MAPPING LAST-MILE WAREHOUSES IN NEW YORK CITY

By: Kevin Garcia

Background

- Increase in online shopping in recent years.
- Increase in last-mile warehouses in New York.
- Over 2 million packages delivered in New York City daily.
- Last-mile warehouses can only be constructed in manufacturing and C8 districts.

Question

Given the rise in last-mile warehouses and that these facilities can only be built in certain areas, is there a clustering of last-mile warehouses?

Data Gathering

1. Find last-mile warehouses in New York City.
2. Create a spreadsheet of these facilities and load it into the notebook.

```
#Loading the CSV that contains identified last-mile warehouses into a dataframe
df = pd.read_csv(r"G:\My Drive\INFO-615  Spatial Statistics GIS\Final Assignment\DATA\NYCLast-MileWarehouses.csv")
```

Data Gathering

1. Gather the borough boundaries

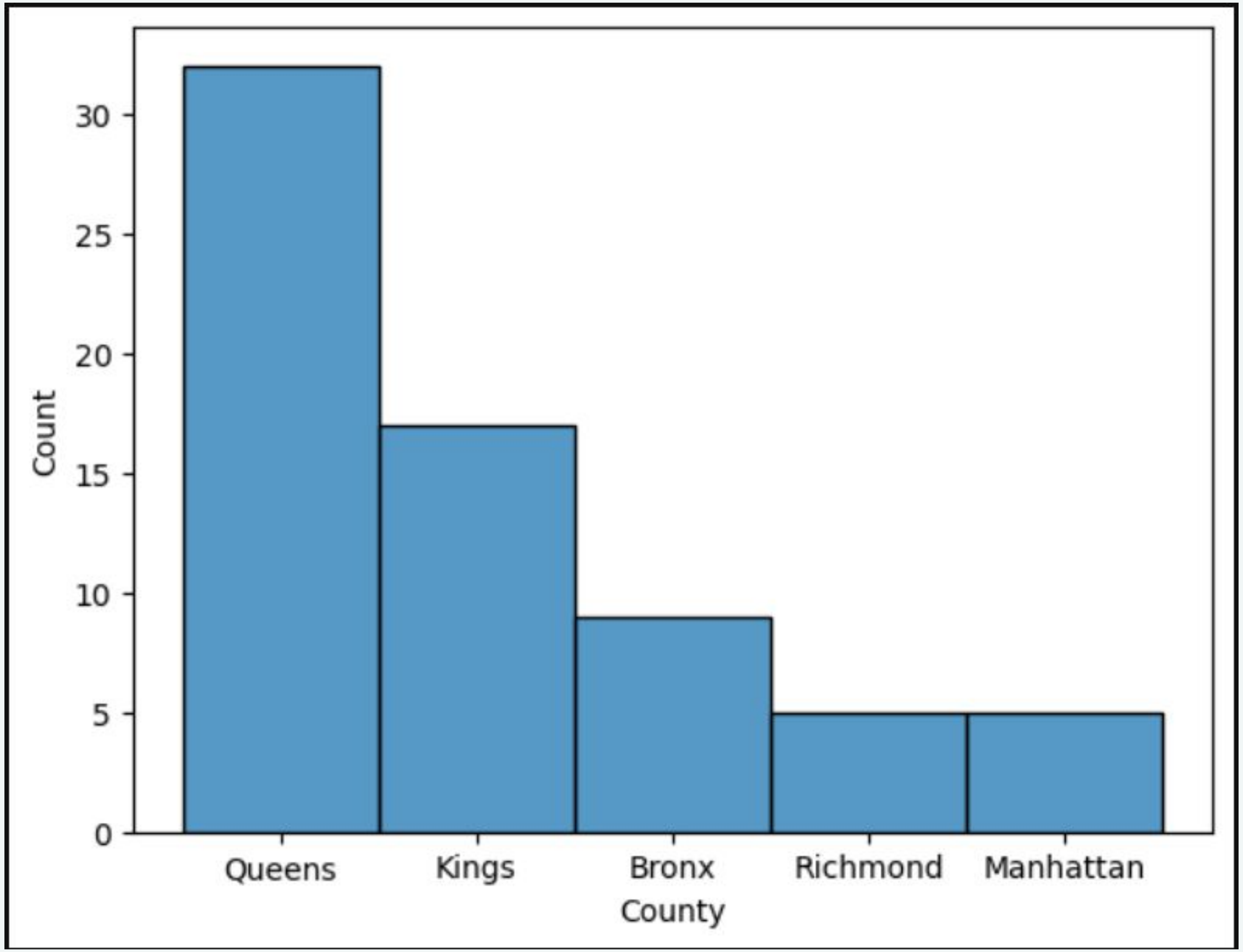
2. Gather the zoning map.

```
#Getting the boundaries for the five boroughs
borosURL = "https://data.cityofnewyork.us/api/geospatial/tqmj-j8zm?method=export&format=GeoJSON"
boros = gpd.read_file(borosURL)
```

```
#This data set consists of 6 classes of zoning features: zoning districts, special purpose districts,
#special purpose district subdistricts, limited height districts, commercial overlay districts,
#and zoning map amendments.
zonesURL = "https://data.cityofnewyork.us/api/geospatial/kdig-pewd?method=export&format=GeoJSON"
zones = gpd.read_file(zonesURL)
```

