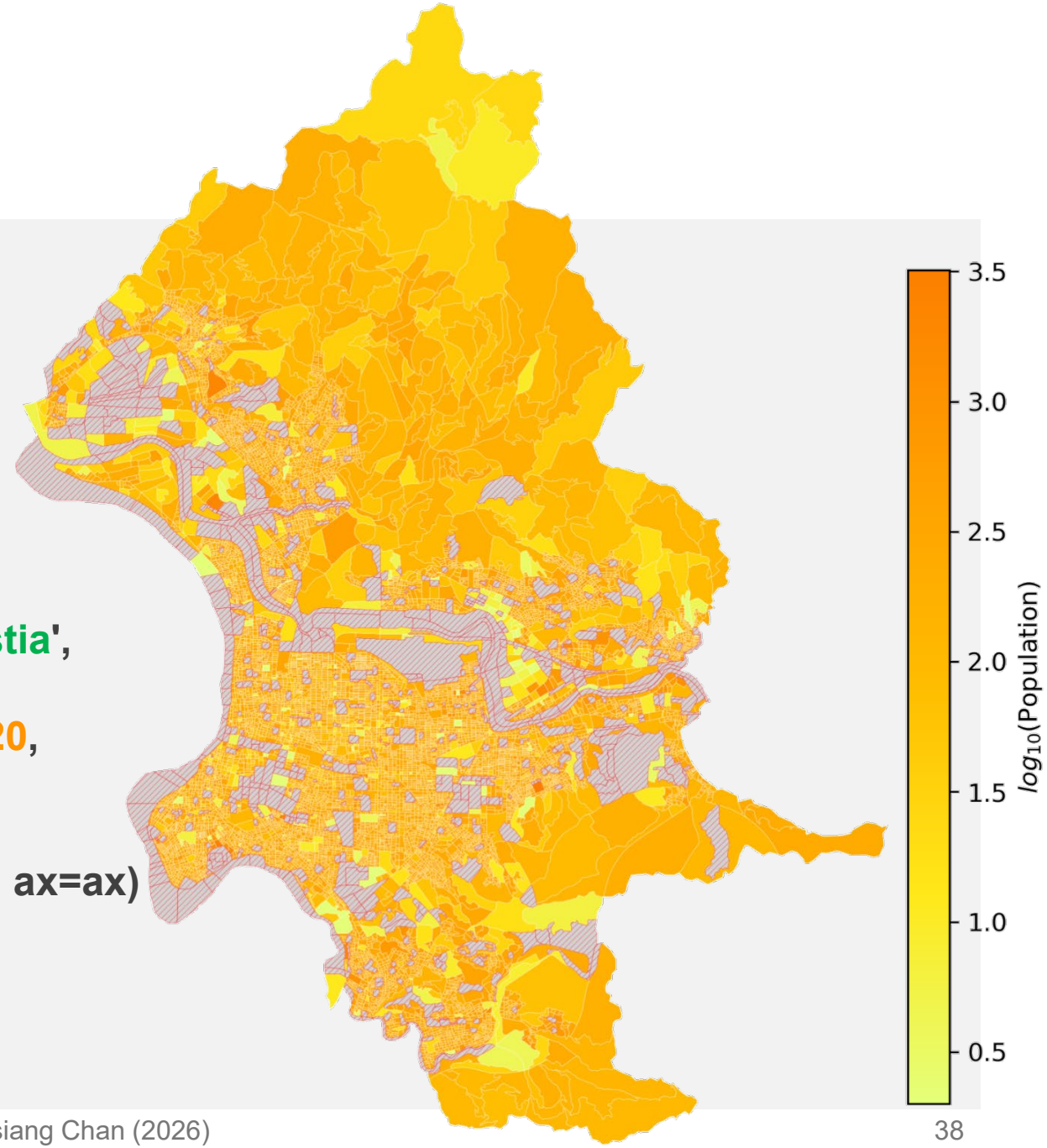
Map Visualization

```
# set figure
fig, ax = plt.subplots(figsize=[8,12],dpi=300)
# plot population data
cb.plot(column='log_pop', ec='w',
            linewidth=0.1, legend=True,
            legend_kwds={'shrink': 0.5,
            'aspect': 20,
            'label':r'$log_{10}$(Population)'},
            ax=ax)
# off axis
plt.axis('off')
# show map
plt.show()
```



Map Visualization

```
# set nan
cb.loc[cb['log_pop']==0, 'log_pop'] = np.nan
# import packages
import matplotlib as mpl
mpl.rcParams['hatch.linewidth'] = 0.1
# set figure
fig, ax = plt.subplots(figsize=[8,12],dpi=300)
# plot population data
cb.plot(column='log_pop', ec='w', cmap='Wistia',
        linewidth=0.1, legend=True,
        legend_kwds={'shrink': 0.5, 'aspect': 20,
        'label':r'$log_{10}$(Population)'},
        missing_kwds={"color": "lightgrey",
        "edgecolor": "red","hatch": "///////////"}, ax=ax)
# off axis
plt.axis('off')
# show map
plt.show()
```



GeoProcessing :: Fundamentals

- Geopandas offers several simple but useful functions similar to those with the same name in ArcGIS. Therefore, if you are unfamiliar with the following functions, please refer to my Geographic Information System slides.

Methods	Functionality
clip	Clip points, lines, or polygon geometries to the mask extent. Both layers must be in the same Coordinate Reference System (CRS).
dissolve	Dissolve geometries within *groupby* into single observation. This is accomplished by applying the *union_all* method to all geometries within a groupself. Observations associated with each *groupby* group will be aggregated using the *aggfunc*.
sjoin	Spatial join of two GeoDataFrames.
overlay	Currently only supports data GeoDataFrames with uniform geometry types, i.e. containing only (Multi)Polygons, or only (Multi)Points, or a combination of (Multi)LineString and LinearRing shapes.

GeoProcessing :: Clip

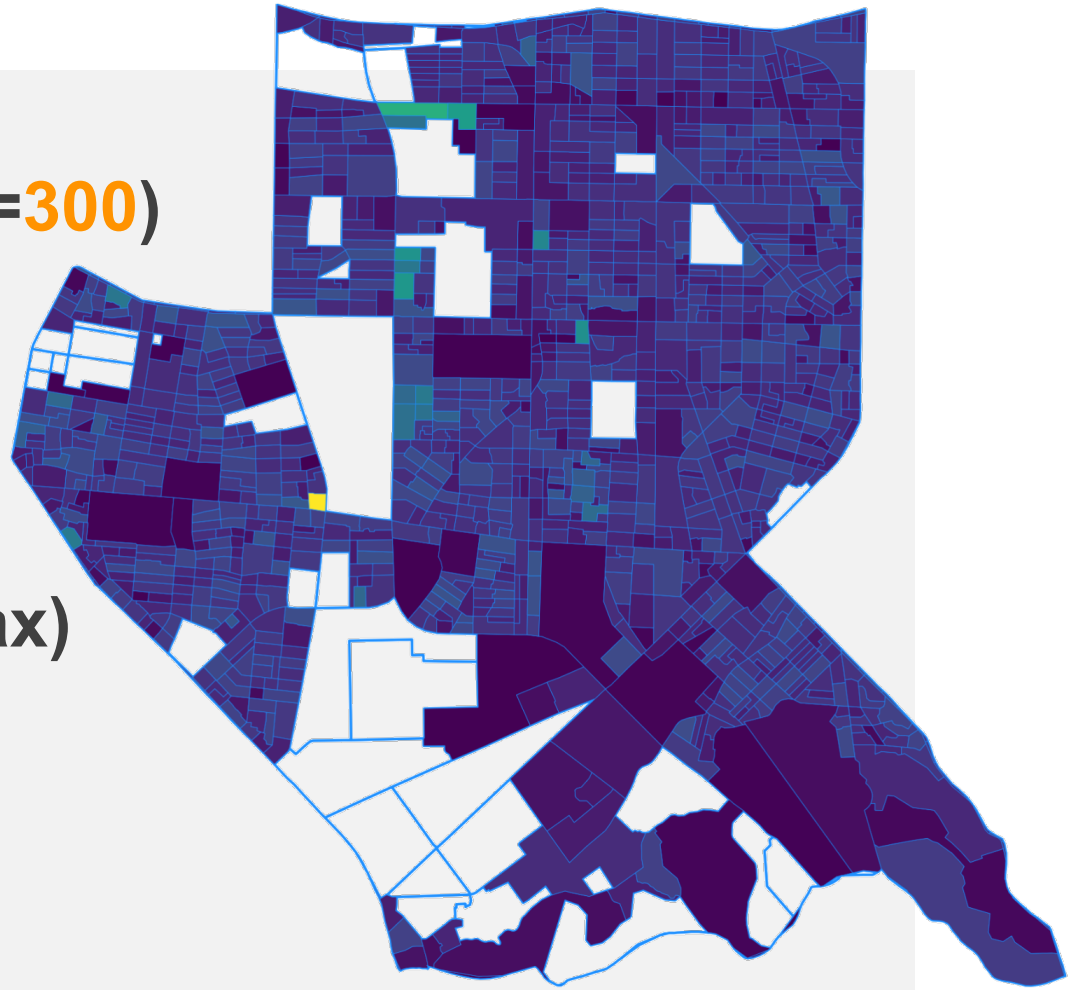
```
# extract Da'An district
daan = town[(town['COUNTYNAME']=='臺北市') &
            (town['TOWNNAME']=='大安區')].reset_index(drop=True)
# preview result
daan
```

