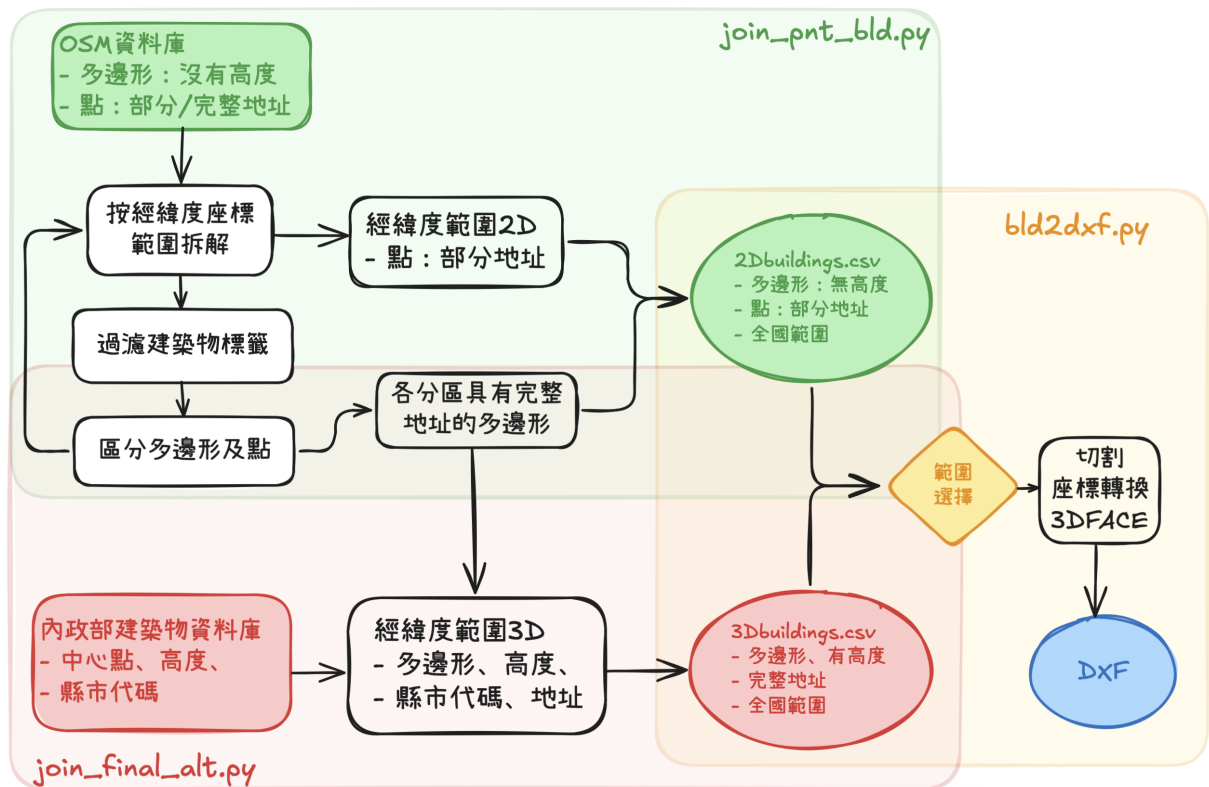


建築物的整理與轉換

- 內政部3維建築物資訊及OSM建築物之整合有其必要性及需求，這個作業除了輸出格式的特殊性之外，過程也充滿了處理工具的開發應用。因為這2項資料來源都會不斷更新，未來在工具面還有待更簡化與整合，來提高流程的自動化與智慧化。
- 整體工作流程如圖所示，區分為3大區塊

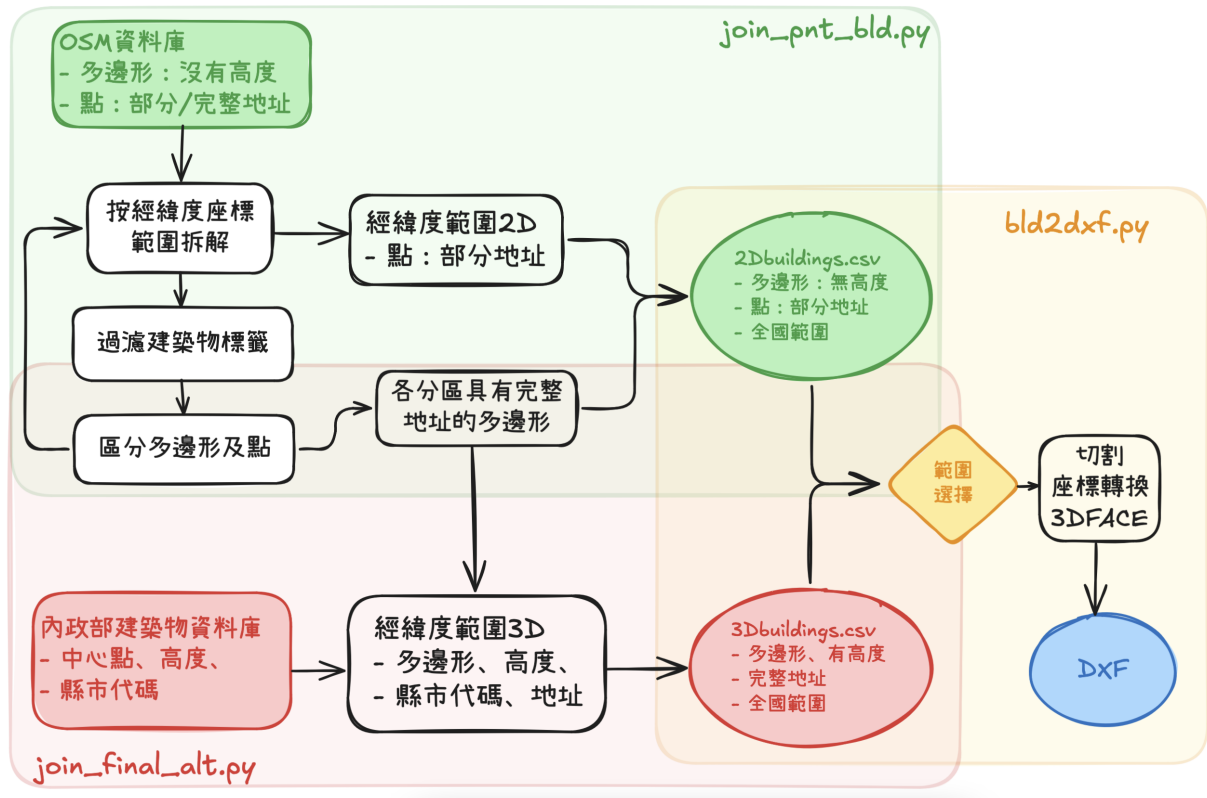


- OSM建築物與節點高度的整併
- OSM建築物與內政部資訊的整併
- bld2dxf.py
- OSM處理過程所用到的工具

OSM檔案的拆分

背景

- 整體建築物資料庫的整理、整併、與切割應用的工作流程如圖所示。
- 大致上整體工作區分為3大區塊。OSM資料庫的拆分屬join_pnt_bld.py的前處理。



• 任務說明如下：

- 拆分taiwan.osm
- 過濾建築物標籤
- 區分多邊形與節點
- 從地址中可以辨識所在縣市之篩選
- 整合節點與多邊形

全台OSM數據的拆分策略

-任務說明

- OSM檔案是個ASCII檔，按順序區分為節點(node)、道路(way含LineString及Polygon, MultiPolygon)，與關聯(relation)等3個段落。何以可以寫成這樣不理解，猜測應該是資料庫輸出的結果。
- 因為道路與關聯是用節點標籤來表示，如果在同一個程式內來處理檔案，會造成記憶體不足的衝擊，必須先行拆分。
- 拆分的官方建議：使用online服務 (overpass)、QGIS、使用osmconvert或ogr2ogr等命令列工具(參考GIS StackExchange的討論)。QGIS個人就不建議了，(猜)記憶體也會被卡死。
- 任務策略檢討說明如下
 - 0方案：不區分直接進行空間切割。好處是不必太多的前處理。壞處是必須將節點全部載入記憶體，才會讀取道路，這超過記憶體容量、且無法同步運作。不論速度與容量，似乎都不太合理。
 - 1方案：將OSM按照各個維度區分另存備用。
- 區分的維度
 - OSM數據的所有維度包括：段落、行政區、空間、與建立的時間
 - 按照段落區分：這個想法應該是比較正統、容易作業化。要資料庫容器分別儲存節點、道路及關聯的定義方式，再以資料庫程式進行關聯計算，擷取所要範圍的幾何物件。

- 按照縣市區分：並不是所有的元件都有完整的地址，道路、relation就沒有太多的屬性信息，用縣市區分不能貫徹。
- 按照經緯度（解析度0.5/0.1度）區分：對節點與多邊形有其可行性。

OSM幾何物件的拆解

- 切割後的物件區分：[osm2csv_points.py](#)、[osm2csv_buildings.py](#),
- 節點會有較完整的訊息，分別按照經緯度順序編號儲存、只儲存具有building屬性內容的節點。
- 整併節點座標後的道路（含建物）。

osm2csv_points.py程式說明

這段程式碼的功能是從 OSM (OpenStreetMap) 檔案中提取含有完整地址的節點，並將其轉換為 CSV 格式的檔案，具體步驟如下：

程式結構

- 匯入必要的庫: 使用 `geopandas` 處理地理數據，`osmium` 處理 OSM 檔案，`shapely` 處理幾何圖形。
- 定義 `OSMHandler` 類別: 繼承自 `osmium.SimpleHandler`，用於解析 OSM 数据。
- 節點處理: 在 `node` 方法中，僅處理包含 "addr:full" 標籤的有效節點，並將其經緯度和標籤儲存到 `self.nodes` 字典中。
- 應用文件: 使用提供的 OSM 檔案名來解析數據。
- 檢查節點: 如果找不到任何節點，則退出程式。
- 創建 `GeoDataFrame`: 將節點數據轉換為 `GeoDataFrame`，並設置坐標系。
- 刪除不必要的欄位: 如果存在 'addr:floor' 欄位，則將其刪除。
- 處理地址: 對於包含 "號" 的地址，修剪出完整地址。
- 去重: 刪除重複的地址，並重設索引。
- 輸出 CSV: 將結果儲存為 CSV 檔案，檔名由原 OSM 檔名轉換而來。整體來說，此程式碼能有效地從 OSM 數據中提取地理信息並導出為可用的 CSV 格式。

