

Final Assignment

Jin Xu

2019/5/2

Introduction

In this assignment, Python script is created for exploring the relation of spatial distributions of trees, buildings, and street light poles in relation to litter using grid. The major steps are listed as follow:

- Obtaining data from its URL.
- Spatial analysis: feature to point, projection, creating grid, intersection, statistical analysis, table join.
- Calculating correlation coefficient.
- Visualization.

In some cases when proper geography (e.g. neighborhoods or census tracts that divide a city) is hard to find, grids can be created to divide a larger space into several small parts with equal sizes. In this way, the spatial distributions of two interested map features can be visualized, and the relation between them can be quantified with correlation coefficients.

The Python script is created for these purposes. It first creates a grid covering the area of Philadelphia. Then points of trees, buildings, light poles, and litter index are intersected with the grid to gain the cell id of grid for table join. They are later summarized into tables, and the results are joined back to the grid for visualization and calculation for correlation coefficient. This script generates a report in .txt format for correlation coefficients between the spatial distribution of trees-litter, buildings-litter, and street light poles-litter, as well as four maps in .pdf format visualizing the distribution of trees, buildings, light poles, and litter index.

Two paths are required as system argument input. One for arcpy work-space for storing files generated from the process (e.g. G:/FA). Another one for a folder saving the final text report and maps (e.g. G:/FA/result).

Obtaining Data

For the purpose of this assignment, all data for analysis are originally downloaded from Open Data Philly(<https://www.opendataphilly.org/>). Mapping file and layer files for visualization are manually created in ArcMap and uploaded to a previously

created repository on my GitHub

Gist(https://github.com/XuJinGabie/FA_GUS_8066_public) for download. The files downloaded with Python scripts are as follow:

- **Census Tract:** Polygon in .shp file. It is only applied for getting the coordinate of Philadelphia bounding box for creating grid. Although creating grids only need the coordinates of two opposite points of the bounding box, census tract is used here for getting these coordinates since they are unknown. It is downloaded from:
http://data.phl.opendata.arcgis.com/datasets/8bc0786524a4486bb3cf0f9862ad0fbf_0.zip
- **Building Footprint:** Polygon in .shp file. The number of building in each cell of grid is counted as the number of building centroids. The polygons will be turned into points for eliminating double counts. It's downloaded from:
http://data.phl.opendata.arcgis.com/datasets/ab9e89e1273f445bb265846c90b38a96_0.zip
- **Tree Inventory:** Point in .shp file. All tree locations in Philadelphia. It is for counting the number of trees in each cell of grid. It is downloaded from:
http://data.phl.opendata.arcgis.com/datasets/957f032f9c874327a1ad800abd887d17_0.zip
- **Street Pole:** Point in .shp file. All street light pole locations in Philadelphia. It is counted for the number in each cell of grid. It is downloaded from:
http://data.phl.opendata.arcgis.com/datasets/9059a7546b6a4d658bef9ce9c84e4b03_0.zip
- **Litter Index:** Point in .shp file. It is the survey points measuring the amount of litter, and the average litter index in each cell of grid is calculated. Its rating column contains the litter index ranging from 1 to 4, where 1 meaning minimal litter and 4 meaning extremely littered. The original link on Open Data Philly is https://phl.carto.com/api/v2/sql?q=SELECT+*+FROM+litter_index_survey&filename=litter_index_survey&format=shp&skipfields=cartodb_id. But it returns http error "400 Bad Request" when trying to download it with Python. It might means that I do not have full access to the URL. So it is downloaded manually, unzipped, re-zipped with mapping files and stored in the GitHub Gist.
- **Map Files:** The .mxd file contains properly layouted title, legend, scale, and north arrow. There are no layers in this file. There are also 4 .lyr layer files for visualizing the number of trees, buildings, light poles, and average litter index with graduate color. Classification method is natural break, and they all excludes the 0 values. These files are created manually and they are zipped with litter index files. The zipped file is uploaded to my GitHub Gist, and can be found with this URL:
https://github.com/XuJinGabie/FA_GUS_8066_public/blob/master/FA_original

.zip. But this URL need to be modify a little bit for Python download:
https://raw.githubusercontent.com/XuJinGabe/FA_GUS_8066_public/master/FA_original.zip.

An user-defined function is created in Python for download zip file from its URL, unzip the file to the folder storing all intermediate files, and return the .shp file-name.

```
def fetch_unzip(url, out_dir):  
    '''  
    x = fetch_unzip(download_url, unzip_path)  
  
    Function:  
        fetch zipped file from url  
        save to out_dir  
        unzip to out_dir  
  
    Return:  
        .shp filename in the .zip  
    '''  
    # fetch data from url  
    print("Fetching data from {}".format(url))  
  
    response = urllib2.urlopen(url)  
    bin_content = response.read()  
    response.close()  
  
    out_file = out_dir + os.path.basename(url)  
    with open(out_file, "wb") as outf:  
        outf.write(bin_content)  
  
    # unzip data  
    with zipfile.ZipFile(out_file, "r") as zip_obj:  
        zip_obj.extractall(out_dir)  
        namelist = zip_obj.namelist()  
        shp_name = []  
  
        for filename in namelist:  
            #print("Unzipping: {}".format(filename))  
            # recording shpfile name  
            if filename[-3:] == ".shp":  
                shp_name.append(filename)  
  
        # return shp filename  
        return shp_name[0]  
  
    print("Complete: Fetch and Unzip")
```

- It requires os, urllib2 and zipfile packages.
- url argument should be a string.
- out_dir argument is the path for storing intermediate files imputed as system argument.
- shp_name[0]: Return the first .shp file-name for spatial analysis in the following steps. Since each .zip file downloaded from the above URLs has only one .shp file, the shp_name[] list only have 1 string item and it is the file-name that will be used in the following steps.