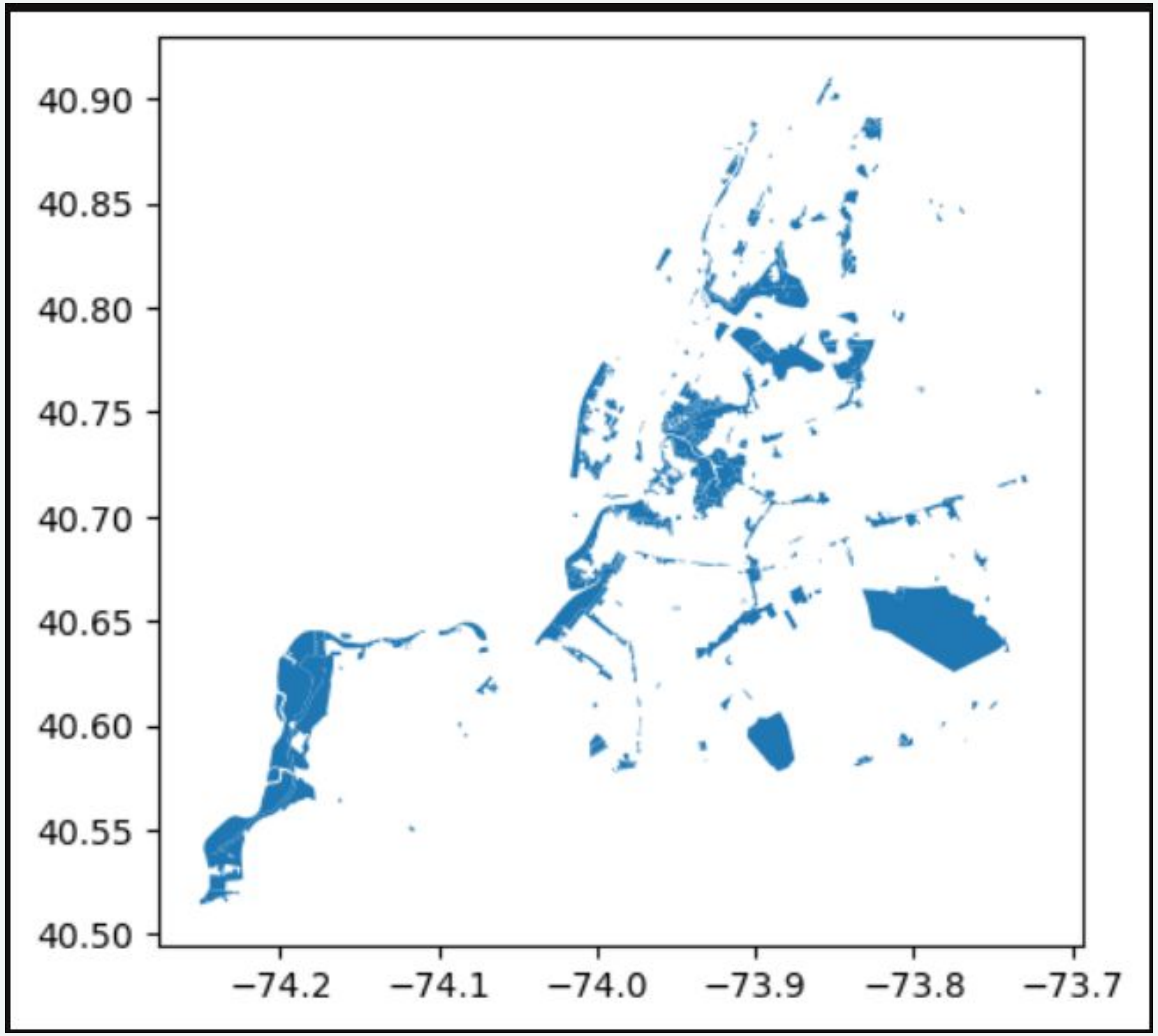
Data Exploration

```
sns.histplot(gdf['County'], stat = 'count')
```