輸入

- **OSM 檔案:** 程式接受一個 OSM 格式的檔案作為輸入，該檔案包含了 OpenStreetMap 中的地理資料，包括節點、路徑和關係等數據。
- **依賴庫:** 需要安裝 `geopandas`, `osmium`, 和 `shapely` 等 Python 庫以確保程式正常運行。

輸出

- **CSV 檔案:** 程式會將提取的節點數據轉換並儲存為 CSV 格式的檔案，檔名會以原 OSM 檔名為基礎進行改名，後綴為 `.pnt.csv`。

重要邏輯

1. `OSMHandler` 類別

- 繼承自 `osmium.SimpleHandler`，用於處理 OSM 數據。
- **`node()` 方法:** 當遇到節點時，檢查其位置是否有效，然後提取其標籤並儲存於字典中。

2. 資料篩選: 篩選出具有完整地址的節點，並將地址格式化。

- 透過檢查 `addr:full` 標籤來確認節點是否包含地址信息。
- 使用字符串操作來修剪地址，去掉多餘的部分，並刪除包含 "addr:floor" 的列。

3. **去重與重設索引**: 在輸出之前，程式碼會刪除重複的地址並重設數據框的索引，以確保最終輸出的數據整潔。

較艱澀語法的解釋

- **osmium.SimpleHandler**: 是一個 OSM 數據處理的基類，提供了簡單的接口來處理 OSM 數據的各種元素（如節點、路徑等）。
- **location.valid()**: 檢查節點是否有有效的地理位置，這是提取地址數據的前提。
- **字典推導式**: `{tag.k: tag.v for tag in n.tags}` 用於快速生成一個字典，將標籤的鍵和值配對，簡化了傳統的迴圈方式。

改進建議

1. **錯誤處理**: 增加對檔案讀取和寫入過程中的錯誤處理，以提高程式的穩定性。
2. **進度顯示**: 如果處理的 OSM 檔案非常大，可以考慮增加進度條或日誌輸出，以使用戶了解處理進度。
3. **參數化**: 可以將檔案路徑和輸出格式作為命令行參數來提高靈活性，使用者可以更方便地處理不同的 OSM 檔案。
4. **更全面的數據處理**: 除了地址外，還可以考慮提取其他有用的標籤數據，並進行更深入的數據分析和視覺化。

這些改進可以使得程式更具彈性和易用性，並提升使用者體驗。

下載程式碼 [osm2csv_point.py](#)

```
{% include download.html content="建築物DXF檔之改寫osm2csv\_point.py" %}
```

改進建議

1. **錯誤處理**: 增加對檔案讀取和轉換過程中的錯誤處理，以防止因為格式不正確或檔案不存在而導致程序崩潰。
2. **增強輸出**: 可考慮將結果存儲為 **GeoPackage** 或 **shapefile** 格式，保留地理信息。
3. **性能優化**: 對於大型數據集，可以考慮使用平行處理來加快查詢速度。
4. **使用函數封裝邏輯**: 將主要邏輯分為多個函數，這樣可以提高程式的可讀性和可維護性。
5. **引入日誌功能**: 增加日誌功能以記錄程式運行狀態和錯誤，便於後續調試。

程式說明：osm2csv_buildings.py

輸入

- **OSM 檔案**: 程式接受一個 OSM 格式的檔案作為輸入，該檔案包含了 OpenStreetMap 中的地理資料，包括節點、路徑和關係等數據。
- **依賴庫**: 需要安裝 **geopandas**, **osmium**, 和 **shapely** 等 Python 庫以確保程式正常運行。

輸出

- **CSV 檔案**: 程式會將提取的節點數據轉換並儲存為 CSV 格式的檔案，檔名會以原 OSM 檔名為基礎進行改名，後綴為 **.pnt.csv**。

重要邏輯

1. OSMHandler 類別

- 繼承自 `osmium.SimpleHandler`，用於處理 OSM 數據。
- `node()` 方法: 當遇到節點時，檢查其位置是否有效，然後提取其標籤並儲存於字典中。

2. 資料篩選: 篩選出具有完整地址的節點，並將地址格式化。

- 透過檢查 `addr:full` 標籤來確認節點是否包含地址信息。
- 使用字符串操作來修剪地址，去掉多餘的部分，並刪除包含 "addr:floor" 的列。

3. 去重與重設索引: 在輸出之前，程式碼會刪除重複的地址並重設數據框的索引，以確保最終輸出的數據整潔。

較艱澀語法的解釋

- `osmium.SimpleHandler`: 是一個 OSM 數據處理的基類，提供了簡單的接口來處理 OSM 數據的各種元素（如節點、路徑等）。
- `location.valid()`: 檢查節點是否有有效的地理位置，這是提取地址數據的前提。
- 字典推導式: `{tag.k: tag.v for tag in n.tags}` 用於快速生成一個字典，將標籤的鍵和值配對，簡化了傳統的迴圈方式。

改進建議

1. **錯誤處理**: 增加對檔案讀取和寫入過程中的錯誤處理，以提高程式的穩定性。
2. **進度顯示**: 如果處理的 OSM 檔案非常大，可以考慮增加進度條或日誌輸出，以使用戶了解處理進度。
3. **參數化**: 可以將檔案路徑和輸出格式作為命令行參數來提高靈活性，使用者可以更方便地處理不同的 OSM 檔案。
4. **更全面的數據處理**: 除了地址外，還可以考慮提取其他有用的標籤數據，並進行更深入的數據分析和視覺化。

這些改進可以使得程式更具彈性和易用性，並提升使用者體驗。

下載程式碼 [osm2csv_point.py](#)

```
{% include download.html content="建築物DXF檔之改寫osm2csv\_point.py" %}
```

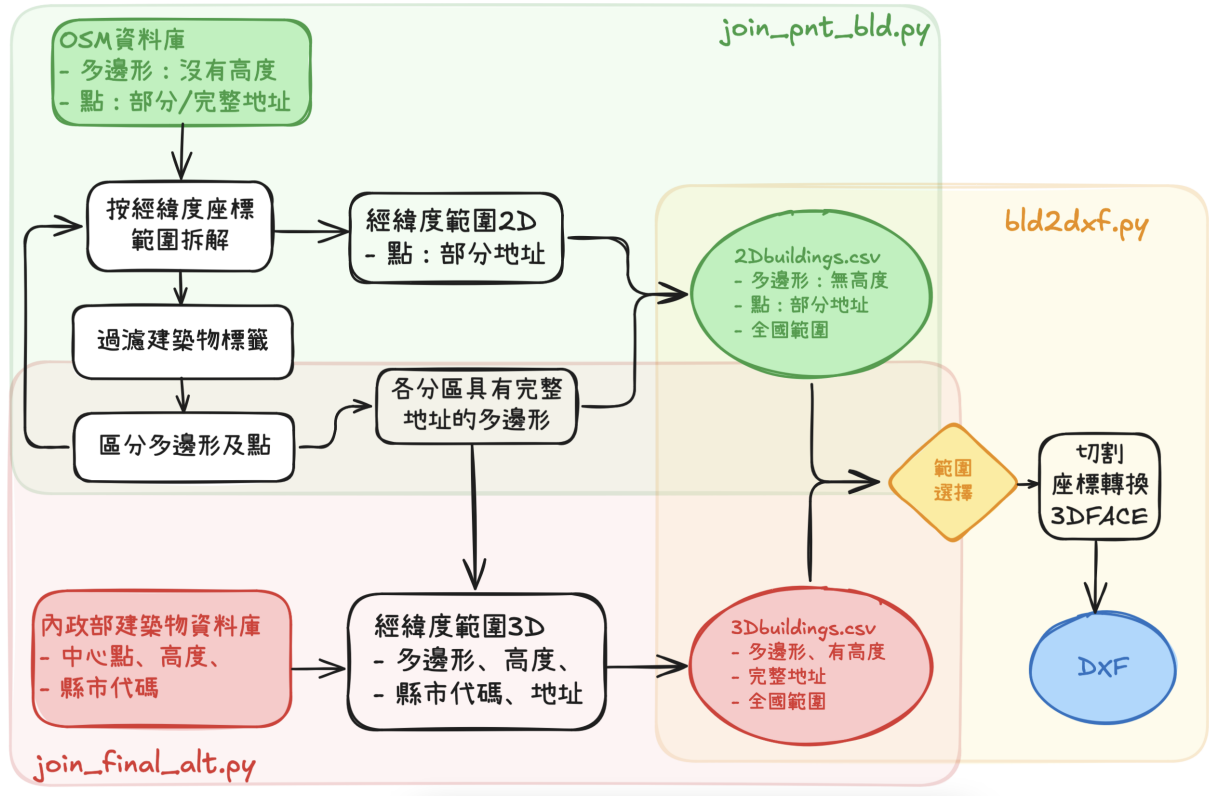
改進建議

1. **錯誤處理**: 增加對檔案讀取和轉換過程中的錯誤處理，以防止因為格式不正確或檔案不存在而導致程序崩潰。
 2. **增強輸出**: 可考慮將結果存儲為 `GeoPackage` 或 `shapefile` 格式，保留地理信息。
 3. **性能優化**: 對於大型數據集，可以考慮使用平行處理來加快查詢速度。
 4. **使用函數封裝邏輯**: 將主要邏輯分為多個函數，這樣可以提高程式的可讀性和可維護性。
 5. **引入日誌功能**: 增加日誌功能以記錄程式運行狀態和錯誤，便於後續調試。
- 道路含多邊形、如何區分
 - 讀取套件不會自行判定是否為封閉曲線