	TOWNID	TOWNCODE	COUNTYNAME	TOWNNAME	TOWNENG	COUNTYID	COUNTYCODE	geometry
0	A02	63000030	臺北市	大安區	Da'an District	A	63000	POLYGON ((304876.765 2770861.505, 304900.207 2...

```
# clip CODEBASE with Da'An district
cb_daan = gpd.clip(cb, daan)
```

GeoProcessing :: Clip

```
# set figure
fig, ax = plt.subplots(figsize=[8,12],dpi=300)
# plot population data
cb_daan.plot(fc='w', ec='dodgerblue',
    ax=ax)
cb_daan.plot(column='POP',
    ec='dodgerblue', linewidth=0.2, ax=ax)
# off axis
plt.axis('off')
# show map
plt.show()
```



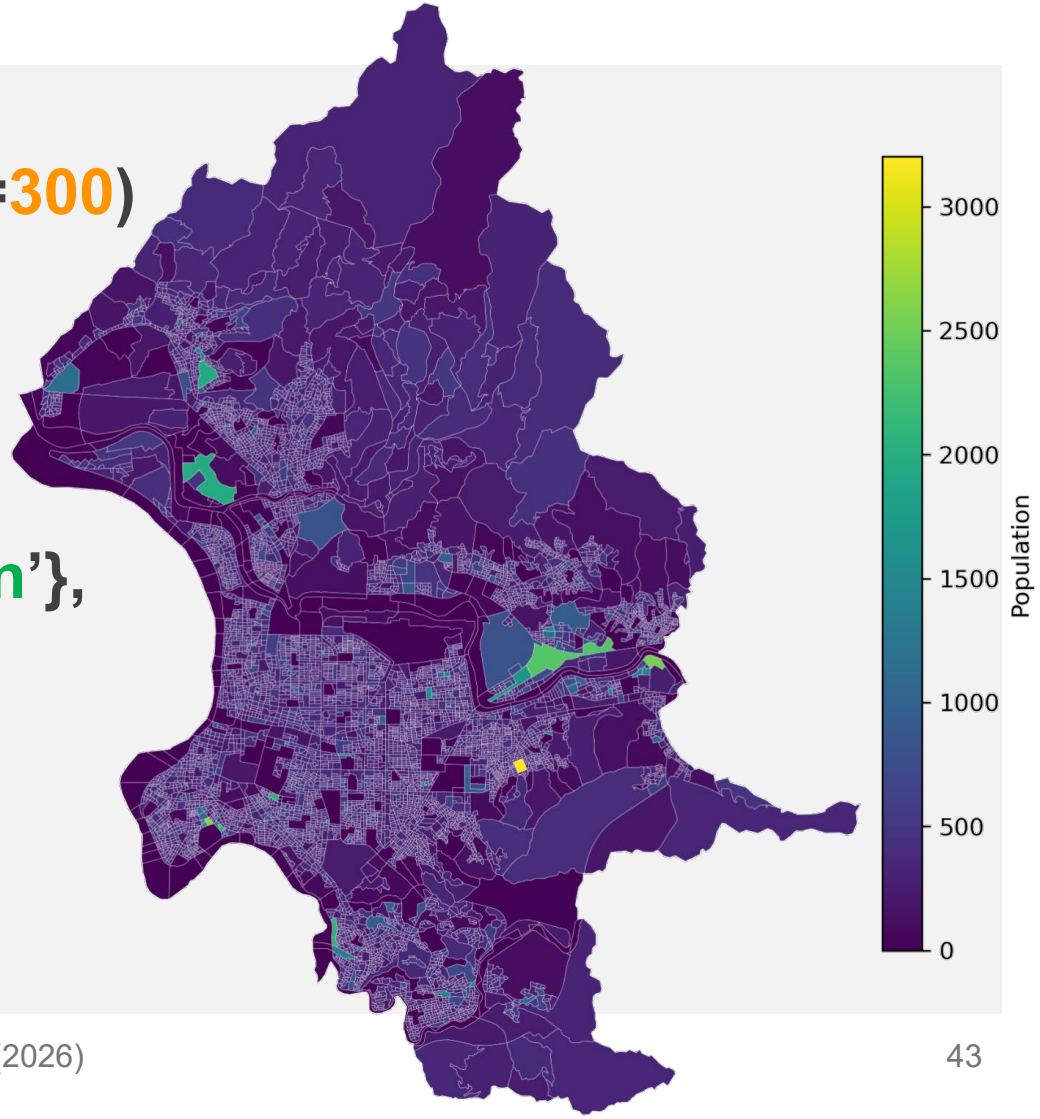
GeoProcessing :: Dissolve

```
# dissolve by CODE1
cb1 = cb.dissolve(by='CODE1', aggfunc={"Households":"sum",
                    "POP":"sum","M_POP":"sum","F_POP":"sum",
                    "log_pop":"sum","X":"mean","Y":"mean","ARE
                    A":"sum"}).reset_index(drop=True)
# select important columns
cb1 = cb1[['Households', 'POP', 'X', 'Y', 'M_POP', 'F_POP',
           'AREA', 'log_pop', 'geometry']]
# show results
cb1
```

	Households	POP	X	Y	M_POP	F_POP	AREA	log_pop	geometry
0	35	68.0	306723.98703	2.773600e+06	35	33	585896.25050	4.561483	POLYGON Z ((307356.325 2772974.783 0.000, 3073...
1	45	112.0	307070.50946	2.773329e+06	62	50	83428.10199	2.053078	POLYGON Z ((307247.391 2773495.672 0.000, 3072...
2	160	412.0	307176.11333	2.773214e+06	206	206	8678.32793	2.615950	POLYGON Z ((307289.353 2773209.310 0.000, 3070...

GeoProcessing :: Dissolve

```
# set figure
fig, ax = plt.subplots(figsize=[8,12],dpi=300)
# plot population data
cb1.plot(column='log_pop', ec='w',
         linewidth=0.1, legend=True,
         legend_kwds={'shrink': 0.5,
         'aspect': 20, 'label':'Population'},
         ax=ax)
# off axis
plt.axis('off')
# show map
plt.show()
```