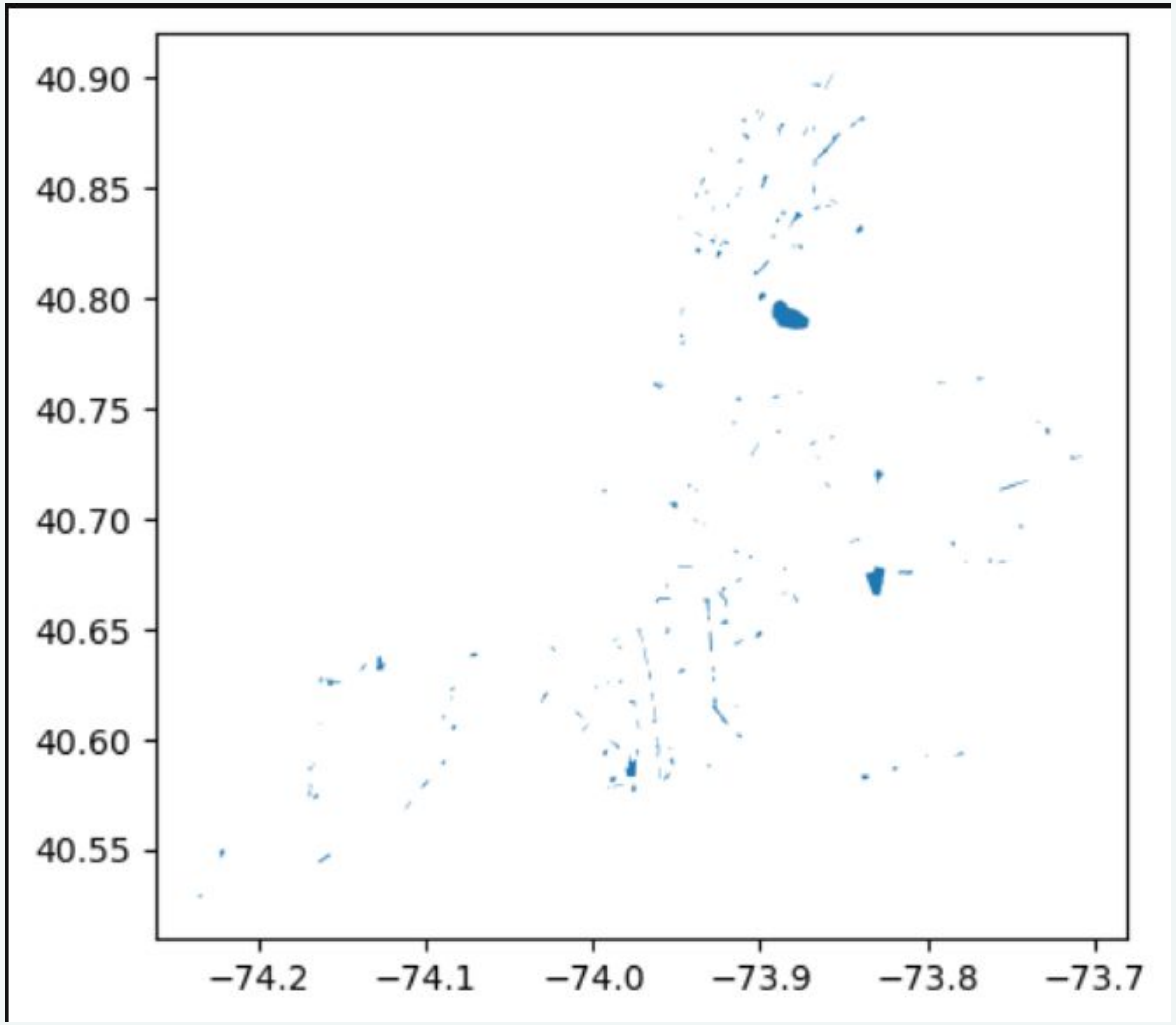


Data Transformation

```
#Create geodataframe of manufacturing zones
zonesM = zones.copy()[zones['zonedist'].str.contains('M', regex=True, na=False)]
zonesM.plot();
```