建築物DXF檔之讀寫

背景

- 整體工作流程如圖所示，區分為3大區塊。本文處理最後的整併、切割、座標轉換與DXF輸出。



- 大多數策略討論分散在各章節段落下處理

多邊形是否重新排序

- 這個議題源自於發展過程中多邊形3維面始終無法閉合之嘗試方案。
- 重新排序會造成災難。
- 不能閉合的對策
 - 對多點的多邊形增加迴圈數
 - 增加polyline2d，強加描繪閉合。

viewer

- viewer不影響我們提供的服務。但是好的viewer會左右我們更新、技術提升的速度。
- 詳情請見[DXF檔案之預覽](#)

納入沒有高度的建築物

- 內政部建築物高度的資訊並不是普遍在每個縣市，有很多地區不是以建築物為標籤，或者沒有高度，造成整併困難，有必要在無法順利進行篩選切割時，適時導入這類品質不佳的資訊。
- 此處設定啟動門檻是範圍之內，如果整併內政部高度的建築物小於4座，此時才讀取點狀的建築物座標。

```
#read the 3d buildings
roots='/nas2/kuang/MyPrograms/CADNA-A/OSM'
dim=['3D','2D']
for d in dim[:]:
    fname=f"{roots}/building{d}.csv"
    polygons_gdf = pd.read_csv(fname)
```



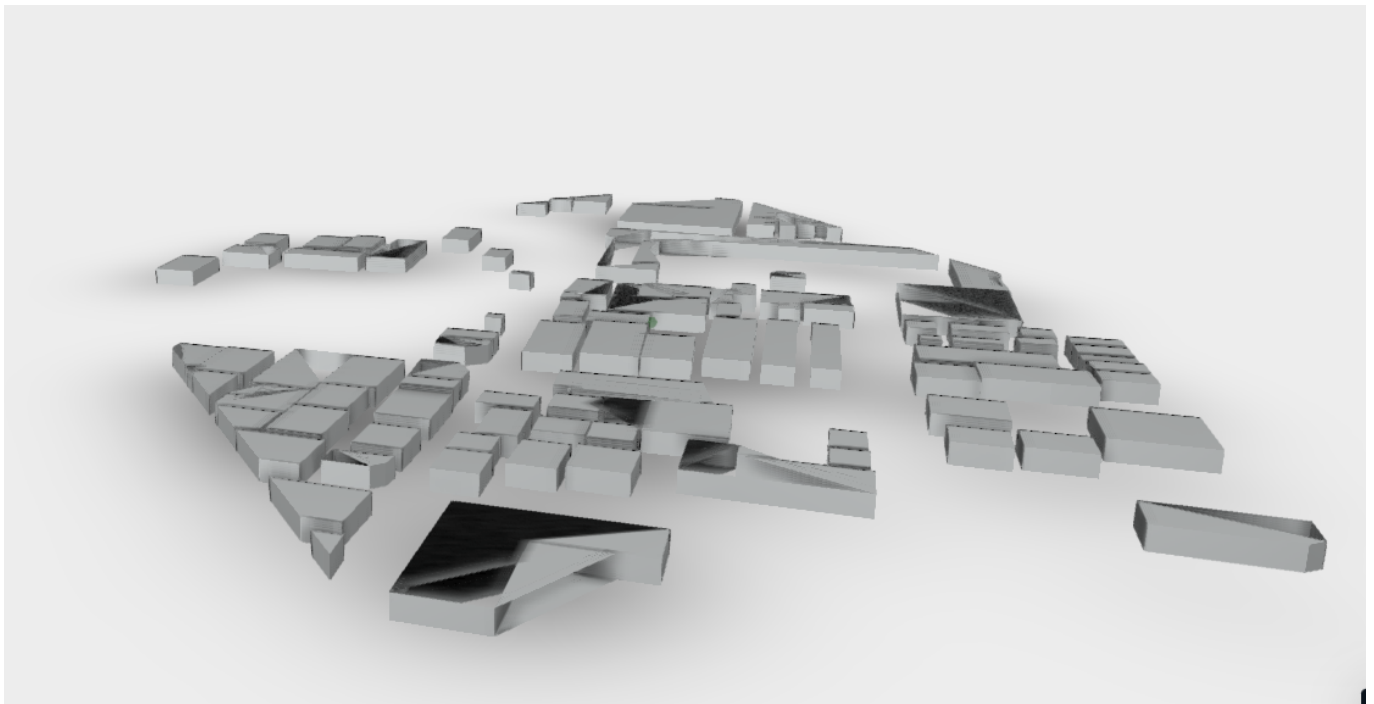
```

polygons_gdf['geometry'] = polygons_gdf['geometry'].apply(loads)
polygons_gdf = gpd.GeoDataFrame(polygons_gdf, geometry='geometry',
crs="EPSG:3857")
p = []
for i, geom in enumerate(['Point', 'Polygon', 'MultiPolygon']):
    gdf_tmp = polygons_gdf.loc[polygons_gdf.geometry.map(lambda x: x.geom_type
== geom)]
    p.append(gpd.overlay(gdf_tmp, bounds_gdf, how='intersection'))
sliced_gdf = pd.concat(p, ignore_index=True)
if len(sliced_gdf)>=4:
    break
if len(sliced_gdf)==0:
    return 'no buildings'
sliced_gdf = gpd.GeoDataFrame(sliced_gdf, geometry='geometry',
crs=polygons_gdf.crs)

```

範圍切割

- 多邊形經`overlay`切割，在邊界上會出現線性切割且保持閉合的結果。如圖所示：



高程數據的應用與程式說明

地形高程的策略考量

- 因為是三維的模型，建築物的基地高程就變得很重要。還好已經有準備好的地形檔案可以快速引用。
- 策略面要考慮的是
 - 高程需不需要內插到多邊形的每一個點？這樣建築物會不會歪斜？這也會牽動到執行量與執行速率。（代表點就足夠）
 - 如果不需要每一點，那建築物該取哪一點來代表它的位置？最低、最高、中心？平均高程？（中心點最近的格點高程）
 - 內插機制怎麼樣做到很快速又正確？（經測試2維內插還不如KDtree又快又正確）

IO摘要

函式	輸入	輸出	邏輯
<code>rd_mem(shape)</code>	從 <code>params.txt</code> 文件讀取參數，並提取出網格的起始點、尺寸以及網格的分辨率。	讀取三個不同的內存映射數據文件 (lat, lon, data)	
<code>cut_data(swLL, neLL)</code>	裁切的地形數據範圍。西南及東北 (緯度, 經度)	地形數據框 <code>grid_df</code>	
<code>grid_prep (grid_df)</code>	格點與高程	pnts_tw,pnts_ll,data_arr,ll_tree, TWD97與經緯度格點陣列、高程陣 列、KD樹操作元	
<code>query_e(points, data_arr, ll_tree)</code>	位置、高程陣列、經緯度查詢KD樹	高程	按最近 的經緯 度回應

較艱澀語法的解釋

- `np.memmap`:
 - 這是一個 `numpy` 的函數，用於創建一個內存映射的數組，允許從磁碟中部分讀取數據，這樣可以處理比內存大得多的數據集。
- `ll_tree.query(coords, k=1)`:
 - 利用空間數據結構（如 `KD 樹`）來快速查找最近的點。`k=1` 表示只查找最近的一個點。

改進建議

- **錯誤處理**: 在讀取文件和處理數據時，應加入錯誤處理機制，防止因文件不存在或格式錯誤導致的崩潰。
- **彈性設定**: 參數如 `dx` 和 `dy` 可以讓使用者自定義，以便在不同的數據集上進行適應性裁切。
- **性能優化**: 如果數據集非常大，可以考慮使用更高效的數據讀取和處理方法，例如分批讀取或使用更高效的數據庫進行查詢。
- **文檔註解**: 增加對每個函數的詳細文檔註解，解釋其參數、返回值和功能，讓未來的維護和使用更加方便。

相關程式碼

```
def rd_mem(shape):
    fnames=['lat','lon','data']
    d = []
    for f in fnames:
        filename = f+'.dat'
        d.append(np.memmap(filename, dtype='float32', mode='r', shape=shape))
    return d

def cut_data (swLL,neLL):
    #load terrain data
    with open('params.txt','r') as f:
        line=[i.strip('\n') for i in f][0]
```



```

x0,y0,nx,ny,dx,dy=(float(i) for i in line.split())
nx,ny=int(nx),int(ny)
shape=(ny, nx)
lat,lon,data=rd_mem(shape)
data = np.where(data < 0, 0, data)
x=[x0+dx*i for i in range(nx)]
y=[y0+dy*i for i in range(ny)]
xg, yg = np.meshgrid(x, y)

idx=np.where((lat>=swLL[0])&(lat<=neLL[0])&(lon>=swLL[1])&(lon<=neLL[1]))
if len(idx[0])==0:
    return 'LL not right!',list(swLL)+list(neLL)
bounds=
[np.min(xg[idx[0],idx[1]]),np.max(xg[idx[0],idx[1]]),np.min(yg[idx[0],idx[1]]),np.
max(yg[idx[0],idx[1]])]
dd=
{'twd97X':xg[idx[0],idx[1]],'twd97Y':yg[idx[0],idx[1]],'lat':lat[idx[0],idx[1]],'lon':lon[idx[0],idx[1]],'data':data[idx[0],idx[1]]}
grid_df=pd.DataFrame(dd)
ib=[x.index(bounds[0]),x.index(bounds[1]),y.index(bounds[2]),y.index(bounds[3])]
return grid_df

def grid_prep (grid_df):
    lon_arr = grid_df['lon'].values
    lat_arr = grid_df['lat'].values
    twd97X_arr = grid_df['twd97X'].values
    twd97Y_arr = grid_df['twd97Y'].values
    pnts_tw=[Point([i,j]) for i,j in zip(twd97X_arr,twd97Y_arr)]
    pnts_ll=[Point([i,j]) for i,j in zip(lon_arr,lat_arr)]
    data_arr = grid_df['data'].values
    ll_coords = np.column_stack((lon_arr, lat_arr))
    ll_tree = cKDTree(ll_coords)
    return pnts_tw,pnts_ll,data_arr,ll_tree

def query_e(points,data_arr,ll_tree):
    coords = np.array([list(p.coords) for p in points])
    _, indices = ll_tree.query(coords, k=1)
    return [data_arr[indices[i]][0] for i in range(len(points))]

...
grid_df=cut_data (swLL,neLL)
pnts_tw,pnts_ll,data_arr,ll_tree=grid_prep (grid_df)
...
sliced_gdf['elevation']=[float(i) for i in
query_e(sliced_gdf.mean_ll,data_arr,ll_tree)]
...
bot=float(sliced_gdf.loc[i,'elevation'])
...