The function is called as follow:

```
# Data
tract_org = fetch_unzip(tract_url, outdir)
building_org = fetch_unzip(building_url, outdir)
tree_org = fetch_unzip(tree_url, outdir)
light_org = fetch_unzip(light_url, outdir)

litter_org= fetch_unzip(original_url, outdir)
```

The files needed for spatial analysis and mapping are now downloaded and unzipped to the arcpy work-space folder with 5 variables recording the .shp file-names.

Since the building footprints are originally polygons, it will be counted for multiple times in grids if the polygon cross the boundaries of two cells in the grid. The feature to point function are needed for getting building centroid. Moreover, the originally downloaded files are not projected. For the purpose of using loops in projection and keep tract of shapefile names, a name-lists of all downloaded files are created. The code is shown as below:

```
# turn building into point
building_point = "Building.shp"
arcpy.FeatureToPoint_management(building_org, building_point, "CENTROID")

# namelist for unprojected filenames
orgs = [tract_org, building_point, tree_org, light_org, litter_org]

del building_point
```

Now buildings, trees, light poles, and litter index are stored as points, and their .shp file-names are recorded together with census tract in orgs list.

Spatial Analysis Process

Projection

With a list storing the .shp file-names of non-projected data called `orgs`, the projection can now be done with loops:

```
# PROCESS: projection
# namelist for object name
objs = ["tract", "building", "tree", "light", "litter"]

# dictionary for object-projected filename
proj_dic = {}

# projection
for i, org in enumerate(orgs):
    key = objs[i]
    value = "P_" + org
    arcpy.Project_management(org, value, coordinate)
    proj_dic[key] = value

#print(proj_dic)

# delete unprojected filename variables, temp variables
del i
del key
del value
del org
del orgs
del tract_org
del building_org
del tree_org
del light_org
del litter_org
```

- `objs`: A list of all objects called `objs` using the order of the unprojected file-name list `orgs` in order to create a dictionary for storing the objects and their corresponding projected shapefile names.
- `proj_dic`: The dictionary storing objects and their corresponding projected shapefile names. It uses the strings in the `objs` as its key, and the projected file-name "P_" + `org` as value. For example, the unprojected shapefile name for census tract is `Census_Tracts_2010.shp`, and the projected shapefile name of it is `P_Census_Tracts_2010.shp`. It is stored in the `proj_dic` as "tract": "P_Census_Tracts_2010.shp".

Creating Grid

The grid can be created in ArcGIS using Create Fishnet tool. In arcpy, arguments required for this tool are: the start coordinate, end coordinate, coordinate for turn, cell height, cell width, number of rows and number of columns. Cell height and cell width can be any number. The number of rows and number of columns can be calculated automatically based on cell height and width if it is imputed as 0. But the coordinate is now unknown. Therefore, the coordinates are get with the extent of census tract, i.e. the extent of Philadelphia, before creating grid:

```
# PROCESS: create grid
# get extent
desc = arcpy.Describe(proj_dic["tract"])
# set xmin and ymin as the begining coordinate
begin_coord = str(desc.extent.lowerLeft)
# set 0 point and direction_coord is the angle for turning fishnet, no
turn
direction_coord = str(desc.extent.XMin) + " " + str(desc.extent.YMax +
10)
# set xmax and ymax as opposit coordinate
end_coord = str(desc.extent.upperRight)
```

- begin_coord: The start point. It is assigned as the southwest corner of the bounding box of Philadelphia.
- end_coord: The ending point, it is the northeast corner.
- direction_coord: The coordinate for turn. It is to define the angle for turning the grid. The grid will be turned with the angle between y-axis and the line between begin_coord and direction_coord. In this case, the grid do not need to be turned, so X coordinate in direction_coord is equal to the x coordinate in begin_coord.

With these variables, a grid with the cell size of 1kmX1km is created as follow:

```
# grid filename
grid = "Grid.shp"

# create 1000X1000m grid
arcpy.CreateFishnet_management(grid, begin_coord, direction_coord,"1000
", "1000", "0", "0", end_coord, "NO_LABELS", "#", "POLYGON")

# delete temporal variables
del desc
del begin_coord
del direction_coord
del end_coord
```

- NO_LABELS: Create a grid without label points in the center.
- POLYGON: It will create a grid where each cell is a polygon.

The created grid is stored in the arcpy work-space imputed as system argument under the file-name Grid.shp. In its attribute table, the Primary Key is its FID, all other fields storing the data of the number of buildings, trees, light poles, and average litter index that are created in the next step will be joined to its attribute table using FID.