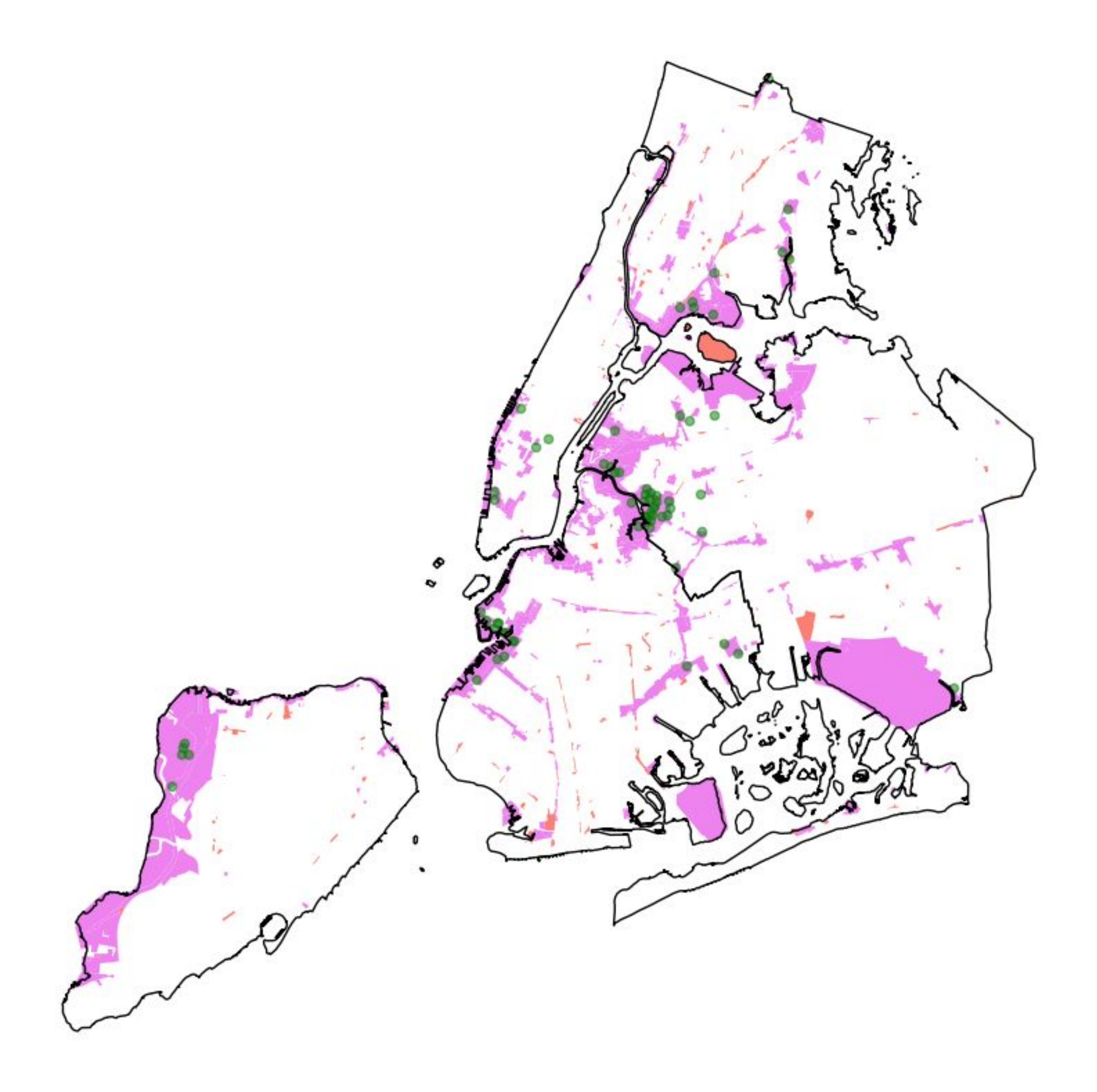

```
#Create geodataframe of C8 zones
zonesC8 = zones.copy()[zones['zonedist'].str.contains('C8', regex=True, na=False)]
zonesC8.plot();
```