```

幾何物件頂點的座標轉換程式說明

策略考量

- 雖然投影系統已經使用`crs="EPSG:3857"`，但使用者還是習慣TWD97座標系統，因此所有的經緯度都必須進行轉換。
- 內政部DTM image檔案中已經有格點的TWD97座標值，是否可以取最近值？或取2維內插？結果證實雖然是很快速，但解析度不足，效果很差。
- 最後還是以Proj4座標轉換效果最好，重要設定如下：
 - 中心點：取切割範圍的中心點經緯度、及其最近格點的TWD97座標值，即使會有網格解析度的誤差（此處為20m），但全區只有一個原點，是不會產生相對誤差的。與其他物件(如DTM、自測高程、其他CAD設計圖檔)整併時，可能還需要進一步校準。
 - lat0/lat1: 10~40

1. 程式的輸入

- `query_g(polygon_ll, pnyc):`
 - `polygon_ll`: 一個多邊形物件，通常來自於 Shapely 庫，包含了多邊形的外部坐標。
 - `pnyc`: 一個函數，負責將經緯度轉換為特定的坐標系（本例中為 TWD97）。
- `query_g2(point, pnyc):`
 - `point`: 一個點物件，包含其經度和緯度。
 - `pnyc`: 同上，為坐標轉換函數。
- `query_mp(polygon_ll, pnyc):`
 - `polygon_ll`: 一個 MultiPolygon 物件，包含多個多邊形的集合。
 - `pnyc`: 同上，為坐標轉換函數。
- `bld(swLL, neLL):`
 - `swLL`: 包含西南角坐標的列表或元組，格式為 (緯度, 經度)。
 - `neLL`: 包含東北角坐標的列表或元組，格式為 (緯度, 經度)。

2. 程式的輸出

- `query_g`: 返回轉換後的多邊形物件 (Polygon)。
- `query_g2`: 返回一個包含四個頂點的正方形多邊形物件 (Polygon)。
- `query_mp`: 返回轉換後的 MultiPolygon 物件。
- `bld`: 更新 `sliced_gdf` 中的 `geometry_twd97` 欄位，包含轉換後的幾何形狀。

3. 重要的邏輯

- 使用 `pnyc` 函數將經緯度轉換為特定坐標系，這是地理資訊系統 (GIS) 中常見的操作。
- `query_g`、`query_g2`和`query_mp` 透過將經緯度轉換為新的坐標系來處理多邊形和點，分別處理不同的幾何類型。
- `bld` 函數負責整合所有的轉換邏輯，並將結果存回到 `sliced_gdf`，這是一個 GeoDataFrame。

4. 較艱澀語法的解釋

- `np.array(...)`: 使用 NumPy 陣列來處理坐標資料，這樣可以進行更高效的數據操作。
- `zip(...)`: 這個函數將多個可迭代對象打包在一起，常用於組合不同的列表或數組。

- `sliced_gdf.geometry.map(...)`: 在 GeoDataFrame 中使用 `map` 方法來過濾資料，這是一種方便的方式來處理大型地理資料集。

5. 改進建議

- **錯誤處理**: 增加對輸入資料的驗證和錯誤處理，確保坐標範圍有效。
- **性能優化**: 在處理大型資料集時，考慮使用並行處理或批次處理來提高效率。
- **文檔註解**: 增加詳細的註解和文檔，解釋每個函數的用途和參數，提升可讀性和可維護性。

座標轉換相關程式碼

```
def query_g(polygon_ll, pnyc):
    lon_new=np.array([coord[0] for coord in polygon_ll.exterior.coords])
    lat_new=np.array([coord[1] for coord in polygon_ll.exterior.coords])
    twd97X,twd97Y=pnyc(lon_new, lat_new, inverse=False)
    return Polygon([(x, y) for x, y in zip(twd97X, twd97Y)])

def query_g2(point, pnyc):
    f=6/2
    x,y=pnyc(point.x, point.y, inverse=False)
    return Polygon([(x-f, y-f),(x+f, y-f),(x+f, y+f),(x-f, y+f),])

def query_mp(polygon_ll, pnyc):
    polygon_twd=[]
    for polygon in polygon_ll.geoms:
        polygon_twd.append(query_g(polygon, pnyc))
    return MultiPolygon(polygon_twd)

def bld(swLL, neLL):
    ...
    south,west=swLL[0],swLL[1]
    north,east=neLL[0],neLL[1]
    bbox_center = Point((west + east) / 2, (south + north) / 2)
    point_tw=query_p(bbox_center, pnts_tw, pnts_ll, ll_tree)
    Xcent, Ycent = point_tw.x, point_tw.y
    pnyc = Proj(proj='lcc', datum='NAD83', lat_1=10, lat_2=40,
        lat_0=bbox_center.y, lon_0=bbox_center.x, x_0=Xcent, y_0=Ycent)
    ...
    p=sliced_gdf.loc[sliced_gdf.geometry.map(lambda x: x.geom_type=='Polygon')]
    if len(p)>0:
        plg=list(p.geometry)
        sliced_gdf.loc[p.index, 'geometry_twd97']=[query_g(i, pnyc) for i in plg]
    m=sliced_gdf.loc[sliced_gdf.geometry.map(lambda x: x.geom_type=='MultiPolygon')]
    if len(m)>0:
        mplg=list(m.geometry)
        sliced_gdf.loc[m.index, 'geometry_twd97']=[query_mp(i, pnyc) for i in mplg]
    s=sliced_gdf.loc[sliced_gdf.geometry.map(lambda x: x.geom_type=='Point')]
    if len(s)>0:
        points=list(s.geometry)
        sliced_gdf.loc[s.index, 'geometry_twd97']=[query_g2(i, pnyc) for i in points]
```

ezdxf輸出函式的應用及程式說明

- **Vec3**物件
 - 一如在數值地形資料的輸出，如果只是輸出線形物件，後續程式仍然不能正確讀取DXF檔案，因為DXF的點都必須是3度空間完整定義(**Vec3**物件)
- **3dface**
 - 使用**3dface**的必要性似乎是很直覺的，畢竟要輸出的是建築物的立方體，會需要上層、底層、各個立面的平面。
 - **layer**的高程設定卡了一陣子無法決定，如果建築物在山坡上，**layer**的高程究竟該取上、下層的高程？平均？答案是：都沒差別、在**polyliine3d**的個案中，直接把**layer**的高程設為0，徹底解決斷層的問題。
- **polyliine2d**物件
 - 會想重複提供多邊形邊框的描繪是查看範例檔案中有很多**LineString**物件，並沒有測試其必要性。
 - 在繪圖預覽過程中，有**polyliine2d**物件對不夠完整的**3dface**集合會有很大的幫助，建築物看起來會比較平整。
- 參考
 - [ezdxf官網說明](#)
 - [等高線DXF輸出的經驗](#)
 - [範例DXF檔案轉python碼](#)

輸入

- **sliced_gdf**: 一個 GeoDataFrame，包含多邊形的幾何資訊及其相關屬性（如 **elevation** 和 **maxAltitude**）。
- **p, s, m**: 這些變數代表不同的索引集合，用於確定當前處理的幾何形狀類型。分別是多邊形、點狀、與多個多邊形。

輸出

- **fname**: 生成的 DXF 檔案名稱。
- **output**: 包含生成的 DXF 檔案的二進位資料，方便後續操作或直接返回給使用者。

重要邏輯

1. **DXF 檔案創建**: 使用 **ezdxf** 庫創建新的 DXF 檔案，並設置模型空間。
2. **分層管理**: 根據每個多邊形的索引創建不同的圖層，並將多邊形的底部和頂部高度計算出來。
3. **多邊形處理**: 根據 **sliced_gdf** 中的幾何資料，將多邊形的外部點提取出來，並為每個多邊形生成上下平面及其立面。
 - 使用 **add_polyline2d** 和 **add_3dface** 方法將生成的點添加到模型空間中。
4. **立面生成**: 對於每個多邊形的邊，生成對應的立面，並將其添加到 DXF 檔案中。

較艱澀語法的解釋

- **Vec3**: 用於表示三維空間中的點，通常包含 **x, y, z** 三個座標。
- **add_polyline2d**: 用於將二維折線添加到模型空間中。
- **add_3dface**
 - 用於將**三維面**添加到模型空間中，通常用於構建立體幾何。

- **三維面**的限制是一次只能有3點、至多4點、同一平面的點。因此當點數多於4點的建築物，就必須執行迴圈，同時也要注意每一點都必須有前後點作為**三維面**，否則會出現多邊形無法閉合的情況。
- MultiPolygon的處理：其函式`.geoms`的形態為個別多邊形所形成的序列，依序執行原來多邊形的任務即可。

改進建議

1. **錯誤處理**: 增加對於輸入資料有效性的檢查，避免因資料格式不正確而導致的執行錯誤。
2. **性能優化**: 對於大規模的多邊形數據，可以考慮使用多執行緒或異步處理來加速生成過程。
3. **功能擴展**: 可以考慮添加選項來支持不同的 DXF 版本，以滿足不同用戶的需求。