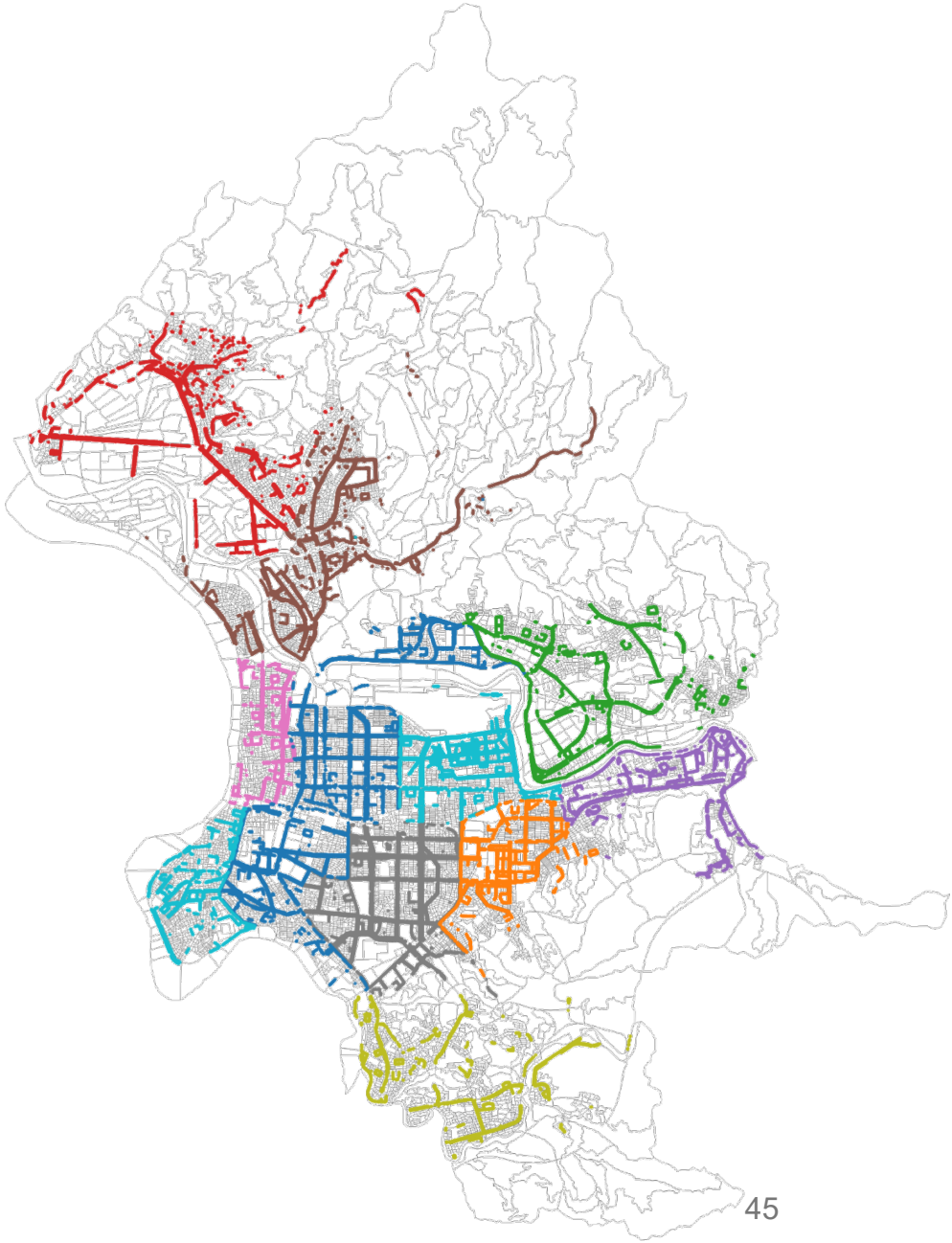
GeoProcessing :: Sjoin

```
# sjoin CODEBASE into Tree
cbTree = gpd.sjoin(tree, cb, how='left')
# preview the dataset
cbTree.head(4)
```

	TreeID	Dist	Region	RegionRemark	TreeType	Diameter	TreeHeight	SurveyDate	TWD97X	TWD97Y	...	COUNTY_ID	COUNTY	Households
0	BT0614021096	士林區	文林路	東側人行道	榕樹	54	16	2022-11-18	302404.92	2777358.41	...	63000	臺北市	134.0
1	BT0614021097	士林區	文林路	東側人行道	白千層	45	6.8	2022-11-18	302408.24	2777352.36	...	63000	臺北市	134.0
2	BT0614021098	士林區	文林路	東側人行道	茄苳	6	8.7	2022-11-18	302413.58	2777342.69	...	63000	臺北市	134.0
3	BT0614031095	士林區	文林路	西側人行道	白千層	22	7.7	2022-11-18	302369.5	2777349.52	...	63000	臺北市	247.0

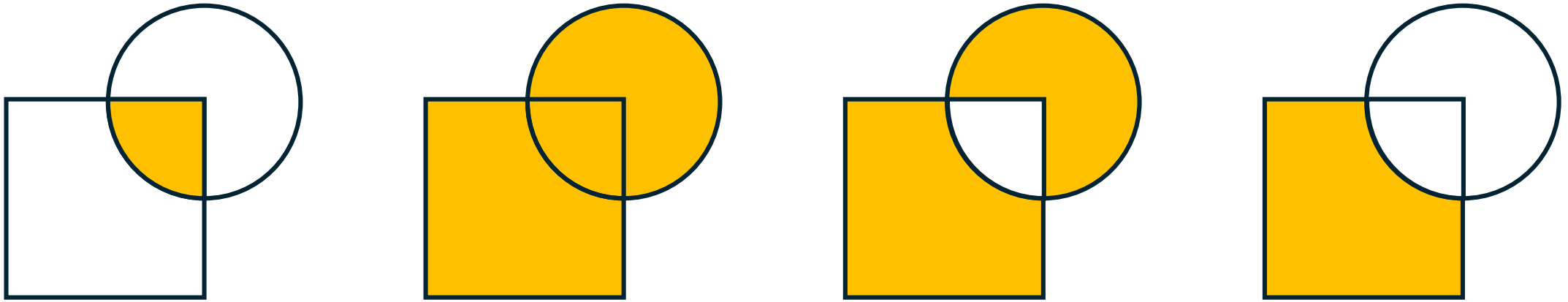
GeoProcessing :: Sjoin

```
# set figure
fig, ax = plt.subplots(figsize=[0,12],dpi=300)
# plot tree data
codebase.plot(fc='w',
                    edgecolor=(0.2,0.2,0.2,1),
                    linewidth=0.1, ax=ax)
cbTree.plot(column='Dist', s=0.2, ax=ax)
# off axis
plt.axis('off')
# show map
plt.show()
```



GeoProcessing :: Overlay

Overlay analysis in **GeoPandas** is a powerful spatial operation that allows users to combine multiple geospatial datasets based on their geometric relationships.



Intersection | Union | Symmetric Difference | Difference

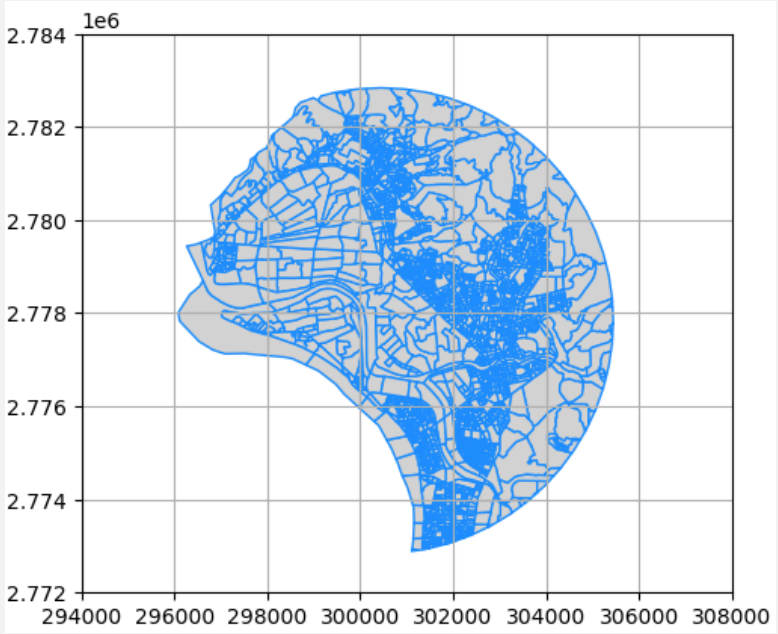
GeoProcessing :: Overlay

```
# read Beitou Refuse Incineration with 1500m buffer
b5000 = gpd.read_file('Beitou_5000m.shp')
# preview the dataset
b5000
```

	Location	geometry
0	Taipei 101	POLYGON ((308501.200 2769627.481, 308493.977 2...

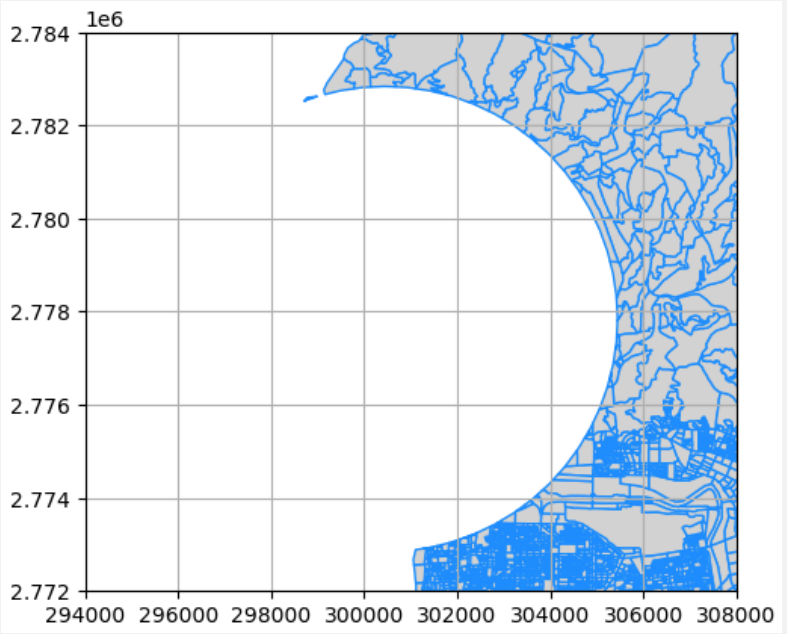
GeoProcessing :: Overlay
Intersection

```
# overlay analysis :: intersection
o1 = gpd.overlay(codebase, b5000, how='intersection')
# plot result
o1.plot(fc='lightgray', ec='dodgerblue')
plt.axis([294000, 308000,
          2772000, 2784000])
plt.grid(True)
# show map
plt.show()
```