Spatial Distribution of Points

The following code aims at calculating the number of buildings, trees, light poles, and the average litter index in each cell of the grid, then join all these fields back to the grid based on the cell id.

```
# PROCESS: MAIN ANALYSIS
# namelist for analyst object
analyst_objs = ["building", "tree", "light", "litter"]

# dictionary for object-column name
field_dic = {}

# count for how many time have calculated for point number
count_freq = 0

# proj_dic[anl_obj] = filename of object for analyst
for anl_obj in analyst_objs:

    # Variables

    intersect_point = "Inter_" + proj_dic[anl_obj]
    count_table = "Sum_" + anl_obj + ".dbf"

    print("Analyzing: {0}".format(proj_dic[anl_obj]))

    # Intersect for grid id
    arcpy.Intersect_analysis([grid,proj_dic[anl_obj]], intersect_point,
"ALL", "", "INPUT")

    if anl_obj == "litter":
        # average Litter index
        arcpy.Statistics_analysis(intersect_point,count_table,[["rating
", "MEAN"]], "FID_Grid")

        # average Litter index column name
        #field_names.append("MEAN_FID")
        key = anl_obj
```

```

        value = "MEAN_ratin"
        field_dic[key] = value
        print("Data in MEAN_FID")

    else:
        # count number of points
        arcpy.Statistics_analysis(intersect_point, count_table, [{"FID", "COUNT"}], "FID_Grid")

        # counted frequency for 1 time
        count_freq += 1

        # first time counting frequency column name
        if count_freq == 1:
            #field_dic.append("FREQUENCY")
            key = anl_obj
            value = "FREQUENCY"
            field_dic[key] = value

            print("Data in FREQUENCY")

        # after first time counting frequency column name
        else:
            #field_dic.append("FREQUENC_{0}".format(i))
            key = anl_obj
            value = "FREQUENC_{0}".format(count_freq-1)
            field_dic[key] = value

            print("Data in FREQUENC_{0}".format(count_freq-1))

        # Join to grid
        arcpy.JoinField_management(grid, "FID", count_table, "FID_Grid")

        #print("Finish Analyzing {0}".format(proj_dic[anl_obj]))

print("Complete Spatial Analysis")

del anl_obj
del count_freq
del intersect_point
del count_table

```

The analyst_objs is the name-list for all point objects that will be counted for the number or calculated for the mean. It is created for the purpose of loop.

The intersect process of points and grid is for the purpose of getting the cell ids where each point falls in. In this way, each point will be assign to a cell id of their location, and they are summarized by cell ids.

For litter index, the summary is aiming at calculating the average litter index in each cell of the grid. The output table is expected to have a primary key as the cell id, and another field of the average litter index in each cell. Therefore, the field containing litter index rating should be summarized by MEAN and keep the FID_Grid field as the unique value. The field of the average litter index will be named as MEAN_ratin automatically in the output table.

For buildings, trees, and light poles, the summary is aiming at counting the number of them in each cell of grid. The output table is expected to be similar as the average litter index table with a primary key of the cell id, and a field containing the number. Since each points with an unique id is assigned to a cell id, the frequency of the point id is the number of points in each cell. Therefore, the FID field is summarized by COUNT and the FID_Grid is kept as the unique value. The result of COUNT is the same as FREQUENCY because they all represents the number of points with a specific cell id.

But for the number of buildings, trees, and light poles, the number of points are all under the same name as FREQUENCY in separate tables. So this field will be named as FREQUENCY, FREQUENC_1, FREQUENC_2 for the first, second, and third FREQUENCY field that is added to the grid attribute table. In order to distinguish which object is summarized in each FREQUENCY fields, a count_freq variable is created for doing the following function:

- when it is the 1st time to count for the number of points: count_freq = 1, and the field name in the grid attribute table after join is FREQUENCY.
- When it is the n-th time to count for the number of points: count_freq = n, and the field name in the grid attribute table after join is FREQUENC_(n-1).

Meanwhile, a dictionary field_dic is created for storing the object and its corresponding summarized field name. It uses the strings in the analyst_objs as its key, and the field name MEAN_ratin or FREQUENCY as value. Such as the litter index will be stored in the dictionary as "litter": "MEAN_ratin".

The final results are all joined to the grid attribute table for calculating correlation coefficients in the next step.

Final Outputs

Correlation Coefficient

Correlation Coefficient is calculated using numpy array:

```
# PROCESS: Correlation
# result of analysis in
result_dir = sys.argv[2] + "/"

# report for correlation coefficient
corr_file = result_dir + "Correlation Coefficients.txt"
```

```

# object for correlated with Litter
corr_objs = ["building", "tree", "light"]

field_names = list(field_dic.values())
corr_table = arcpy.da.FeatureClassToNumPyArray(grid, field_names)
# data in Litter field
litter_field = corr_table[field_dic["litter"]]

# result for storing coefficient
corr_result = open(corr_file, "a")

# title for content
corr_result.write("Correlation Coefficient for Feature Number-Litter In  
dex: \n")
corr_result.write("\n")

# content
for corr_obj in corr_objs:

    # object's field for correlation
    obj_field = corr_table[field_dic[corr_obj]]
    # correlation: obj-Litter
    obj_lit_corr = np.corrcoef(obj_field, litter_field)
    # correlation coefficient
    obj_lit_cef = obj_lit_corr[0][1]

    # write in txt
    corr_result.write("{0}-litter = {1} \n".format(corr_obj, obj_lit_cef))

corr_result.close()

del corr_obj
del field_names
del corr_table
del litter_field

```

- numpy package is required. It is imported as np.
- corr_objs: Since the correlation coefficient is between the average litter index and the number of buildings, trees, and light poles. This variable is created for calculating the correlation coefficient between litter and all other features using loop.
- field_names: It is a list of all field names that will be converted into a numpy array, i.e. all values in field_dic dictionary.