這段程式碼的主要目的是將地理數據轉換為 DXF 格式的三維模型，並能夠根據多邊形的特性生成相應的結構。

程式碼

```
...
doc = ezdxf.new(dxfversion="R2010")
msp = doc.modelspace()
align=TextEntityAlignment.CENTER
ii=0
for i in sliced_gdf.index:
    layer_name = f"Polygon_{ii}"
    layer = doc.layers.add(layer_name)
    bot=float(sliced_gdf.loc[i,'elevation'])
    Vbot=Vec3(0, 0, bot)
    top=bot+sliced_gdf.loc[i,'maxAltitude']
    if i in p.index or i in s.index:
        polygons=[sliced_gdf.loc[i,'geometry_twd97']]
    if i in m.index:
        polygons=[polygon for polygon in sliced_gdf.loc[i,'geometry_twd97'].geoms]
    for polygon in polygons:
        points=[j for j in polygon.exterior.coords] #reorder_polygon_points(polygon)
        #上下平面
        for hgt in [bot,top]:
            pnts=[Vec3(p[0],p[1],hgt) for p in points]
            npnts=len(pnts)
            msp.add_polyline2d(pnts, dxfattribs={ "layer": layer_name,
            'elevation':Vbot })
            if npnts <=4 and npnts in [3,4]:
                msp.add_3dface(pnts, dxfattribs={'layer': layer_name})
            else:
                rep=npnts//4
                for k in range(rep):
                    for n in range(2*k, npnts, 4):
                        nnd=min(n+4,npnts)
                        if nnd-n not in [3,4]:continue
                        msp.add_3dface(pnts[n:nnd], dxfattribs={'layer': layer_name})
        #立面
        for j in range(len(points)):
            pj = points[j]
```

```

        p1 = Vec3(pj[0],pj[1],bot)
        pn = points[(j + 1) % len(points)]
        p2 = Vec3(pn[0],pn[1],bot)
        p3 = Vec3(p2[0], p2[1], top)
        p4 = Vec3(p1[0], p1[1], top)
        pnts=[p1, p2, p3, p4]
        msp.add_polyline2d(pnts, dxfattribs={ "layer": layer_name, 'elevation':
Vbot })
        msp.add_3dface(pnts, dxfattribs={'layer': layer_name})
        ii+=1

    ran=tf.NamedTemporaryFile().name.replace('/', '').replace('tmp', '')
    fname='bldn_'+ran+'.dxf'
    output=BytesIO()
    doc.write(output, fmt='bin')
    output.seek(0) # 重置指针位置
    doc.saveas('./dxfs/'+fname)
    return fname,output
...

```

切割套件與應用

app.py

- 這個程式繼承自[API伺服器的設計](#)，新增建築物資料庫的切割功能。
- 輸入bld2dxf模組
- 新增bld(swLL, neLL)之呼叫


index.html

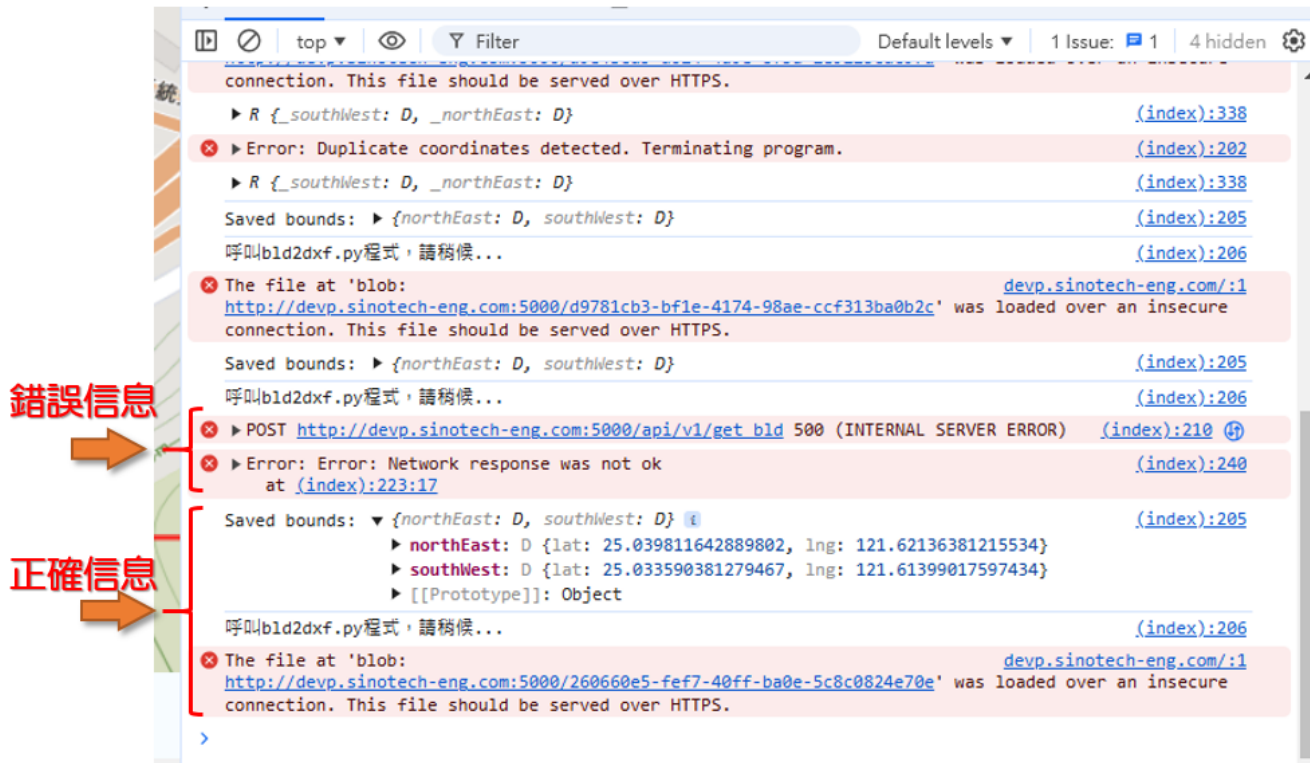
- 新增SaveButtom_b javascript函式

檢查執行進度

- 因每次程式執行都會重新讀取資料庫、切割及座標轉換，因此會需要一些時間，可以進入瀏覽器[檢查](#)介面了解實際情況，是否正確運作。
- 可能無法正確執行的原因
 - 切割範圍的資訊位正確傳遞(檢視saved bounds)
 - 背景數據品質問題(訊息為Network Response not OK，請將經緯度範圍複製給研資部進一步追蹤除錯，TWD97值還需轉換。)
- 空白處按右鍵進入[檢查](#)



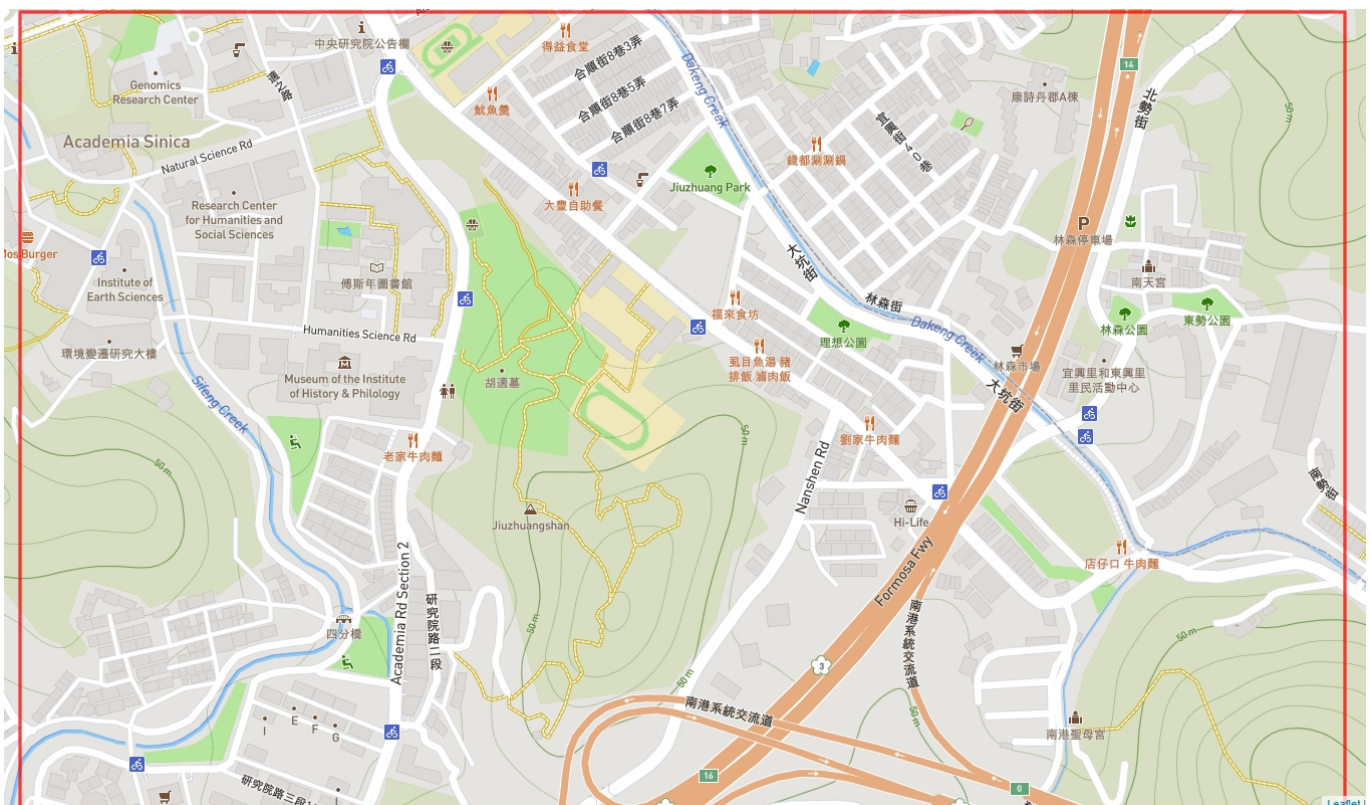
- 點選console(或紅色停止標誌-帶數字 )