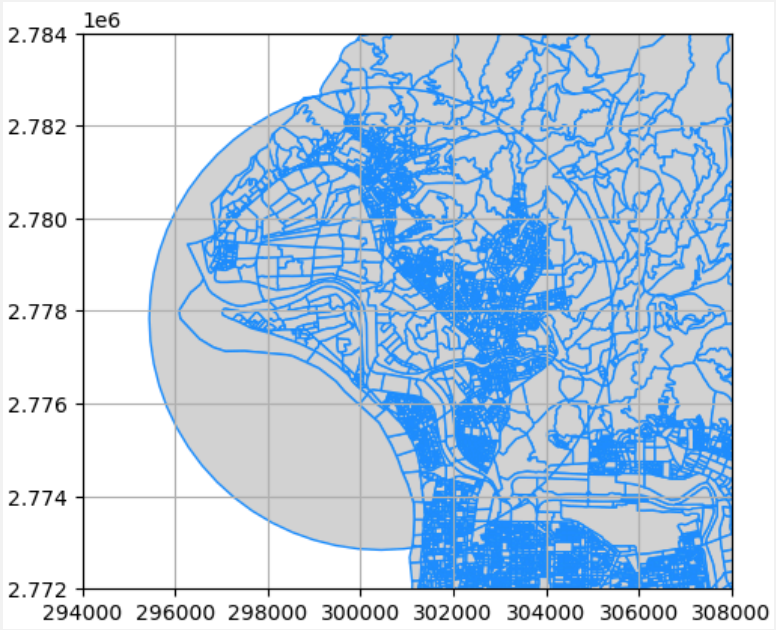
GeoProcessing :: Overlay
Difference

```
# overlay analysis :: difference
o1 = gpd.overlay(codebase, b5000, how='difference')
# plot result
o1.plot(fc='lightgray', ec='dodgerblue')
plt.axis([294000, 308000,
          2772000, 2784000])
plt.grid(True)
# show map
plt.show()
```



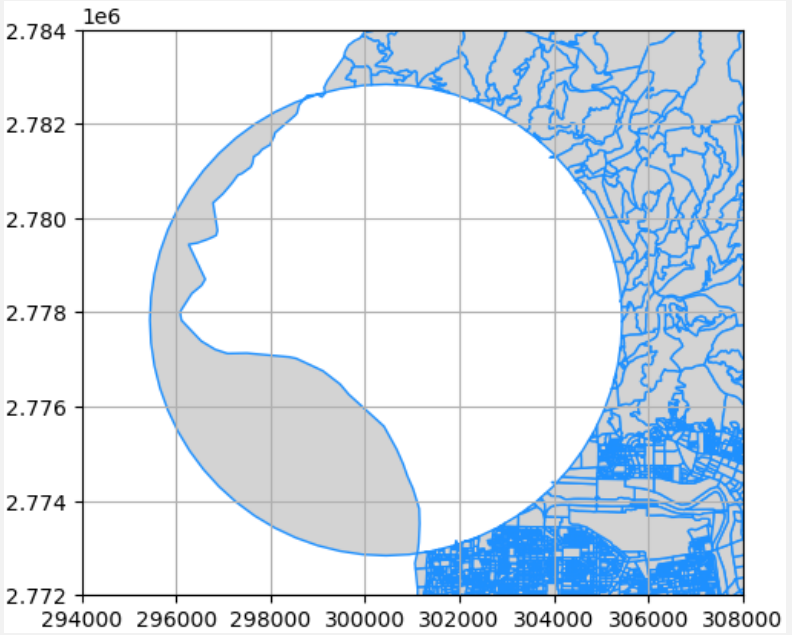
GeoProcessing :: Overlay
Union

```
# overlay analysis :: union
o1 = gpd.overlay(codebase, b5000, how='union')
# plot result
o1.plot(fc='lightgray', ec='dodgerblue')
plt.axis([294000, 308000,
          2772000, 2784000])
plt.grid(True)
# show map
plt.show()
```



GeoProcessing :: Overlay
Symmetric Difference

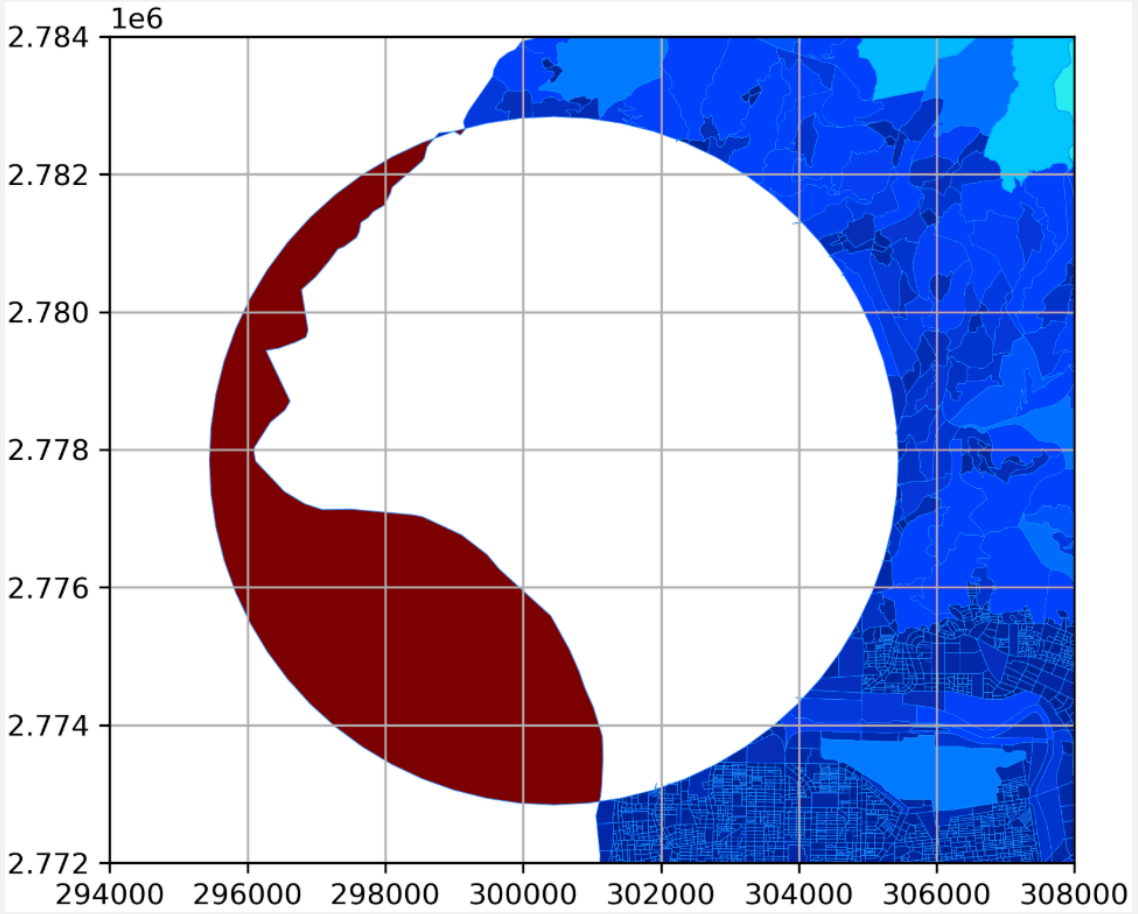
```
# overlay analysis :: symmetric_difference
o1 = gpd.overlay(codebase, b5000,
                 how='symmetric_difference')
# plot result
o1.plot(fc='lightgray', ec='dodgerblue')
plt.axis([294000, 308000,
          2772000, 2784000])
plt.grid(True)
# show map
plt.show()
```



GeoProcessing :: Shape
Length

```
# calculate length of the shape
o4['length'] = o4.length
o4['length']
```