- `open(txt_file, "a")`: Open the text file and append words after existed words in it.

Within the loop, correlation coefficient matrix is created with `np.corrcoef`. But what is needed here is the correlation coefficient, which is stored in the second position in the first row. Therefore, it is extracted with `obj_lit_corr[0][1]` and written into the text file in the result folder imputed by user as system argument.

Map

User-defined function for mapping shown as below:

```
# data visualization
def visual(in_lyr, in_mxd, field, til_obj, out_path):
    '''
        visual(input_lyr_path, in_mxd, value_field, title_obj, out_png_path
    )

    Function:
        modify the first Layer in thefirst data frame
        output png

    Note:
        in_mxd = arcpy.mapping.MapDocument(mxd_path)
        mxd should be opened
    ...

    # first dataframe
    data_frames = arcpy.mapping.ListDataFrames(in_mxd)
    df = data_frames[0]
    # get first Layer to modify
    lyr_list = arcpy.mapping.ListLayers(in_mxd)
    layer_modify = lyr_list[0]

    src_obj = arcpy.mapping.Layer(in_lyr)
    arcpy.mapping.UpdateLayer(df, layer_modify, src_obj)

    # value field
    layer_modify.symbology.valueField = field
    arcpy.RefreshActiveView()

    # change title and Legend
    elems = arcpy.mapping.ListLayoutElements(in_mxd)
    # finding title, and Legend, only 1 text element/Legend in mxd
    for e in elems:
        if "TEXT_ELEMENT" in e.type:
            title = e
        elif "LEGEND_ELEMENT" in e.type:
            legend = e
```

```

# changing title name
title.text = til_obj
# change legend height
legend.elementHeight = 7

# refresh view
arcpy.RefreshActiveView()

# export
out_file = out_path + til_obj + ".pdf"
arcpy.mapping.ExportToPDF(in_mxd, out_file)

#print("Mapped")

```

- There seems to have a problem with the title map element, and it is not showing the name title. But the only text element in the .mxd file is the title. so title is actually found by the element type TEXT_ELEMENT.
- Same thing happened to the legend, so it is also found by the element type LEGEND_ELEMENT. The size of it is also changed to make it bigger than default size.
-in_lyr: The input layer file previously created. It is unzipped from the file downloaded from GitHub Gist.
- in_mxd: It is also unzipped from the GitHub Gist download.
- field: The field for mapping. In this assignment, all fields to map are in the grid attribute table.
- til_obj: The title name, meaning which object is mapped.

This function requires the .mxd file to be open first as follow. It opens the .mxd file and add the grid .shp file.

```

# PROCESS: Map
# storing everthing generated from thhis script
outdir = arcpy.env.workspace + "/"
# the mxd file is created mannually as blank file with map elements
map_name = "empty.mxd"

# open mxd and data frames
mxd = arcpy.mapping.MapDocument(outdir + map_name)
# 1st dataframe
data_frames = arcpy.mapping.ListDataFrames(mxd)
df = data_frames[0]
# add grid to 1st data frame 1st layer
lyr_file = arcpy.mapping.Layer(grid)
arcpy.mapping.AddLayer(df, lyr_file)

```

Maps can be generated with a loop:

```
# visualization
# input source .lyr, export filename, .lyr manually create, natural break, excluded 0
for k in field_dic.keys():

    src_lyr = outdir + "/" + field_dic[k] + ".lyr"
    field = field_dic[k]

    visual(src_lyr, mxd, field, k, result_dir)

    print("Complete Mapping {0}".format(k))

# save new file
mxd.saveACopy(arcpy.env.workspace + "/" + "new" + map_name)

del mxd
del data_frames
del lyr_file
del df
del src_lyr
del field
```

It creates 4 maps in .pdf for the number of buildings, trees, light poles and average litter index in each cell of grid. The results are in the final result folder with the .txt file for correlation coefficients.

Results

The Python code generates these result files:

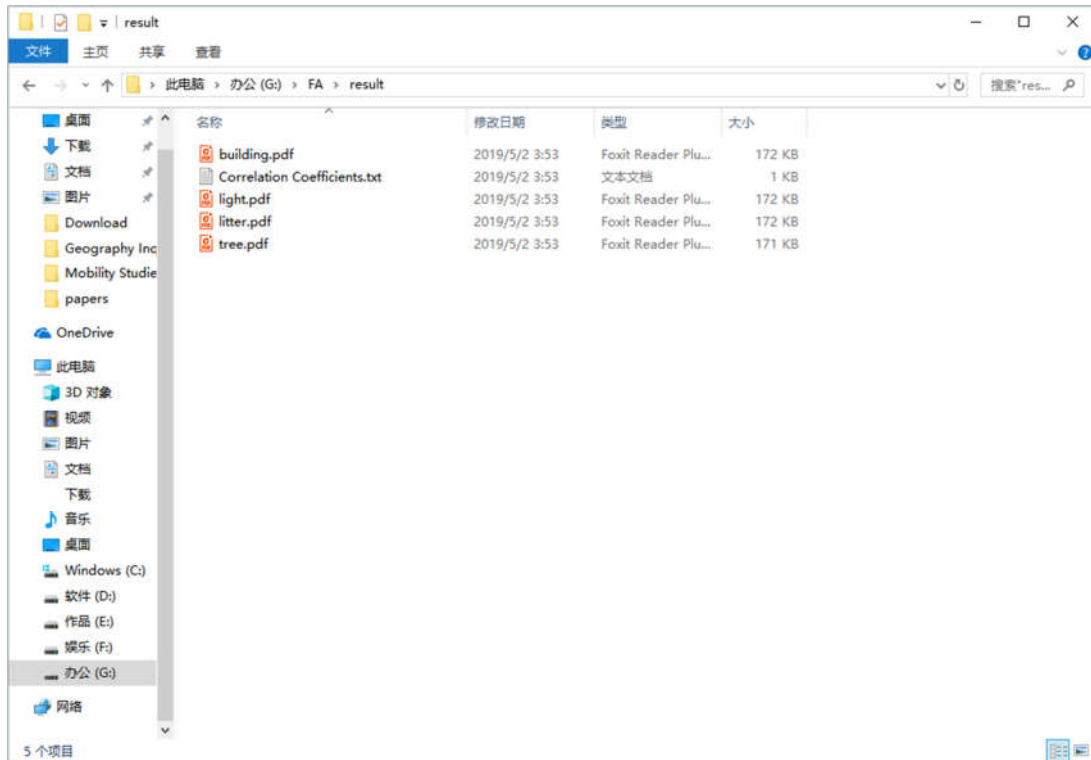
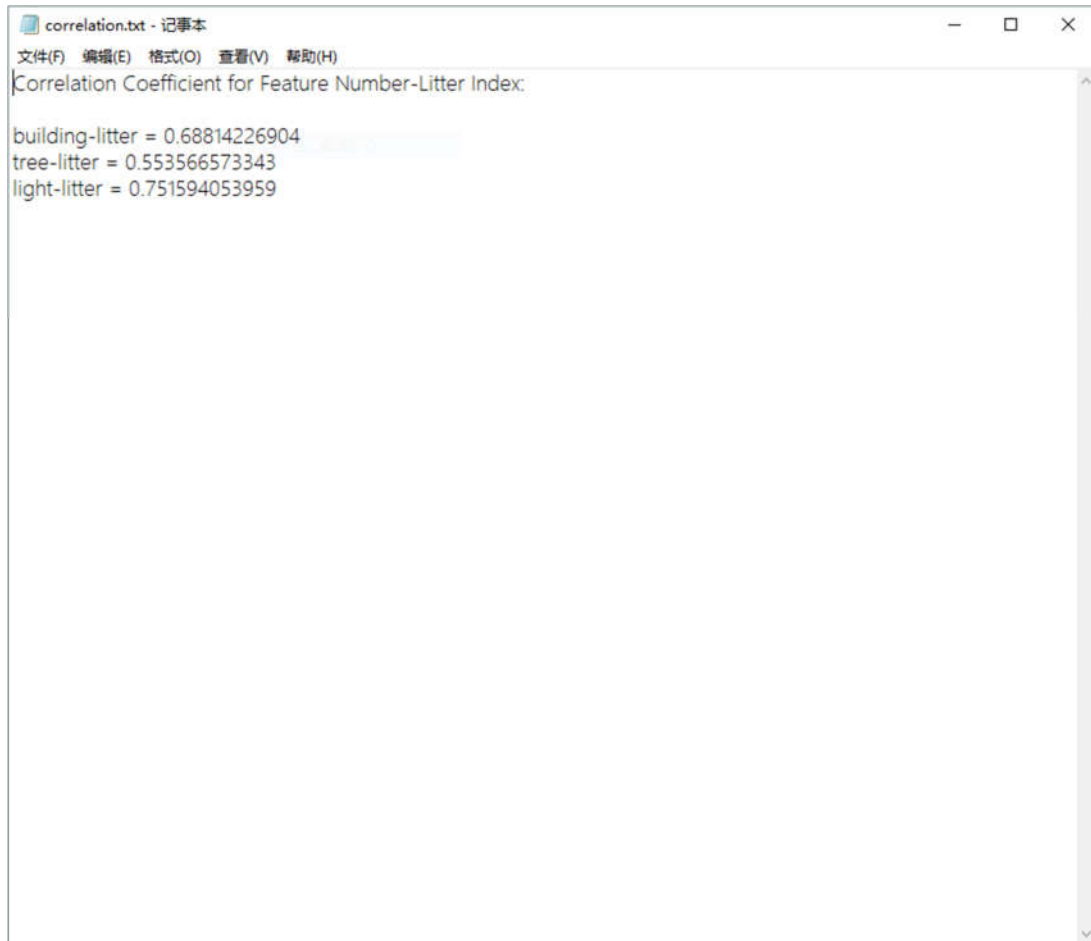


Figure 1. Result Files

The result for correlation is in a text file. The report looks like this:



```
correlation.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
Correlation Coefficient for Feature Number-Litter Index:

building-litter = 0.68814226904
tree-litter = 0.553566573343
light-litter = 0.751594053959
```

Figure 2. Correlation Coefficient Report

The correlation coefficients are between 0.55-0.75, meaning that the number of buildings, trees, light poles are positively related to average litter index at a moderate to strong level. It means that places with more buildings, trees, or street light poles are usually littered more.

