Chapter 7 - Measure primary characters

November 10, 2020

Input is a `geometry` from previous notebook.

This notebook measures 74 primary morphometric characters. Requires `momepy` 0.2.1 or newer.

It does save intermediate parquet files as a backup.

[ ]:
```
import geopandas as gpd
import momepy as mm
import libpysal
import pandas as pd
```

[ ]:
```
path = 'files/geometry.gpkg'

tess = gpd.read_file(path, layer="tessellation")
blg = gpd.read_file(path, layer="buildings")
blocks = gpd.read_file(path, layer='blocks')
```

Get building height. Use x if it is more than 3, otherwise use 3m. `'roof-0.75'` is an input form 3dBAG data.

[ ]:
```
blg['sdbHei'] = blg['roof-0.75'].apply(lambda x: x if x > 3 else 3)
```

0.1 Measure characters

[ ]:
```
blg['sdbAre'] = mm.Area(blg).series
blg['sdbVol'] = mm.Volume(blg, 'sdbHei', 'sdbAre').series
blg['sdbPer'] = mm.Perimeter(blg).series
blg['sdbCoA'] = mm.CourtyardArea(blg, 'sdbAre').series

blg['ssbFoF'] = mm.FormFactor(blg, 'sdbVol', 'sdbAre').series
blg['ssbVFR'] = mm.VolumeFacadeRatio(blg, 'sdbHei', 'sdbVol', 'sdbPer').series
blg['ssbCCo'] = mm.CircularCompactness(blg, 'sdbAre').series
blg['ssbCor'] = mm.Corners(blg).series
blg['ssbSqu'] = mm.Squareness(blg).series
blg['ssbERI'] = mm.EquivalentRectangularIndex(blg, 'sdbAre', 'sdbPer').series
blg['ssbElo'] = mm.Elongation(blg).series
```

[ ]:
```
cencon = mm.CentroidCorners(blg)
blg['ssbCCM'] = cencon.mean
```

```
blg['ssbCCD'] = cencon.std
```

[ ]:
```
blg['stbOri'] = mm.Orientation(blg).series

tess['stcOri'] = mm.Orientation(tess).series
blg['stbCeA'] = mm.CellAlignment(blg, tess, 'stbOri', 'stcOri', 'uID', 'uID').
 ↪series
```

[ ]:
```
tess['sdcLAL'] = mm.LongestAxisLength(tess).series
tess['sdcAre'] = mm.Area(tess).series
tess['sscCCo'] = mm.CircularCompactness(tess, 'sdcAre').series
tess['sscERI'] = mm.EquivalentRectangularIndex(tess, 'sdcAre').series

tess['sicCAR'] = mm.AreaRatio(tess, blg, 'sdcAre', 'sdbAre', 'uID').series
tess['sicFAR'] = mm.AreaRatio(tess, blg, 'sdcAre', 'HPP', 'uID').series
```

[ ]:
```
blg["mtbSWR"] = mm.SharedWallsRatio(blg, "uID", "sdbPer").series

queen_1 = libpysal.weights.contiguity.Queen.from_dataframe(tess, ids="uID")

blg["mtbAli"] = mm.Alignment(blg, queen_1, "uID", "stbOri").series
blg["mtbNDi"] = mm.NeighborDistance(blg, queen_1, "uID").series
tess["mtcWNe"] = mm.Neighbors(tess, queen_1, "uID", weighted=True).series
tess["mdcAre"] = mm.CoveredArea(tess, queen_1, "uID").series
```

[ ]:
```
blg_q1 = libpysal.weights.contiguity.Queen.from_dataframe(blg)

blg["libNCo"] = mm.Courtyards(blg, "bID", blg_q1).series
blg["ldbPWL"] = mm.PerimeterWall(blg, blg_q1).series

blocks["ldkAre"] = mm.Area(blocks).series
blocks["ldkPer"] = mm.Perimeter(blocks).series
blocks["lskCCo"] = mm.CircularCompactness(blocks, "ldkAre").series
blocks["lskERI"] = mm.EquivalentRectangularIndex(blocks, "ldkAre", "ldkPer").
 ↪series
blocks["lskCWA"] = mm.CompactnessWeightedAxis(blocks, "ldkAre", "ldkPer").series
blocks["ltkOri"] = mm.Orientation(blocks).series

blo_q1 = libpysal.weights.contiguity.Queen.from_dataframe(blocks, ids="bID")

blocks["ltkWNB"] = mm.Neighbors(blocks, blo_q1, "bID", weighted=True).series
blocks["likWBB"] = mm.Count(blocks, blg, "bID", "bID", weighted=True).series
```

Save data to parquets.