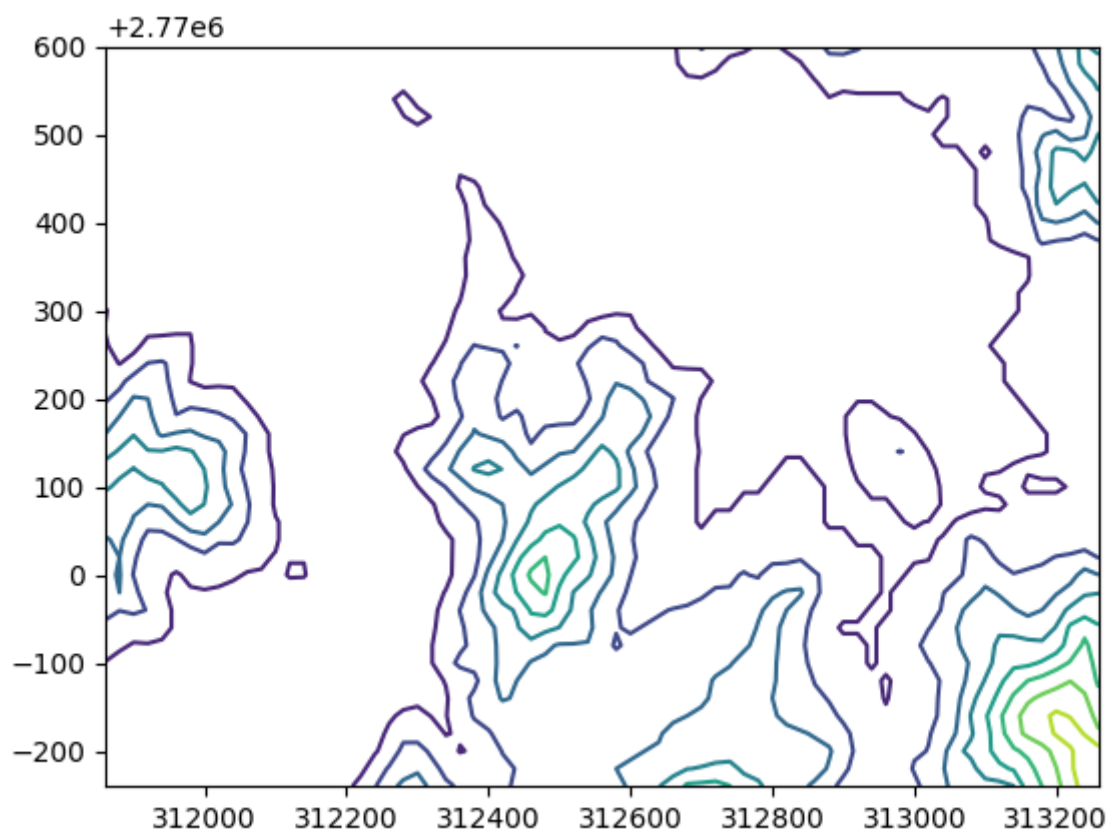
- The file at blob ... is loaded : 是正確信息
 - 請檢視瀏覽器的下載介面。
 - 因為結果檔案是隨機碼，瀏覽器會認為是病毒拒絕直接下載，需進一步確認。

結果

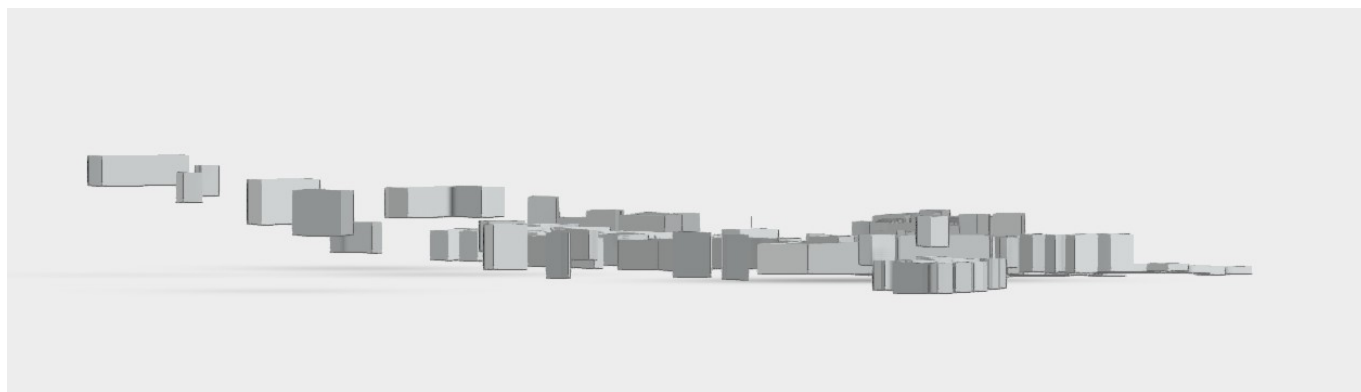
- 中研院附近山坡與平地
- 範圍



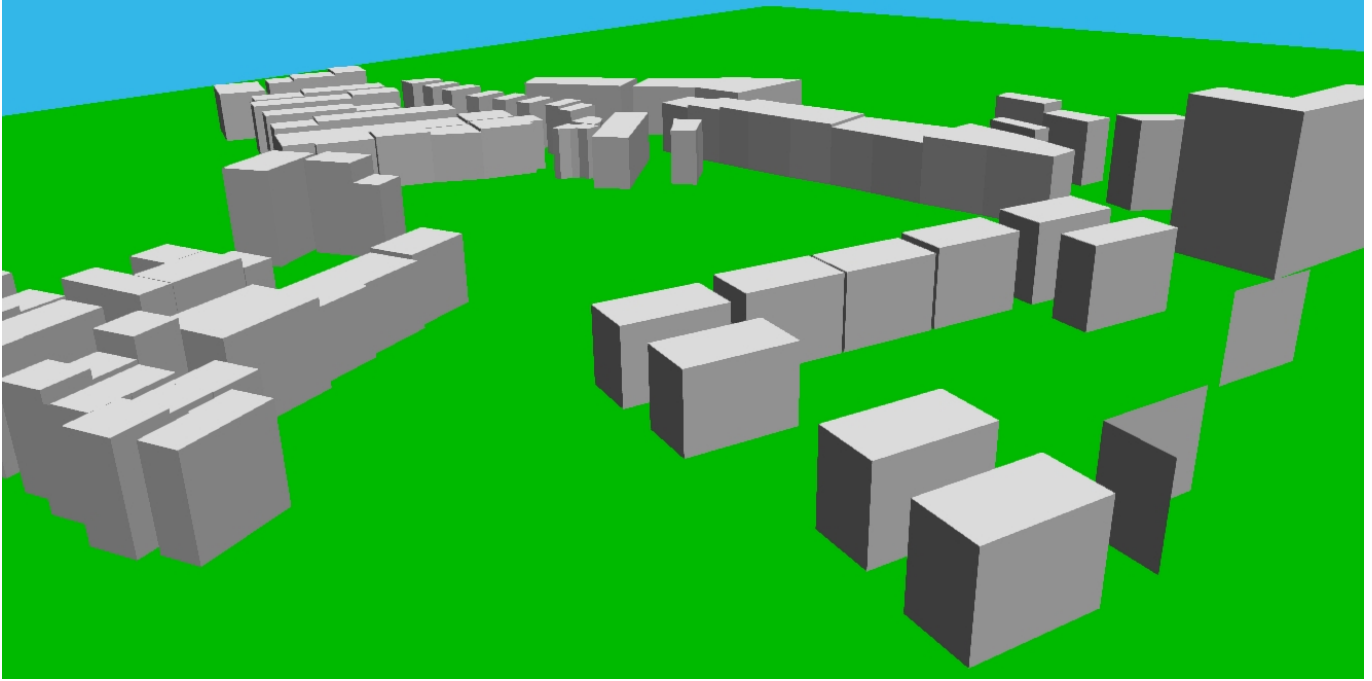
- 地形



- 建築物群



- 進入CADNA模式系統檢視



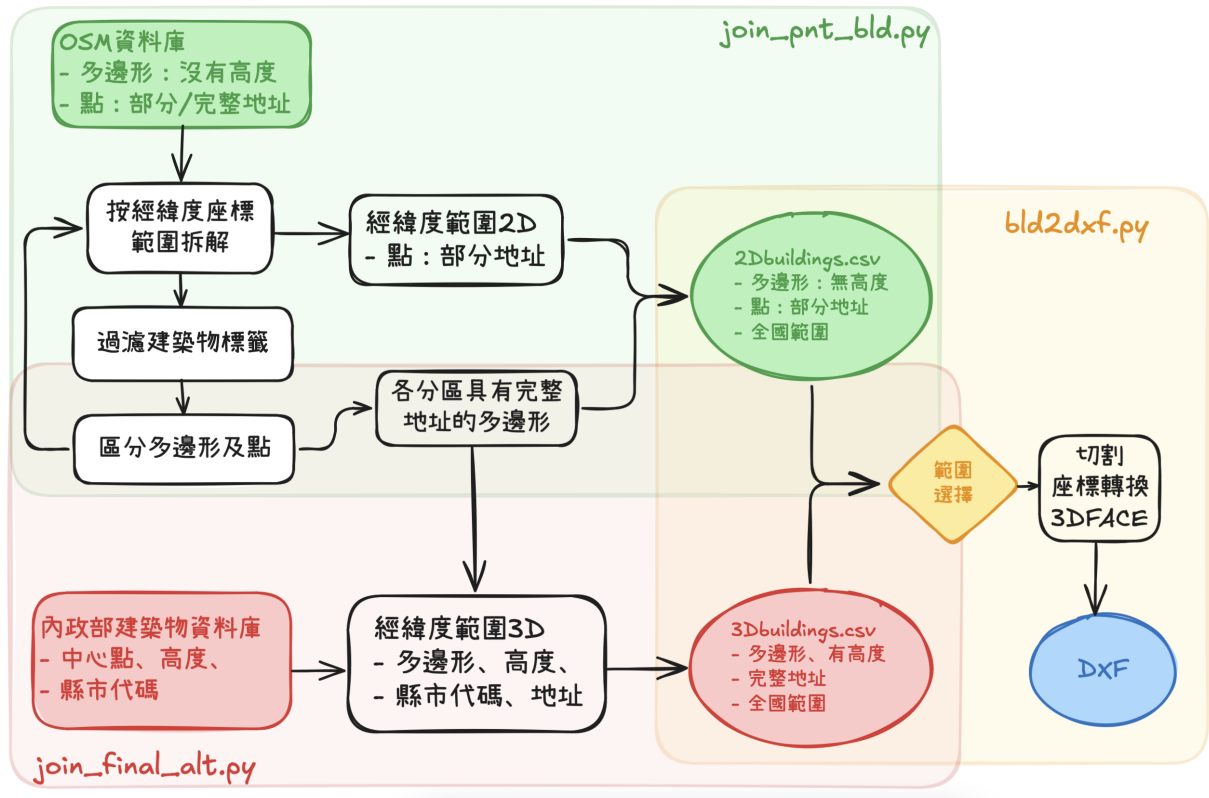
完整程式碼[bld2dxf.py](#)

```
{% include download.html content="建築物DXF檔之改寫bld2dxf.py" %}
```