These results can also be observed with the maps shown as below:

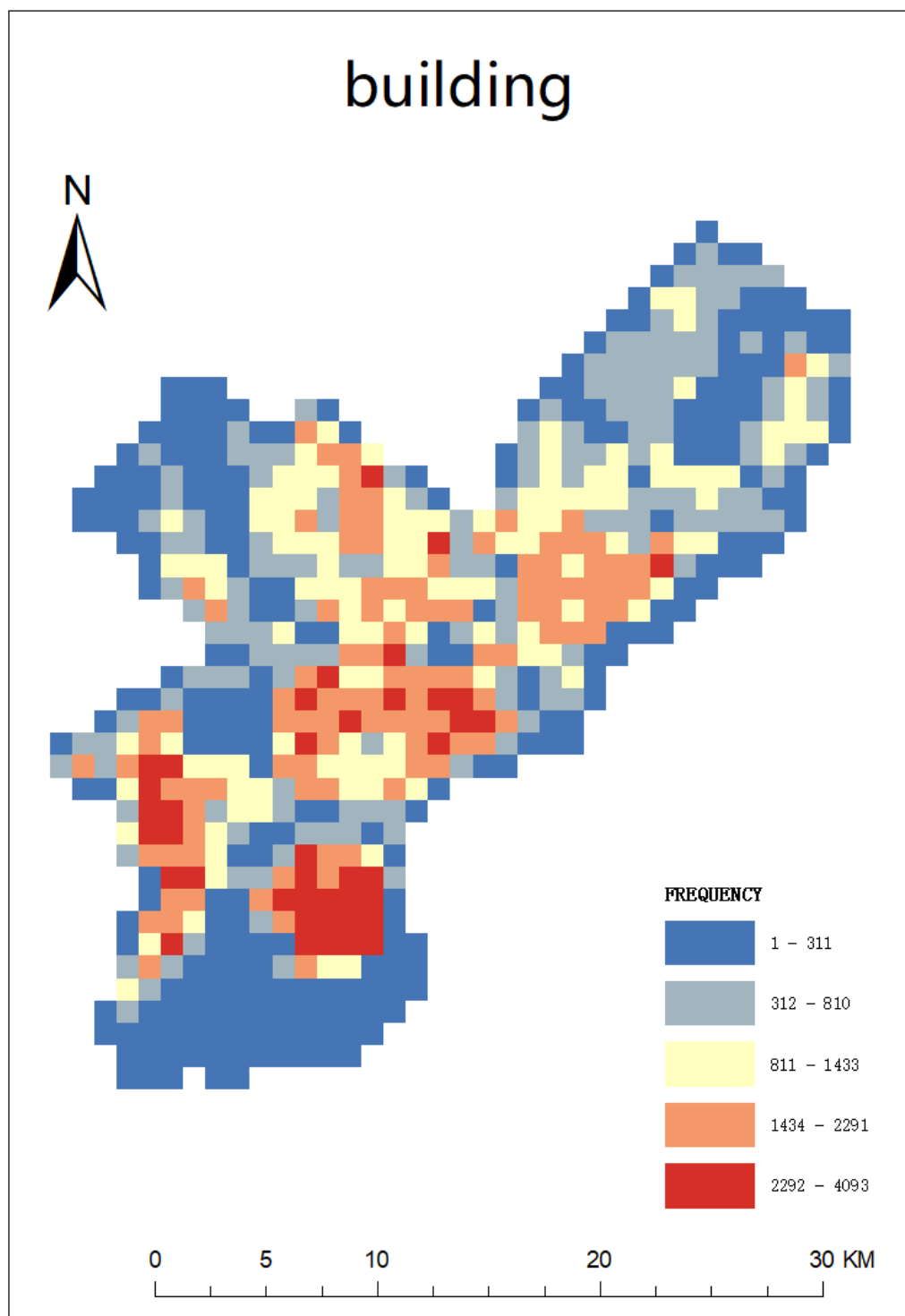


Figure 3. Building Number Distributon