Data Transformation

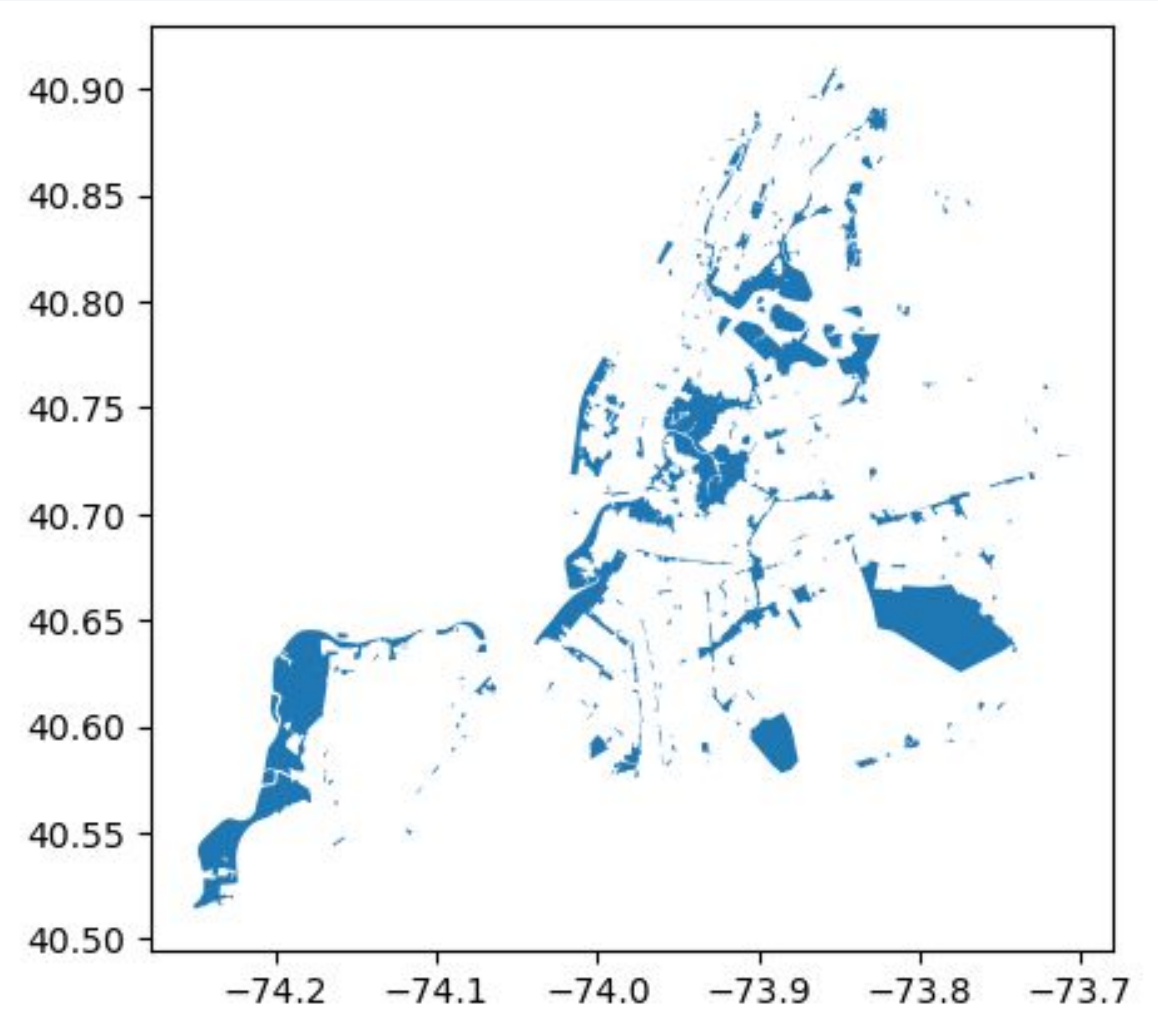
```
#Mapping the identified last-mile warehouses over the manufacturing zones and C8 zones.
fig, ax = plt.subplots(figsize = (10,10))

zonesC8.plot(facecolor = 'salmon', edgecolor = 'none', linewidth = 0.6, ax = ax)
zonesM.plot(facecolor = 'violet', edgecolor = 'none', linewidth = 0.6, ax=ax)
gdf.plot(alpha = 0.5, color = 'green', markersize = 20, ax = ax)
boros.to_crs("EPSG:4326").plot(facecolor = 'none', edgecolor = 'black', linewidth = 1.0, ax = ax)
ax.axis('off');
```



Data Transformation

```
# Concatenate manufacturing zones and C8 zones into a single GeoDataFrame
zonesMandC8 = pd.concat([zonesM, zonesC8]).dissolve()
zonesMandC8.plot();
```