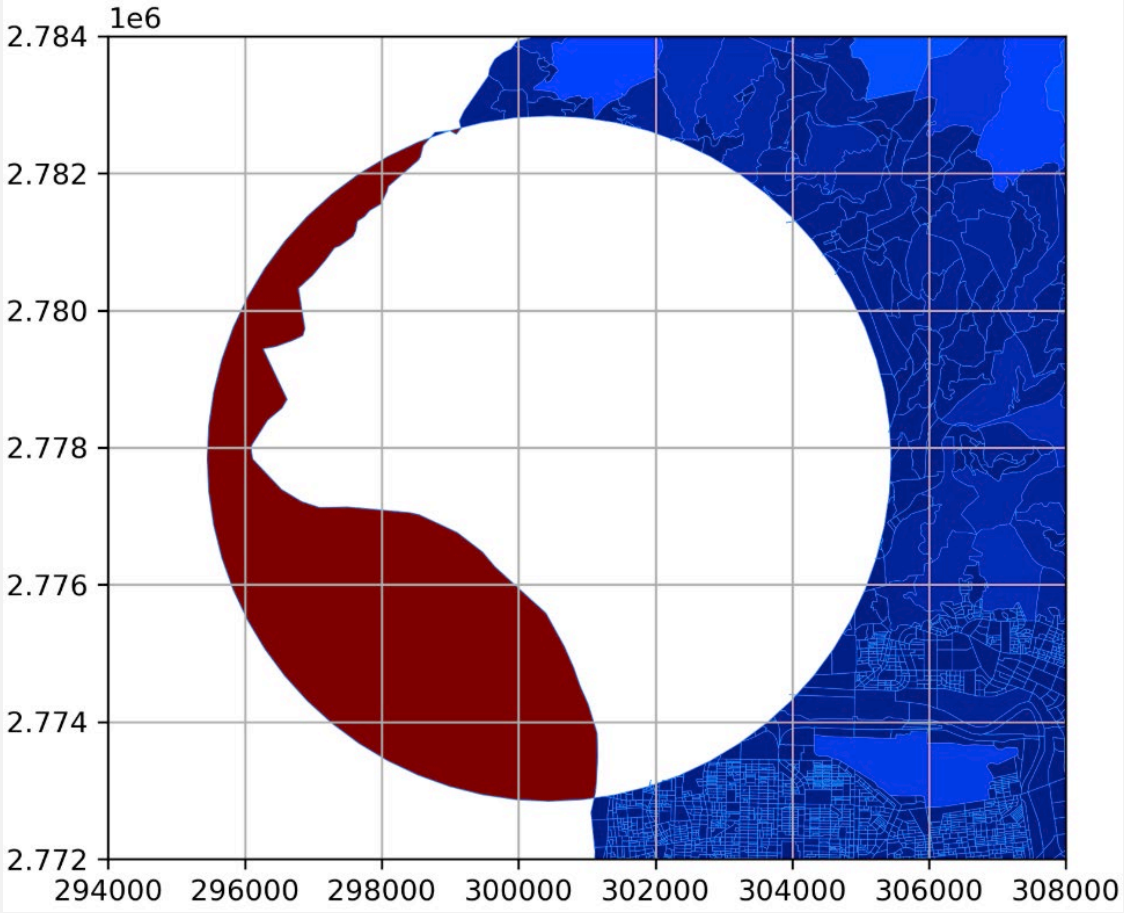
```
0        3126.625671
1         251.983909
2         205.200011
3         281.487492
4         409.138173
             ...
8956      398.964607
8957      291.388624
8958      354.013871
8959     2016.512301
8960    30307.667192
Name: length, Length: 8961, dtype: float64
```



GeoProcessing :: Shape
Area

```
# calculate length of the shape
o4['area'] = o4.area
o4['area']
```

```
0       3.621129e+05
1       2.457990e+03
2       2.599756e+03
3       2.433966e+03
4       9.721254e+03
             ...
8956    6.921591e+03
8957    3.357220e+03
8958    3.858130e+03
8959    9.681873e+04
8960    2.030747e+07
Name: area, Length: 8961, dtype: float64
```



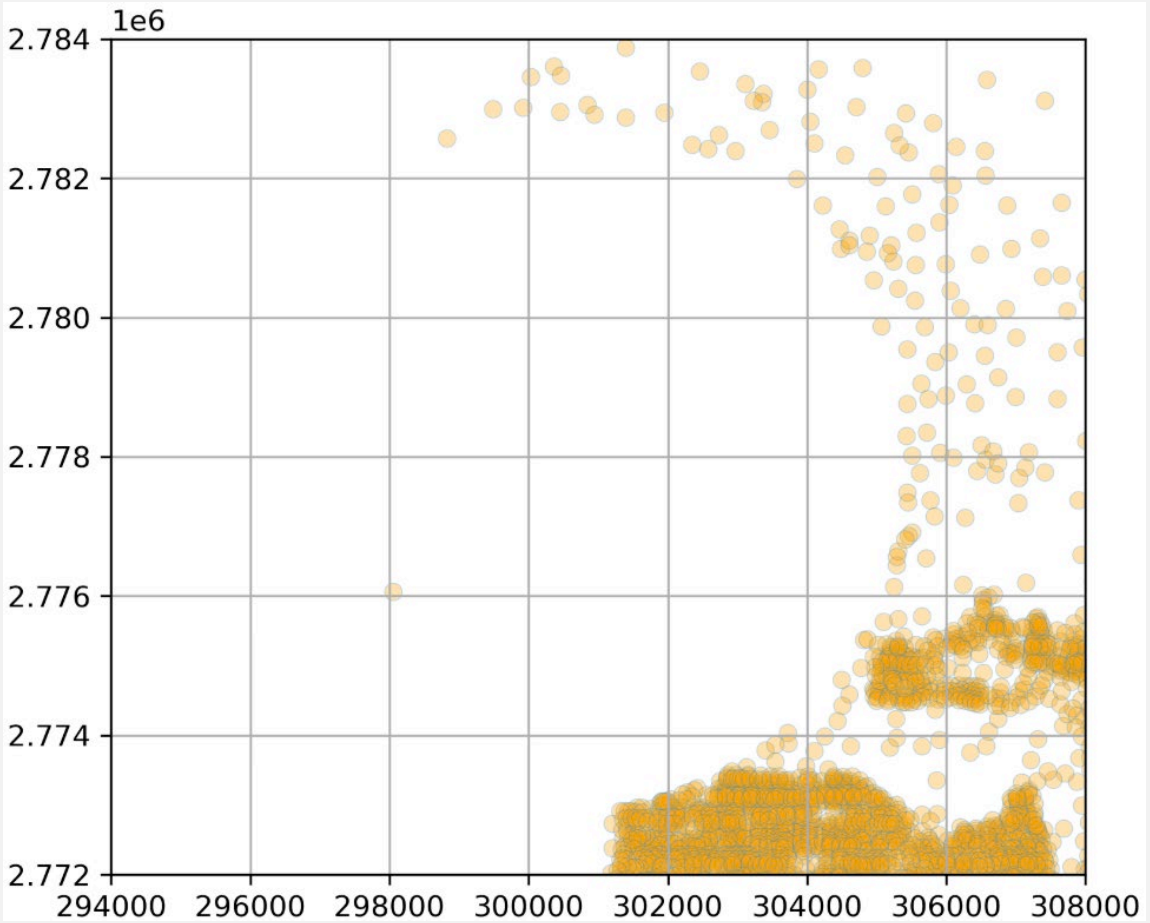
GeoProcessing :: Shape
Centriod

```
# generate the centriod of each polygon and to DataFrame
o5_centroid = o4.centroid.to_frame()
# convert to GeoDataFrame
o5_centroid = gpd.GeoDataFrame(o5_centroid, geometry=
                    o5_centroid[0], crs=3826)
# delete redundant column
o5_centroid = o5_centroid[['geometry']]
# preview data
o5_centroid
```

GeoProcessing :: Shape
Centriod

	geometry
0	POINT (310982.911 2775890.695)
1	POINT (308916.356 2775828.326)
2	POINT (310485.396 2775817.856)
3	POINT (308910.854 2775807.425)
4	POINT (310729.994 2775502.169)
...	...
8956	POINT (301527.748 2771696.512)
8957	POINT (306365.507 2771755.623)
8958	POINT (305678.956 2771754.059)
8959	POINT (308027.507 2771629.698)
8960	POINT (298053.767 2776062.122)