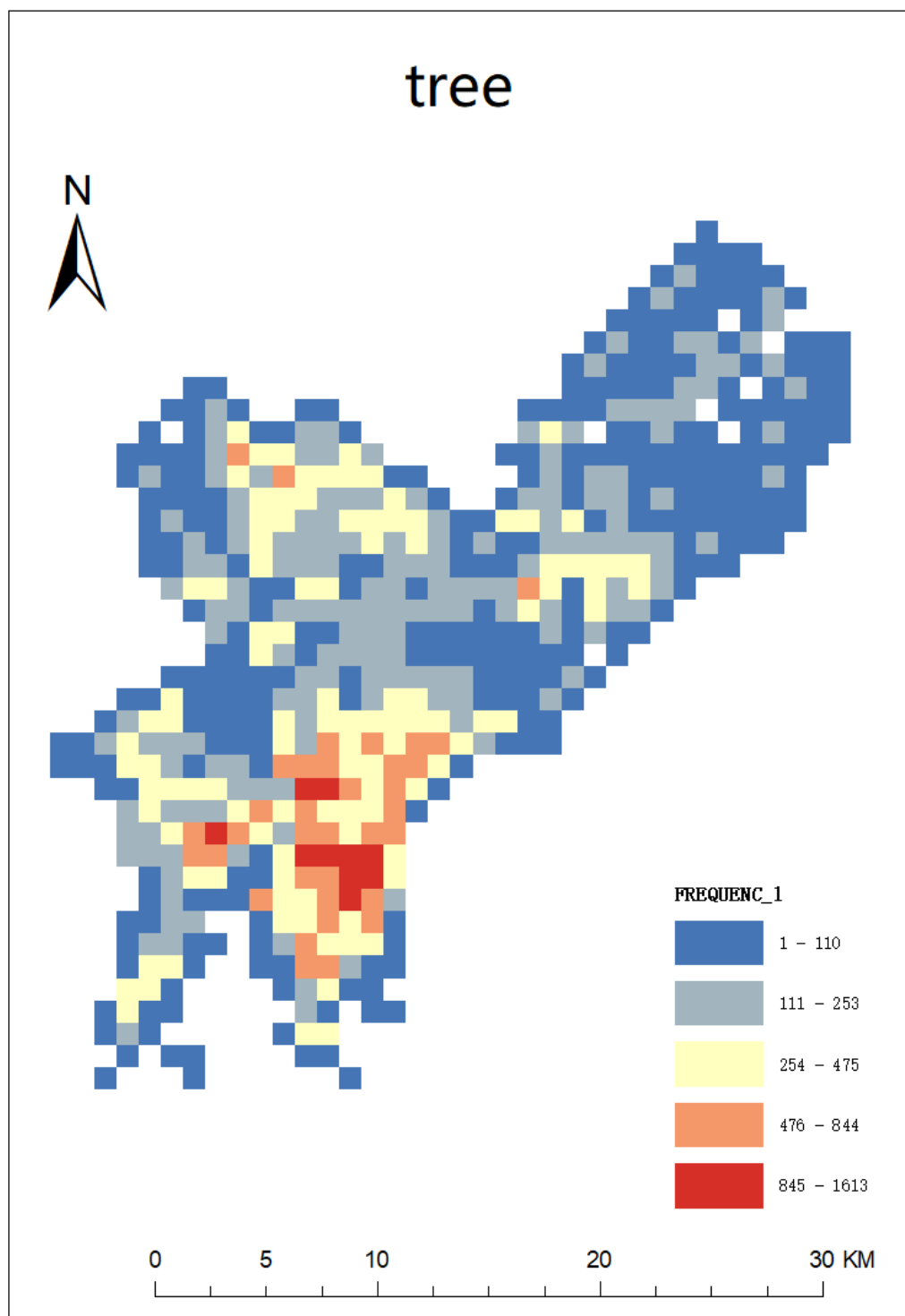


Figure 4. Tree Number Distributon

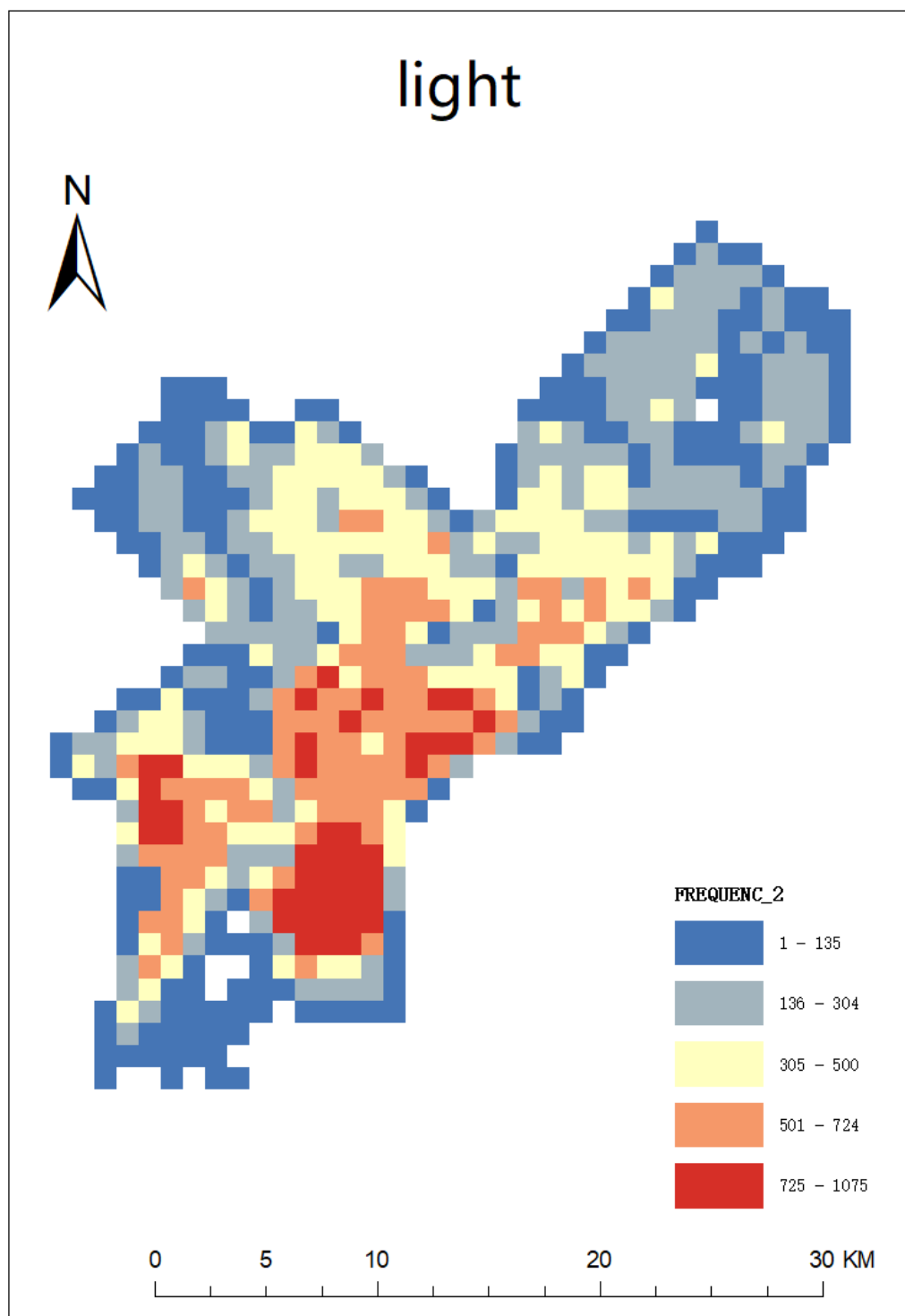