[ ]:
```
tess.drop(columns='geometry').to_parquet('files/tess_data.parquet')
blg.drop(columns='geometry').to_parquet('files/blg_data.parquet')
```

```
blocks.drop(columns='geometry').to_parquet('files/blocks_data.parquet')
```

[ ]:
```
queen3 = mm.sw_high(k=3, weights=queen_1)
queen1 = queen_1
blg_queen = blg_q1

blg['ltbIBD'] = mm.MeanInterbuildingDistance(blg, queen1, 'uID', queen3).series
blg['ltcBuA'] = mm.BuildingAdjacency(blg, queen3, 'uID', blg_queen).series
```

[ ]:
```
tess = tess.merge(blg[['HPP', 'uID']], on='uID', how='left')
tess['licGDe'] = mm.Density(tess, 'HPP', queen3, 'uID', 'sdcAre').series
tess = tess.drop(columns='HPP')
tess['ltcWRB'] = mm.BlocksCount(tess, 'bID', queen3, 'uID').series
```

Save data to parquets and spatial weights matrices to gal files.

[ ]:
```
tess.drop(columns='geometry').to_parquet('files/tess_data.parquet')
blg.drop(columns='geometry').to_parquet('files/blg_data.parquet')

fo = libpysal.io.open('files/AMSqueen1.gal', 'w')
fo.write(queen1)
fo.close()

fo = libpysal.io.open('files/AMSqueen3.gal', 'w')
fo.write(queen3)
fo.close()

fo = libpysal.io.open('files/AMSblg_queen.gal', 'w')
fo.write(blg_queen)
fo.close()
```

[ ]:
```
streets = gpd.read_file(path, layer="network")
```

[ ]:
```
streets["sdsLen"] = mm.Perimeter(streets).series
tess["stcSAl"] = mm.StreetAlignment(tess, streets, "stcOri", "nID").series
blg["stbSAl"] = mm.StreetAlignment(blg, streets, "stbOri", "nID").series

profile = mm.StreetProfile(streets, blg, heights='sdbHei', distance=3)
streets["sdsSPW"] = profile.w
streets["sdsSPH"] = profile.h
streets["sdsSPR"] = profile.p
streets["sdsSPO"] = profile.o
streets["sdsSWD"] = profile.wd
streets["sdsSHD"] = profile.hd

streets["sssLin"] = mm.Linearity(streets).series
```

```
streets["sdsAre"] = mm.Reached(streets, tess, "nID", "nID", mode="sum",␣
 ↪values="sdcAre").series
streets["sisBpM"] = mm.Count(streets, blg, "nID", "nID", weighted=True).series
```

[ ]:
```
tess.drop(columns='geometry').to_parquet('files/UAP/tess_data.parquet')
blg.drop(columns='geometry').to_parquet('files/UAP/blg_data.parquet')
streets.drop(columns='geometry').to_parquet('files/UAP/streets_data.parquet')
```

[ ]:
```
str_q1 = libpysal.weights.contiguity.Queen.from_dataframe(streets)

streets["misRea"] = mm.Reached(
    streets, tess, "nID", "nID", spatial_weights=str_q1, mode="count"
).series
streets["mdsAre"] = mm.Reached(streets, tess, "nID", "nID",␣
 ↪spatial_weights=str_q1,
                               mode="sum").series
```