8961 rows × 1 columns



GeoProcessing :: Shape
Boundary

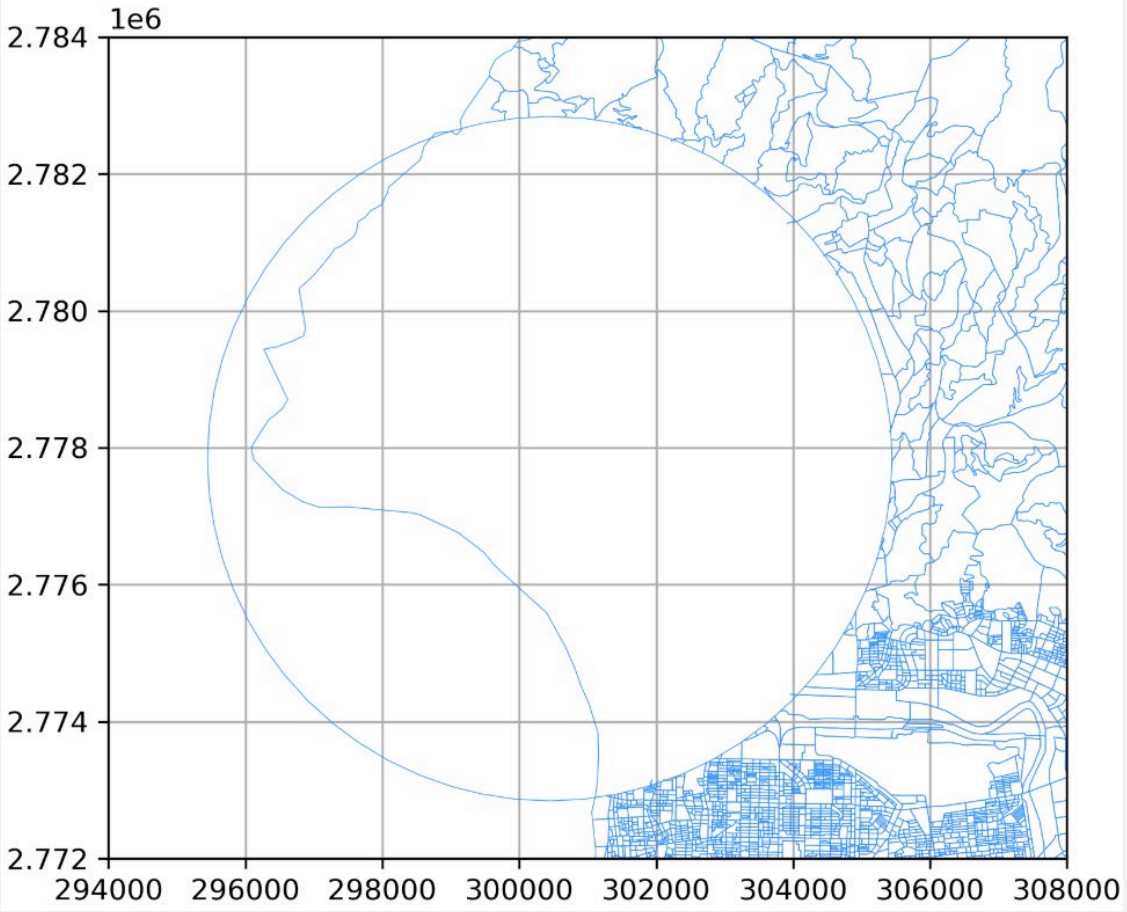
```
# generate the centriod of each polygon and to DataFrame
o5_boundary = o4.boundary.to_frame()
# convert to GeoDataFrame
o5_boundary = gpd.GeoDataFrame(o5_boundary, geometry=
                    o5_boundary[0], crs=3826)
# delete redundant column
o5_boundary = o5_boundary[['geometry']]
# preview data
o5_boundary
```

GeoProcessing :: Shape

Boundary

	geometry
0	LINESTRING Z (310568.684 2775912.72 0, 310582....
1	LINESTRING Z (308879.33 2775859.466 0, 308954....
2	LINESTRING Z (310471.413 2775851.529 0, 310502...
3	LINESTRING Z (308864.903 2775843.025 0, 308900...
4	LINESTRING Z (310770.337 2775554.824 0, 310806...
...	...
8956	LINESTRING Z (301488.645 2771771.247 0, 301525...
8957	LINESTRING Z (306306.877 2771771.101 0, 306367...
8958	LINESTRING Z (305609.096 2771768.084 0, 305720...
8959	LINESTRING Z (307752.66 2771709.18 0, 307763.1...
8960	MULTILINESTRING Z ((301940.599 2773069.524 0, ...

8961 rows × 1 columns



GeoProcessing :: Shape
Bounds

```
# generate the centriod of each polygon and to DataFrame
o5_boundary = o4.bounds
# preview data
o5_boundary
```

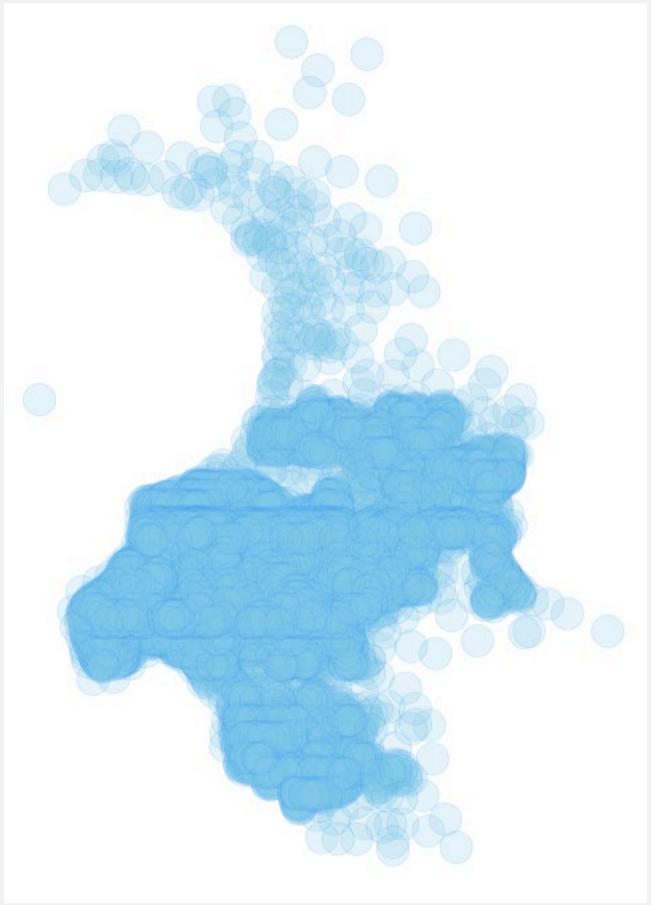
	minx	miny	maxx	maxy
0	310551.113304	2.775497e+06	311290.426752	2.776375e+06
1	308864.903459	2.775799e+06	308970.536484	2.775859e+06
2	310454.196070	2.775786e+06	310519.369402	2.775852e+06
3	308851.148383	2.775779e+06	308967.850445	2.775843e+06
4	310673.306625	2.775432e+06	310806.070273	2.775555e+06
...	...	...	...	...
8956	301488.645279	2.771638e+06	301568.607486	2.771771e+06
8957	306306.877437	2.771740e+06	306424.930301	2.771771e+06
8958	305608.727548	2.771740e+06	305758.027017	2.771771e+06
8959	307615.262665	2.771522e+06	308514.728858	2.771770e+06
8960	295446.353568	2.772839e+06	305425.342596	2.782664e+06