Creating a Square Grid

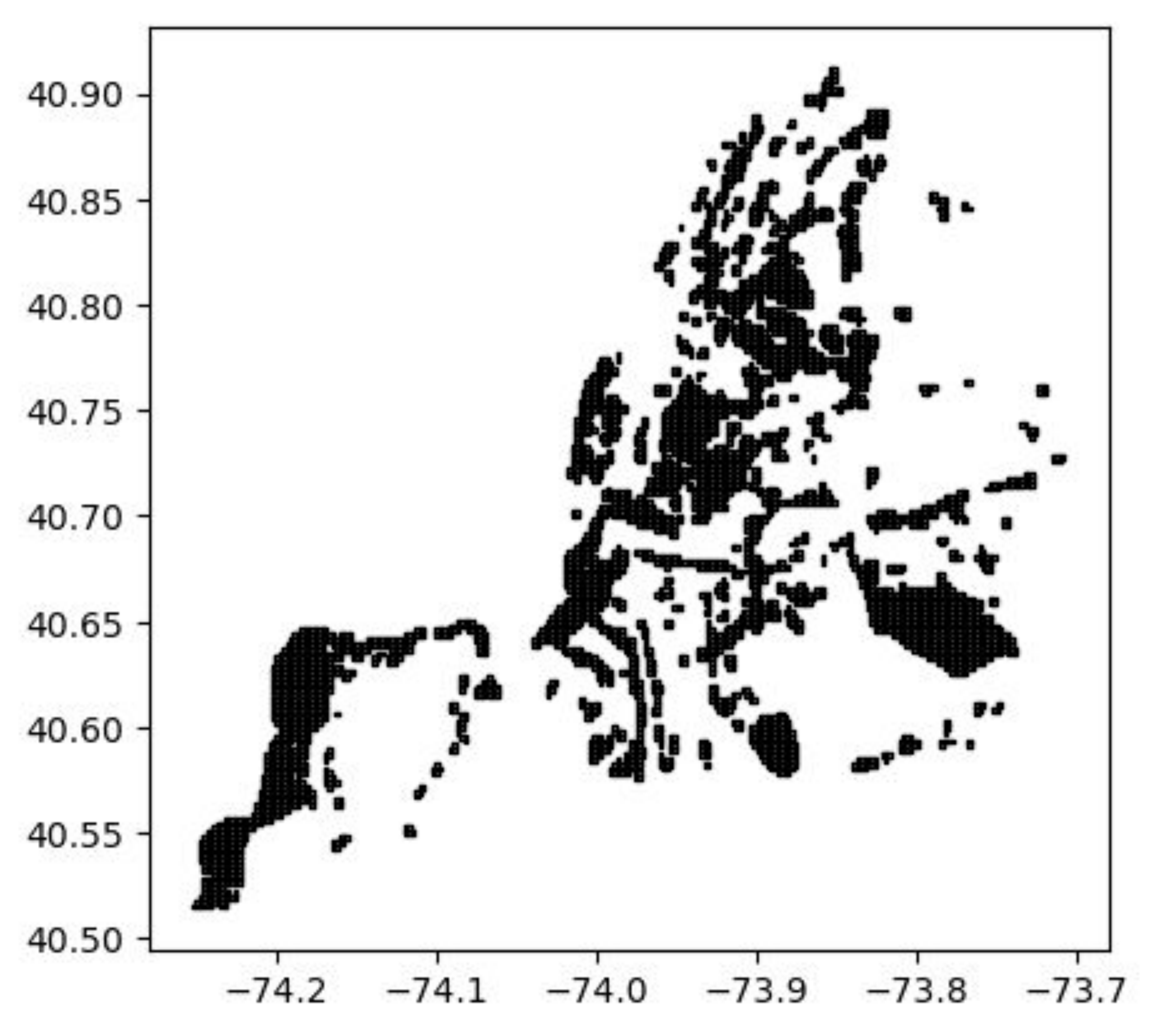
```python
#Create a function that creates a grid of squares
def make_grid(gdf, n_cells):
    gdf = gdf.copy()
    xmin, ymin, xmax, ymax = gdf.total_bounds
    cell_size = (xmax - xmin) / n_cells

    #create the cells in a loop
    grid_cells = []
    for x0 in np.arange(xmin, xmax + cell_size, cell_size):
        for y0 in np.arange(ymin, ymax + cell_size, cell_size):
            x1 = x0 - cell_size
            y1 = y0 + cell_size
            grid_cells.append(shapely.geometry.box(x0, y0, x1, y1))
    grid = gpd.GeoDataFrame(grid_cells, columns = ['geometry'], crs = gdf.crs)
    return grid
```

```python
#Generate a grid of squares
grid = make_grid(zonesMandC8, 250)
grid.plot(facecolor='white', edgecolor='black', linewidth=0.1);
```

```python
#Keep only the grid cells which intersect with a manufacturing zone or C8 zone
grid_trimmed = gpd.sjoin(grid, zonesMandC8, how='inner', predicate='intersects')

grid_trimmed.plot(facecolor='white', edgecolor='black');
```



Hypothesis

H_0: Each grid cell has equal (uniform) probability.