[ ]:
```
graph = mm.gdf_to_nx(streets)

print("node degree")
graph = mm.node_degree(graph)

print("subgraph")
graph = mm.subgraph(
    graph,
    radius=5,
    meshedness=True,
    cds_length=False,
    mode="sum",
    degree="degree",
    length="mm_len",
    mean_node_degree=False,
    proportion={0: True, 3: True, 4: True},
    cyclomatic=False,
    edge_node_ratio=False,
    gamma=False,
    local_closeness=True,
    closeness_weight="mm_len",
)
print("cds length")
graph = mm.cds_length(graph, radius=3, name="ldsCDL")

print("clustering")
graph = mm.clustering(graph, name="xcnSCl")

print("mean_node_dist")
graph = mm.mean_node_dist(graph, name="mtdMDi")
```

```
nodes, edges, sw = mm.nx_to_gdf(graph, spatial_weights=True)

print("saving")
nodes.to_file(path, layer="nodes", driver="GPKG")
edges.to_file(path, layer="edges", driver="GPKG")

fo = libpysal.io.open("files/UAP/UAPnodes.gal", "w")
fo.write(sw)
fo.close()

edges_w3 = mm.sw_high(k=3, gdf=edges)
edges["ldsMSL"] = mm.SegmentsLength(edges, spatial_weights=edges_w3, mean=True).
 ↪series

edges["ldsRea"] = mm.Reached(edges, tess, "nID", "nID",␣
 ↪spatial_weights=edges_w3).series
edges["ldsRea"] = mm.Reached(
    edges, tess, "nID", "nID", spatial_weights=edges_w3, mode="sum",␣
 ↪values="sdcAre"
).series

nodes_w5 = mm.sw_high(k=5, weights=sw)
nodes["lddNDe"] = mm.NodeDensity(nodes, edges, nodes_w5).series
nodes["linWID"] = mm.NodeDensity(
    nodes, edges, nodes_w5, weighted=True, node_degree="degree"
).series

blg["nodeID"] = mm.get_node_id(blg, nodes, edges, "nodeID", "nID")
tess = tess.merge(blg[["uID", "nodeID"]], on="uID", how="left")

nodes_w3 = mm.sw_high(k=3, weights=sw)

nodes["lddRea"] = mm.Reached(nodes, tess, "nodeID", "nodeID", nodes_w3).series
nodes["lddARe"] = mm.Reached(
    nodes, tess, "nodeID", "nodeID", nodes_w3, mode="sum", values="sdcAre"
).series

nodes["sddAre"] = mm.Reached(
    nodes, tess, "nodeID", "nodeID", mode="sum", values="sdcAre"
).series
nodes["midRea"] = mm.Reached(nodes, tess, "nodeID", "nodeID",␣
 ↪spatial_weights=sw).series
nodes["midAre"] = mm.Reached(
    nodes, tess, "nodeID", "nodeID", spatial_weights=sw, mode="sum",␣
 ↪values="sdcAre"
).series
```

```
nodes.rename(
    columns={
        "degree": "mtdDeg",
        "meshedness": "lcdMes",
        "local_closeness": "lcnClo",
        "proportion_3": "linP3W",
        "proportion_4": "linP4W",
        "proportion_0": "linPDE",
    }, inplace=True
)
```

[ ]:
```
tess.drop(columns='geometry').to_parquet('files/UAP/tess_data.parquet')
blg.drop(columns='geometry').to_parquet('files/UAP/blg_data.parquet')
nodes.drop(columns='geometry').to_parquet('files/UAP/nodes_data.parquet')
edges.drop(columns='geometry').to_parquet('files/UAP/edges_data.parquet')
```

[ ]:
```
merged = tess.merge(blg.drop(columns=['nID', 'bID', 'nodeID', 'geometry']),␣
↪on='uID')
merged = merged.merge(blocks.drop(columns='geometry'), on='bID', how='left')
merged = merged.merge(edges.drop(columns='geometry'), on='nID', how='left')
merged = merged.merge(nodes.drop(columns='geometry'), on='nodeID', how='left')
```

Clean columns to keep only data.

[ ]:
```
primary = merged.drop(columns=['nID', 'bID', 'nodeID', 'mm_len', 'cdsbool',
                               'node_start', 'node_end', 'geometry', 'greedy',
                               ])
```

Final save of data.

[ ]:
```
primary.to_parquet('files/primary.parquet')
```