8961 rows × 4 columns

GeoProcessing :: Buffer

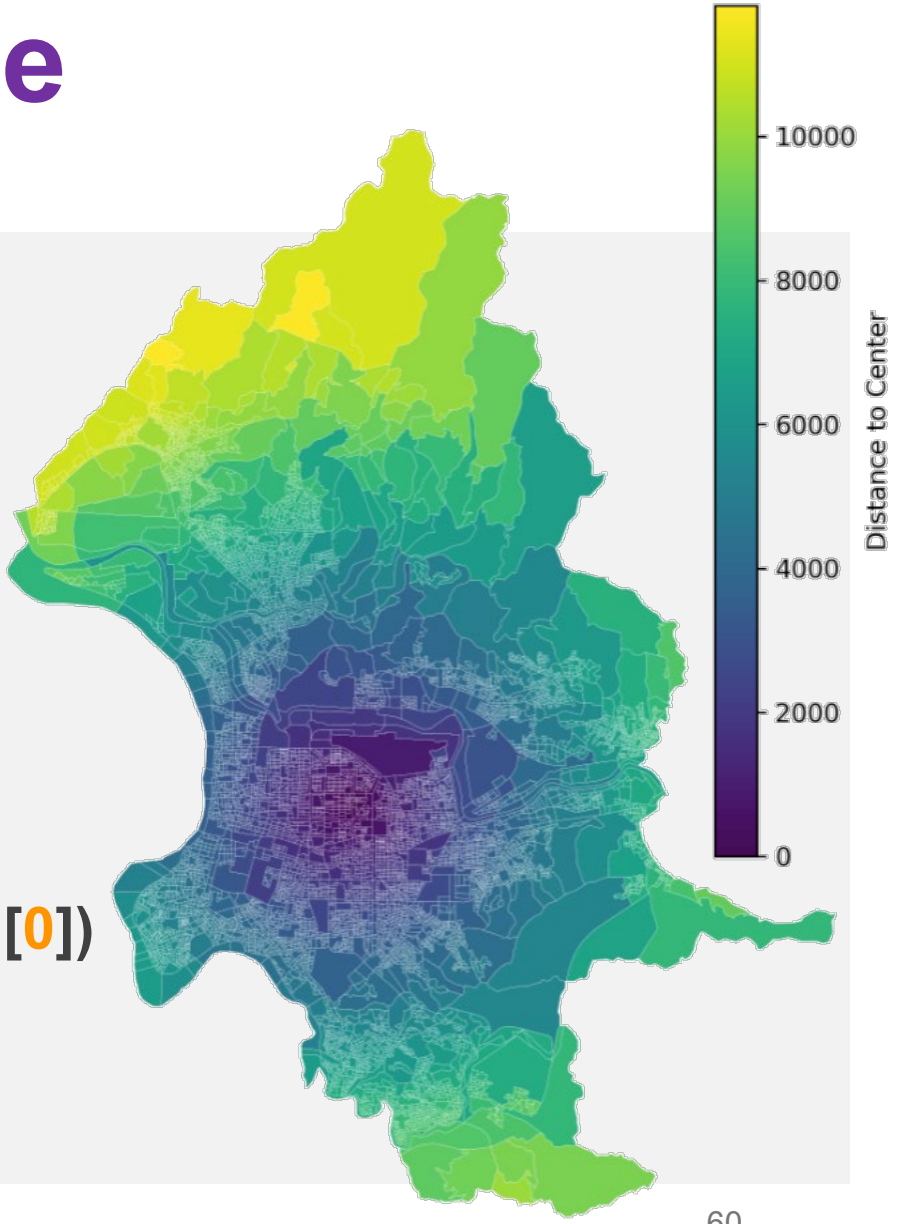
Buffer

```
# buffer with 500m
buffer500m = o5_centroid.buffer(500)
# plot result
fig, ax = plt.subplots(figsize=[12,6], dpi=300)
buffer500m.plot(fc='skyblue', ec='dodgerblue',
                linewidth=0.3, alpha=0.3, ax=ax)
plt.grid(True)
plt.axis('off')
# show map
plt.show()
```



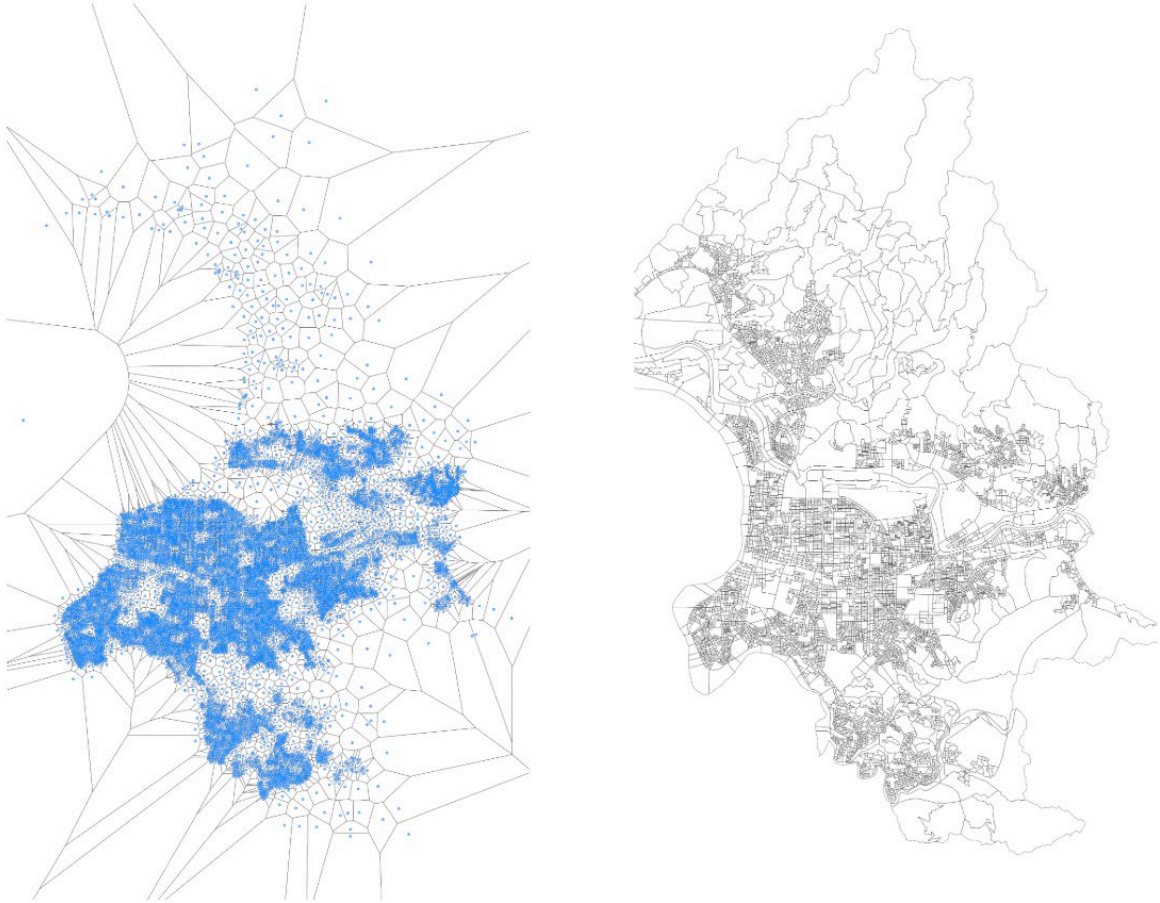
GeoProcessing :: Distance

```
# get the center of taipei
x_ = np.mean(cb1.centroid.x)
y_ = np.mean(cb1.centroid.y)
# convert to Point
s = gpd.GeoSeries(Point([x_, y_]), crs=3826)
# compute distance
dist = []
for i in range(cb1.shape[0]):
    dist.append(s.distance(cb1['geometry'][i]).values[0])
# insert to GeoDataFrame
cb1['dist'] = dist
```



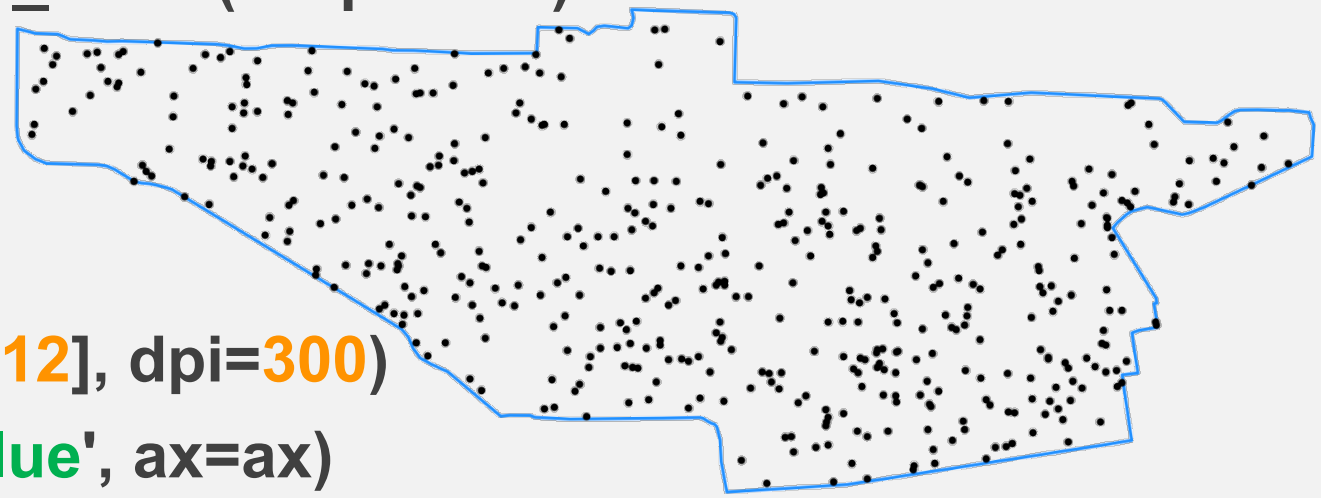
GeoProcessing :: Voronoi Polygon

```
# compute Voronoi polygon
voronoi = o5_centroid.voronoi_polygons()
# set figure
fig, ax = plt.subplots(1, 2, figsize=[10,12],dpi=300)
# plot population data
voronoi.plot(fc='w', ec='k', linewidth=0.1, ax=ax[0])
                o5_centroid.plot(fc='dodgerblue',
                markersize=0.1, ax=ax[0])
# off axis
ax[0].axis([297500, 315000, 2760000, 2790000])
ax[0].axis('off')
# plot population data
cb1.plot(fc='w', ec='k', linewidth=0.1, ax=ax[1])
# off axis
ax[1].axis([297500, 315000, 2760000, 2790000])
ax[1].axis('off')
# show map
plt.show()
```