OSM建築物與內政部資訊的整併 ([join_final_alt.py](#))

背景

- 建築物資料庫的整理、整併、與切割應用的工作流程如圖所示。
- 大致上整體工作區分為3大區塊，[join_pnt_bld.py](#)的範圍與功能，為最終整併成[3Dbuildings.csv](#)的重要程序。



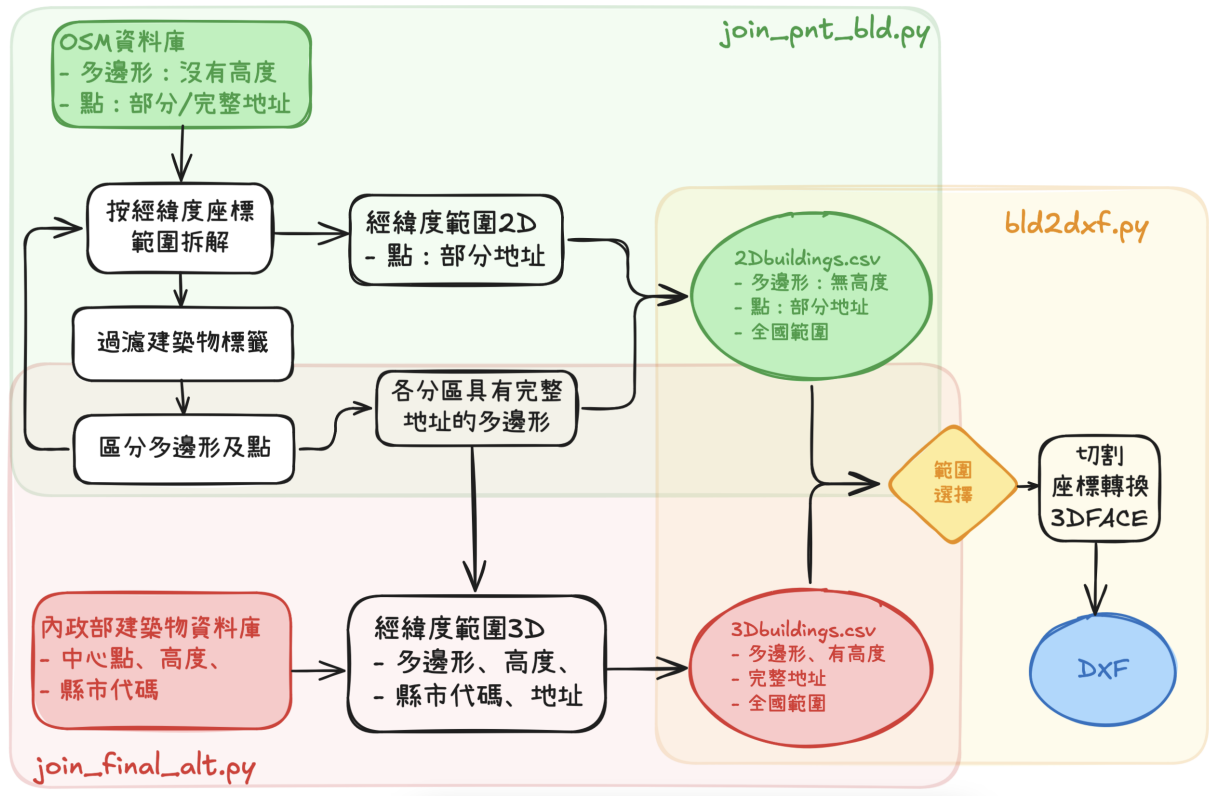
程式說明

程式使用

OSM建築物多邊形與節點的整併 (join_pnt_bld.py)

背景

- 整體建築物資料庫的整理、整併、與切割應用的工作流程如圖所示。
- 大致上整體工作區分為3大區塊。**join_pnt_bld.py**的步驟屬於前期處理，為最終整併成**3Dbuildings.csv**的重要程序。OSM其他沒有高度的建築物資訊，則併入**2Dbuildings.csv**中，在找不到足夠3D資訊的時候可以作為替代。



• 任務說明如下：

- 拆分taiwan.osm
- 過濾建築物標籤
- 區分多邊形與節點
- 從地址中可以辨識所在縣市之篩選
- 整合節點與多邊形

• 任務策略檢討說明如下

全台OSM數據的拆分策略

-任務說明

- OSM檔案是個ASCII檔，按順序區分為節點(node)、道路(way含LineString及Polygon, MultiPolygon)，與關聯(relation)等3個段落。何以可以寫成這樣不理解，猜測應該是資料庫輸出的結果。
- 因為道路與關聯是用節點標籤來表示，如果在同一個程式內來處理檔案，會造成記憶體不足的衝擊，必須先行拆分。
- 拆分的官方建議：使用online服務 ([overpass](#))、QGIS、使用 [osmconvert](#) 或 [ogr2ogr](#) 等命令列工具(參考 [GIS StackExchang](#) 的討論)。QGIS個人就不建議了，(猜)記憶體也會被卡死。

程式說明

這個程式使用 [geopandas](#)、[rtree](#) 和 [shapely](#) 來處理地理空間數據。它的主要功能是將點 (points) 與多邊形 (polygons) 進行空間查詢，找出哪些點位於哪些多邊形內，並將結果寫入 CSV 檔案。

輸入

程式接受兩個命令列參數，這兩個參數是 `GeoDataFrame` 檔案的路徑：

1. 第一個檔案：包含節點的 `GeoDataFrame`。
2. 第二個檔案：包含多邊形的 `GeoDataFrame`。

輸出

程式將生成兩個 CSV 檔案：

1. 第一個檔案：包含匹配的點與多邊形的資料。這個索引的表格是用來檢核結果的正確性，並沒有後續應用的必要性。
2. 第二個檔案：包含最終的多邊形與對應的地址資料。命名原則：`final_pnt_bld.csv`
(`'final'+sys.argv[1]+sys.argv[2]`)

重要邏輯s

1. **讀取 `GeoDataFrames`**：使用 `gpd.read_file()` 讀取點和多邊形資料。
2. **幾何數據轉換**：將 `GeoDataFrame` 中的幾何列從字符串格式轉換為 `Shapely` 對象，以便進行空間運算。
3. **創建 `Rtree` 索引**：使用 `rtree` 包來創建一個 `Rtree` 索引，以加速空間查詢。
4. **查詢 `Rtree` 索引**：對每個點，查詢 `Rtree` 索引以找出可能的多邊形，然後檢查這些多邊形是否包含該點。
5. **生成結果 `GeoDataFrame`**：將匹配的點和多邊形的索引以及幾何資料存儲到一個新的 `GeoDataFrame` 中。
6. **寫入 `CSV`**：將結果 `GeoDataFrame` 輸出為 `CSV` 檔案。

艱澀語法的解釋

- `apply(loads)`：這行代碼將 `loads` 函數應用到幾何列的每一個元素，將其從 `WKT` (Well-Known Text) 格式轉換為 `Shapely` 幾何對象，這樣可以進行空間運算。
- `idx.insert(i, geometry.bounds)`：這行代碼將多邊形的邊界 (`bounds`) 插入到 `Rtree` 索引中，以便後續進行快速查詢。
- `idx.intersection(point.bounds)`：這行代碼查詢 `Rtree` 索引，返回與給定點邊界相交的所有多邊形的索引。

TODO's

下載程式碼 [join_pnt_bld.py](#)

```
{% include download.html content="建築物DXF檔之改寫bld2dxf.py" %}
```