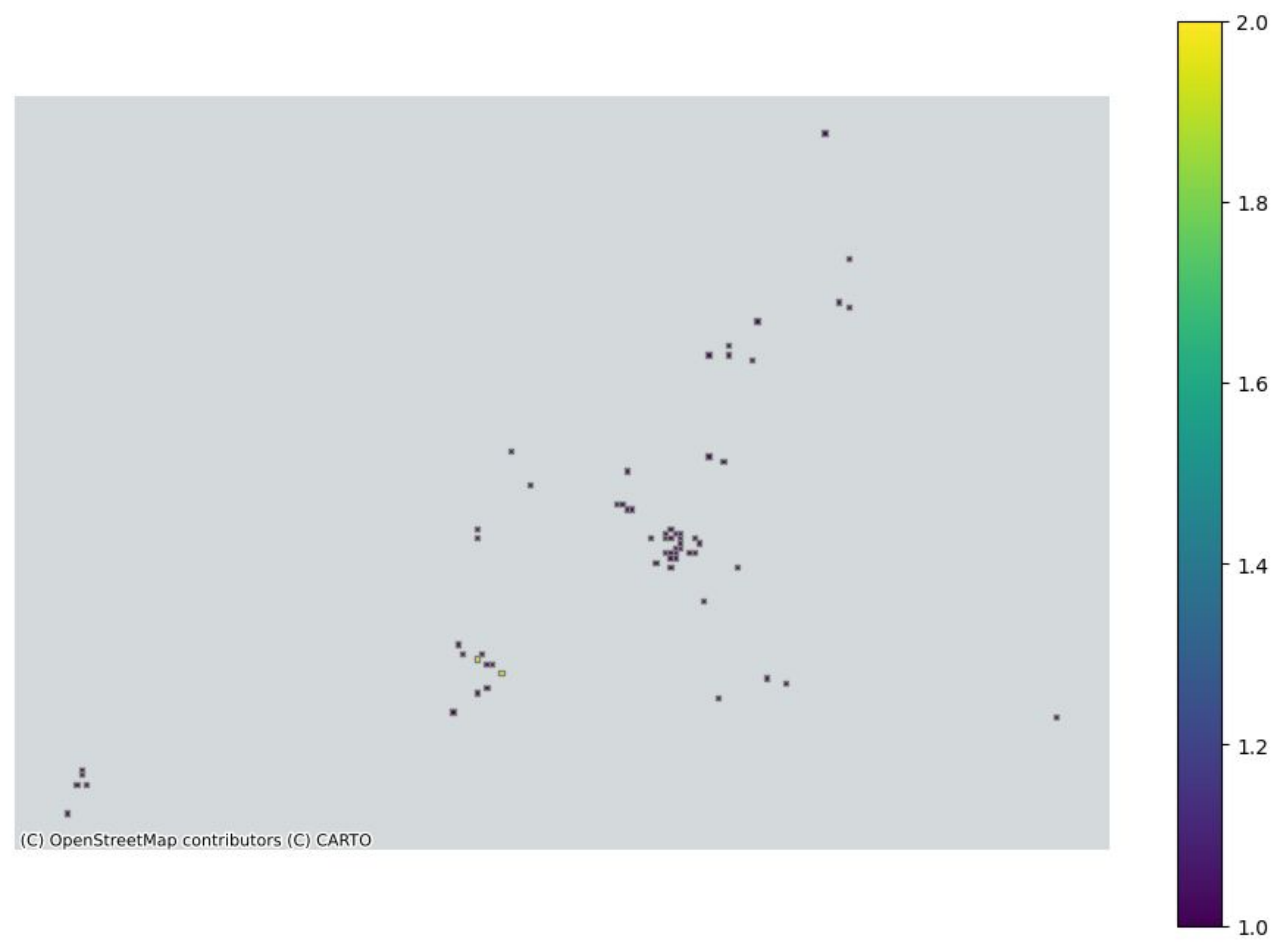
H_a: Each grid cell does not have equal probability.

Aggregate over Square Grid

```
#First we need to remove the index_right column or else the merge will not happen.
grid_trimmed=grid_trimmed.rename(columns ={'index_right':'other_name'})

#Count the number of points per grid cell
merged_sq = gpd.sjoin(gdf, grid_trimmed, how = 'left', predicate = 'within')
merged_sq['n_facilities'] = 1
dissolved_sq = merged_sq.dissolve(by = 'index_right', aggfunc = 'count')
grid_trimmed.loc[dissolved_sq.index, 'n_facilities'] = dissolved_sq.n_facilities.values

#Plot the result
ax = grid_trimmed.plot(column = 'n_facilities', figsize = (12,8),
                       cmap = 'viridis', edgecolor = 'grey', legend = True)
ax.axis('off')
cx.add_basemap(ax, source = cx.providers.CartoDB.Positron)
plt.show()
```