Figure 5. Light Number Distributon

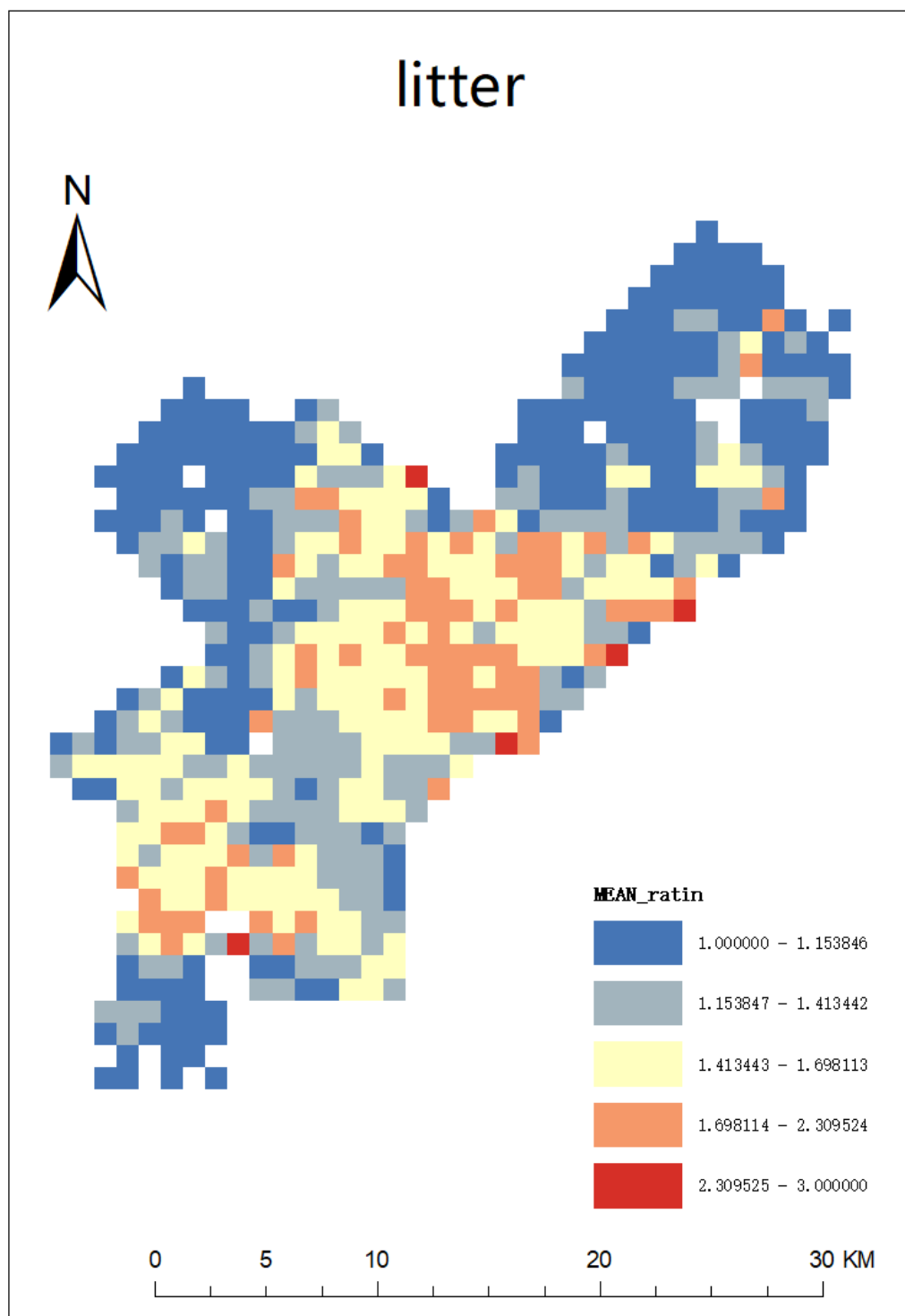


Figure 6. Average Litter Index Distributon