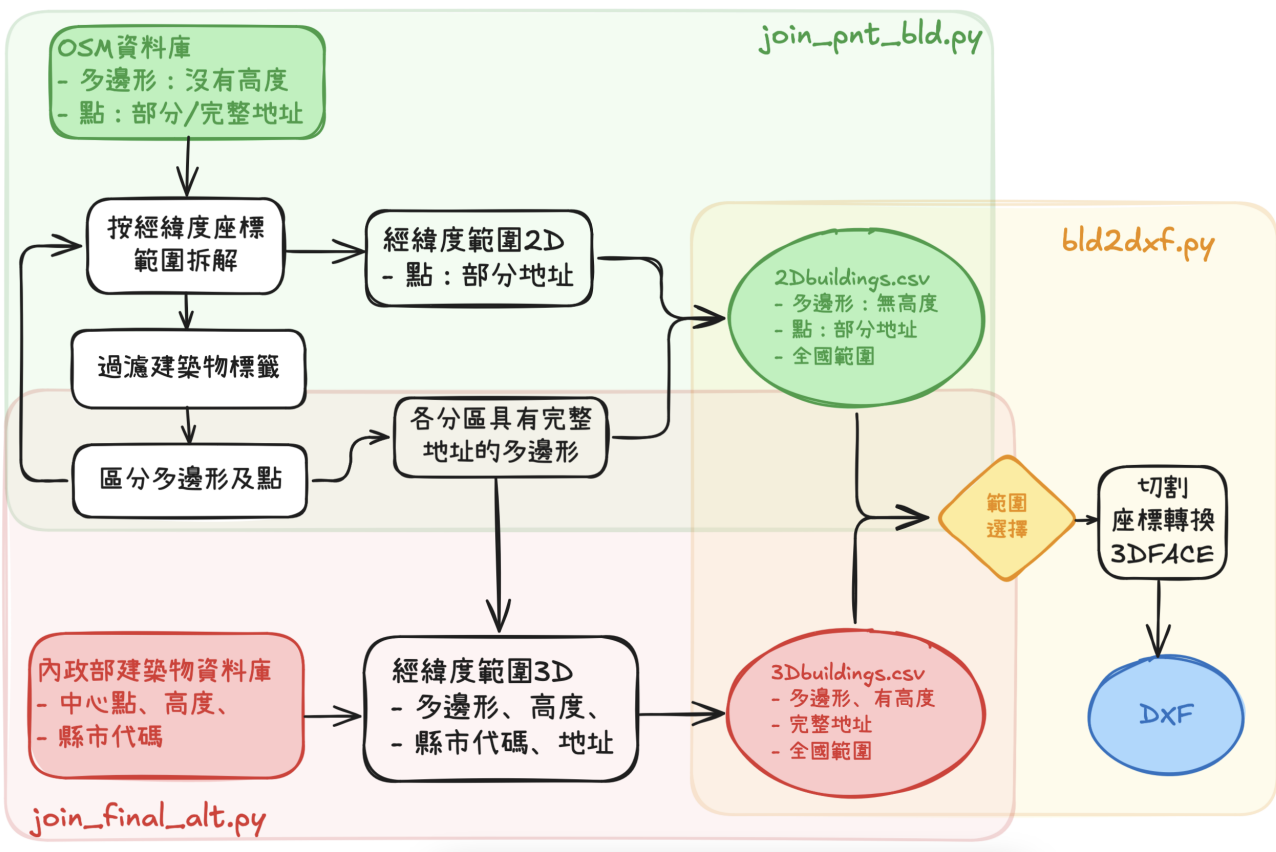
改進建議

1. **錯誤處理**：增加對檔案讀取和轉換過程中的錯誤處理，以防止因為格式不正確或檔案不存在而導致程序崩潰。
2. **增強輸出**：可考慮將結果存儲為 `GeoPackage` 或 `shapefile` 格式，保留地理信息。
3. **性能優化**：對於大型數據集，可以考慮使用平行處理來加快查詢速度。
4. **使用函數封裝邏輯**：將主要邏輯分為多個函數，這樣可以提高程式的可讀性和可維護性。
5. **引入日誌功能**：增加日誌功能以記錄程式運行狀態和錯誤，便於後續調試。

內政部3D建築物數據之處理

背景

- 內政部國土測繪中心多維度服務平台提供全台建築物位置與高度資訊，唯沒有建築物平面的相關訊息，需要與其他平面數據庫如OSM結合應用。
- 建築物命名與檔案切分方式不明，未來合併時以位置座標為參考即可，不需要其他資訊。檔案雖然很多，每個檔案並不是很大，可以平行處理以加速處理效率。
- 為加速整併，此處還是汲取其中的縣市代碼，以備與OSM數據中的地址快速結合。
- 檔案以KML型態儲存，然而並非典型的kml格式，geopandas無法直接讀取，必須以xml.etree.ElementTree 解析。
- 以下為rd_mk13.py的程式說明。



程式說明

這段程式碼的目的是將 KML (Keyhole Markup Language) 檔案轉換為 CSV 格式的檔案，便於進一步分析和使用。程式使用了 Python 的 `pandas` 和 `xml.etree.ElementTree` 模組來處理 KML 檔案中的地理數據。

輸入

- kml_folder**: 包含多個 KML 檔案的資料夾路徑。
- output_csv**: 輸出 CSV 檔案的名稱。
- 在此以所在 (相對) 目錄來定義輸入輸出檔案名稱，以目錄切換作業開啟同步執行。

輸出

- 一個 CSV 檔案 (`output.csv`)，包含從所有 KML 檔案提取的數據，其中每一行代表一個 `Placemark` 的資訊。

重要邏輯

1. **KML 解析**: 使用 `xml.etree.ElementTree` 解析 KML 檔案，並尋找所有的 `Placemark` 元素。
2. **數據提取**:
 - 提取每個 `Placemark` 的名稱 (`name`)。
 - 提取地理位置的經度 (`longitude`) 和緯度 (`latitude`)。
 - 提取相關的區域資料，像是 `maxAltitude` 和 `COUNTY`。
3. **數據整合**: 將從所有 KML 檔案提取的數據合併成一個 Pandas DataFrame，並輸出為 CSV 檔案。

較艱澀語法的解釋

- `ns = {'kml': 'http://www.opengis.net/kml/2.2'}`: 定義了命名空間，以便在解析 XML 時正確識別 KML 元素。
- `placemark.find('..//kml:Model', ns)`: 使用 XPath 查找 `Model` 元素，這是 KML 中存放地理位置的部分。
- `pd.concat(all_data, ignore_index=True)`: 將多個 DataFrame 合併為一個，並重新編排索引。

改進建議

1. **錯誤處理**: 加入對 KML 檔案解析的錯誤處理，以防止因單個檔案格式錯誤而中斷整體處理。
2. **彈性輸出格式**: 可以考慮添加選項，允許用戶選擇輸出的格式（如 JSON、Excel 等）。
3. **參數化命令列**: 使用 `argparse` 模組使得用戶可以透過命令列輸入資料夾路徑和輸出檔案名稱，提高程式的靈活性。

程式之執行

同步執行與結果收穫

- 歷遍將各年度內政部檔案目錄
- 將kml檔案區分成20個子目錄(參考[新創平行運作所需之次目錄](#))
- 移動到每個子目錄、在背景啟動每個子目錄下的[rd_kml3.py](#)工作
- 整併所有目錄的[output.csv](#)成為一個大檔案、整理重複的檔頭，

後續處理

- [join_final_alt.py](#)

OSM處理過程所用到的工具

背景

- 因為工作太過複雜，工具當然是越多越好，如果不夠用也只好自己做。
- 這種分享有時候也很殘忍，因為對某人很好用的工具，對其他人可能就沒那麼好用，見仁見智囉!

區分及整併

新創平行運作所需之次目錄

- 這個問題會發生，是因為檔名沒有規則性，使用**bash**來建目錄很快，但是檔案要區分下去就沒那麼方便了，這一題，LLM的建議是使用**python**來做。
- 將檔案略分為20個群組，這是因為搭配工作站有20個CPU可以同步來執行。
- 使用**np.array_split**，這招厲害，沒聽過也沒用過。
- 中文檔案名稱還可以忍受，半型的小括弧，這就有點挑戰了，需要加個反斜線來進程式化。

```
#kuang@DEVP /nas2/kuang/MyPrograms/CADNA-A/OSM
#$ cat make_sub.py
import numpy as np
import os
kml_folder='./'
fnames=[filename for filename in os.listdir(kml_folder) if
filename.startswith('final')]
split_fnames = np.array_split(fnames, 20)
for i in range(20):
    os.system(f"mkdir -p sub{i}")
p1='/nas2/kuang/MyPrograms/CADNA-A/OSM'
for i in range(20):
    for fname in split_fnames[i]:
        if '(' in fname:
            ff=fname.replace('(', '\\(').replace(')', '\\)')
            os.system(f"ln -sf {p1}/{ff} {p1}/sub{i}/{ff}")
        else:
            os.system(f"ln -sf {p1}/{fname} {p1}/sub{i}/{fname}")
```

整併各個經緯度範圍的building2D結果

- 這不是個小工具，算個小作業，我把ipython上的作業過程留下來，免得重作時還要重新問AI讓AI再想一遍。
- building3D也是2D做出來的，所以需要將其剔除，免得重複處理。
- AI這題的答案也很優秀，很少人會這樣用條件。

```
~building2D['geometry'].isin(building3D['geometry'])
```

```
kuang@DEVP /nas2/kuang/MyPrograms/CADNA-A/OSM
$ cat bld2d.py
cd OSM
import os
import glob
directory = './'
dfs = []
col=['geometry']
for file in glob.glob(os.path.join(directory, 'output??bld.csv')):
    df = pd.read_csv(file)
    dfs.append(df[col])
df[col].head()
```

```
dfs[0].head()
dfs[-1].head()
len(df)
len(dfs)
combined_df = pd.concat(dfs, ignore_index=True)
len(combined_df)
dfs.to_csv('building2D.csv', index=False)
combined_df.to_csv('building2D.csv', index=False)
!lst
mv building2D.csv building2D_all.csv
building3D = pd.read_csv('building3D.csv')
building2D = pd.read_csv('building2D_all.csv')
building2D = building2D[~building2D['geometry'].isin(building3D['geometry'])]
building2D.to_csv('building2D_updated.csv', index=False)
len(building2D)
building2D.head()
history
```

執行腳本

```
$ ls *cs break_osm2.cs do_join.cs (pyn_env) kuang@DEVP /nas2/kuang/MyPrograms/CADNA-A/OSM $ cat *cs
```

全區0.5度解析度之切割腳本

- `break_osm.cs` 這支腳本會連續做5個東西向範圍，再由外部控制南北向(0~7)
- ``

```
for i in {0..5};do
j=$1
left=$(echo "119.2 + $i * 0.5" |bc)
right=$(echo "119.7 + $i * 0.5" |bc)
bottom=$(echo "21.5 + $j * 0.5" |bc)
top=$(echo "22.0 + $j * 0.5" |bc)
osmconvert input.osm -b=${left},${bottom},${right},${top} --complete-ways -
o=output${i}${j}.osm
done
cd sub$1
for i in $(ls final*);do python ../../join_Alt_osmBld.py $i;done
cd ..
```

北高都會區0.1度解析度之切割腳本

```
i=$1
j=$2
start_l=$(echo "119.2 + $i * 0.5" |bc)
start_b=$(echo "21.5 + $j * 0.5" |bc)
for k in {0..5};do
for l in {0..5};do
left=$(echo "$start_l + $k * 0.1" |bc)
```

```
right=$(echo "$left + 0.1" |bc)
bottom=$(echo "${start_b} + $1 * 0.1" |bc)
top=$(echo "$bottom + 0.1" |bc)
sub osmconvert input.osm -b=${left},${bottom},${right},${top} --complete-ways -
o=output${i}${j}${k}${l}.osm
done
done
```

將高度併入build2D檔案內

```
for i in $(ls output??pnt.csv);do j=${i/pnt/bld};sub python join.py $i $j;done
```

```
$ cat do_join.cs
cd sub$1
for i in $(ls final*);do python ../../join_Alt_osmBld.py $i;done
cd ..
```