GeoProcessing :: Random Sampling

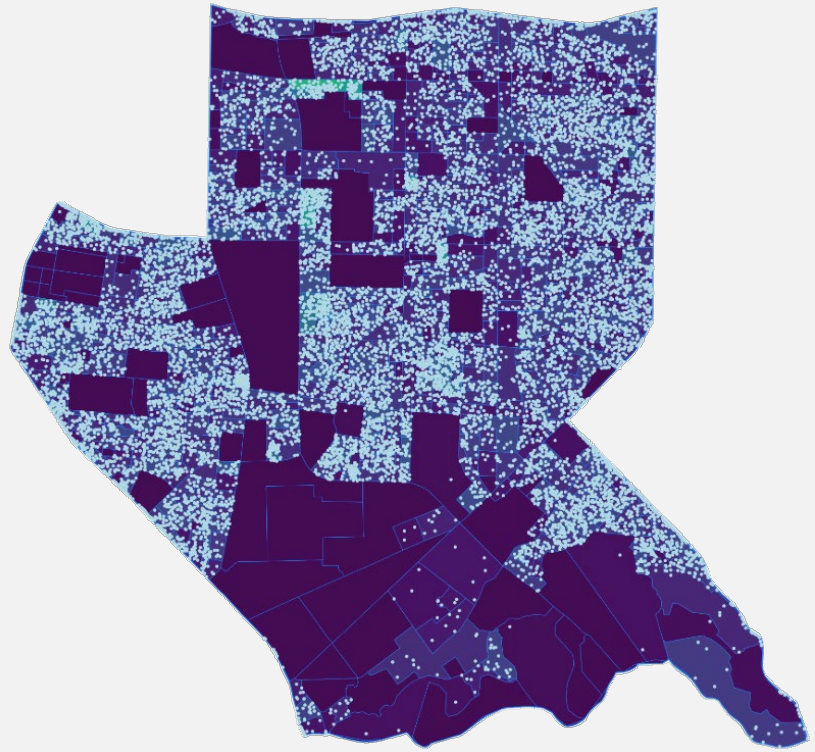
```
# get TSA
tsa = o4[o4['U_ID']==3602].reset_index(drop=True)
# generate 500 random points
rdpt = tsa.sample_points(500)
# plot results
fig, ax = plt.subplots(figsize=[8,12], dpi=300)
tsa.plot(fc='none', ec='dodgerblue', ax=ax)
rdpt.plot(markersize=2, fc='k', ax=ax)
plt.axis('off')
plt.show()
```



GeoProcessing :: Weighting Sampling

```
# generate random points with specific column
daan_pt = cb_daan.sample_points(np.int64(cb_daan['POP']/20))
daan_pt
```

```
5997     MULTIPOINT ((306806.565 2767401.47), (306807.1...
7428                          GEOMETRYCOLLECTION EMPTY
6484     MULTIPOINT ((306858.364 2767350.498), (306863....
6646                          GEOMETRYCOLLECTION EMPTY
6322                          GEOMETRYCOLLECTION EMPTY
                              ...
8113     MULTIPOINT ((303776.54 2770839.106), (303793.0...
8356     MULTIPOINT ((303954.353 2770864.132), (303974....
7615     MULTIPOINT ((304048.838 2770851.475), (304049....
7618     MULTIPOINT ((303892.568 2770886.067), (303893....
11290    MULTIPOINT ((303863.782 2770893.432), (303864....
Name: sampled_points, Length: 1366, dtype: geometry
```



Assignment #01

- 請整併下列資料成為單一的**GeoDataFrame**，再另存成shapefile檔：
  - 113年12月行政區人口指標_村里_雲林縣
  - 113年12月行政區人口統計_村里_雲林縣
  - 113年12月行政區原住民人口指標_村里_雲林縣
  - 113年12月雲林縣統計區15歲以上人口教育程度統計_最小統計區
  - 110年綜稅綜合所得總額各縣市鄉鎮村里統計分析表-雲林縣
- 再利用OLS回歸，來尋找影響收入的重要因子：

  收入中位數 = 人口總數 + 原住民人數 + 平均教育程度 + 老化指數 + 扶老比 + 扶幼比

  將回歸報表轉成pd.DataFrame，再另存成爲csv檔。

Assignment #01

面積加權法:

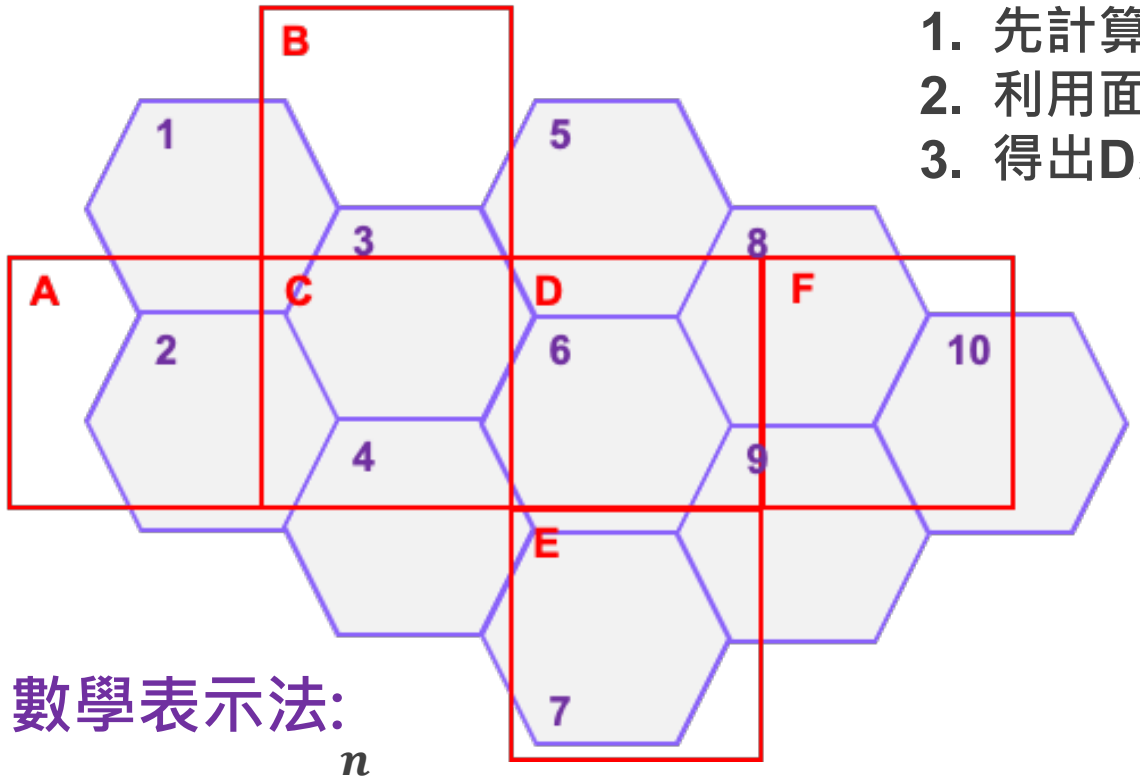
- 當最小統計區資料整併到村裡尺度資料的時候，必須使用面積加權法來進行資料合併的作業。
- 也就是說，當今天村里(v1)橫跨到最小統計區(cb1、cb2與cb3)的時候、必須要先計算v1佔最小統計區(cb1、cb2與cb3)的面積，再乘上每個最小統計區(cb1、cb2與cb3)的人口，加總得出村里(v1)的人口數。

Assignment #01

平均教育程度：

教育程度	累積受教年數	該村里人數	加權後
國小	6	2	12
國中	9	1	9
高中/高職/三專	12	3	36
五專	14	4	56
二技	16	5	80
大學/四技	16	1	16
碩士	18	0	0
博士	23	0	0
	Total	**16**	**209**
		Average	**13.06 years**

Assignment #01



運用面積加權法計算D網格的人口數:

1. 先計算每個六角形佔D網格的面積比例
2. 利用面積比x該六角形佔D網格的面積比例，再加總起來
3. 得出D網格的人口數

數學表示法:

$$POP_{\mathrm{D}} = \sum_{i}^{n} POP_i \times \frac{Area_{i\ onto\ D}}{Area_i} \times 100\%, where\ i = \{3, 4, 5, 6, 8, 9\}$$

The End

Thank you for your attention!

Email: chchan@ntnu.edu.tw

Website: https://toodou.github.io/