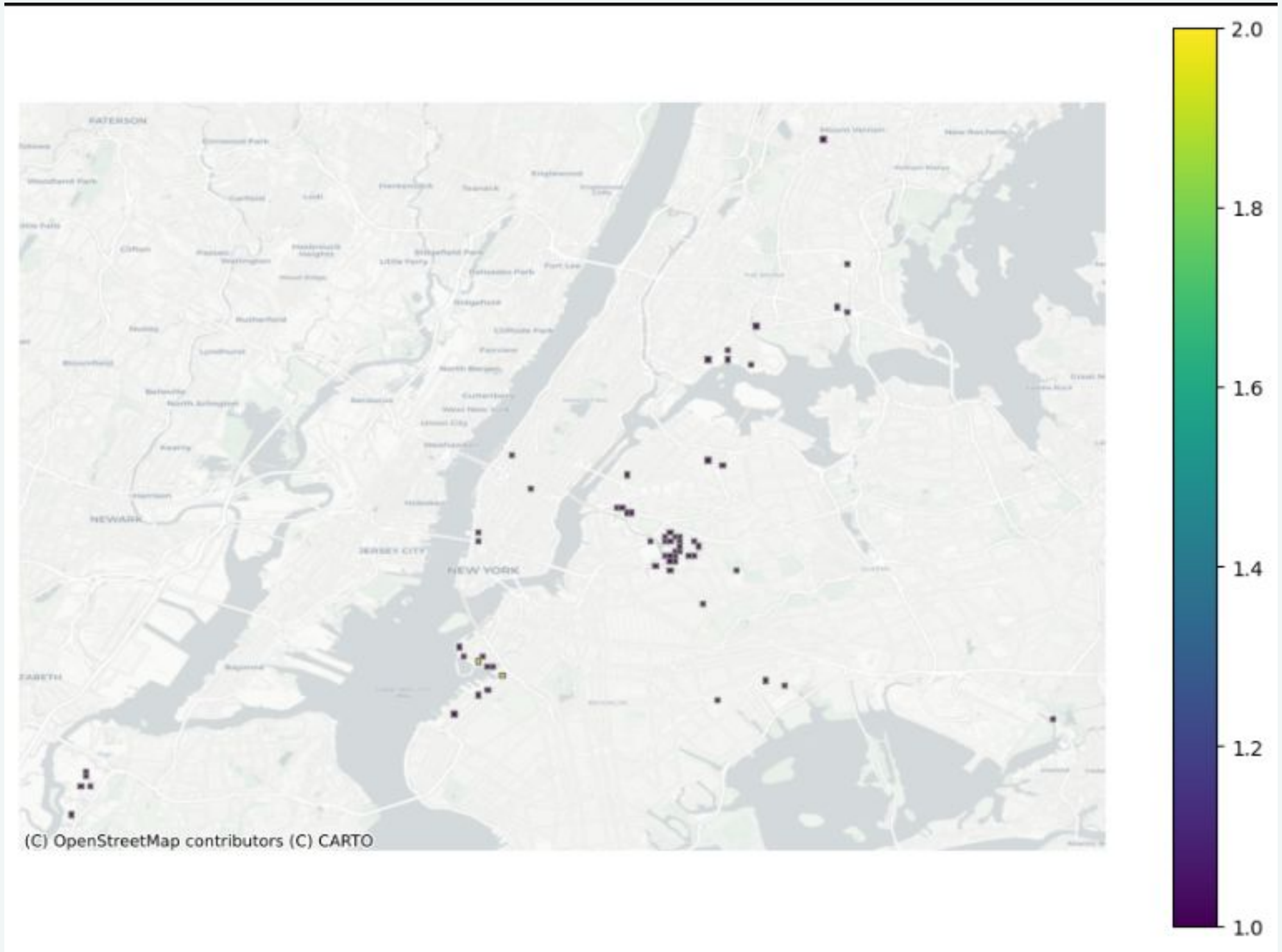


Rasterize and Find p-value for Chi-Square Test

```
#This function aggregates and summarizes point data over a grid
def rasterize(gdf, grid, aggfunc="count", column=None, plot=True):
    merged = gpd.sjoin(gdf, grid, how='left', predicate='within').copy()
    if aggfunc == "count":
        column = 'count'
        output_col = column
        merged[column] = 1
    else:
        output_col = column + "_" + aggfunc
    dissolved = merged.dissolve(by="index_right", aggfunc=aggfunc)[[column]]
    dissolved.columns = [output_col]
    grid.loc[dissolved.index, output_col] = dissolved[output_col].values
    if plot:
        ax = grid.plot(column=output_col, figsize=(12, 8), edgecolor="grey", legend=True)
        ax.axis('off')
        cx.add_basemap(ax,source=cx.providers.CartoDB.Positron,crs=gdf.crs)
        plt.show()
    return grid
```

```
stats.chisquare(r['count'].fillna(0)).pvalue

0.0055655758446668865
```



P-value < 0.05: reject the null hypothesis that the observed frequencies matches expectations.

Rasterize and Find p-value for Chi-Square Test

```
stats.chisquare(r_hex['count'].fillna(0)).pvalue
```

```
5.291700723380377e-13
```

P-value < 0.05: reject the null hypothesis that the observed frequencies matches expectations.

Hypothesis

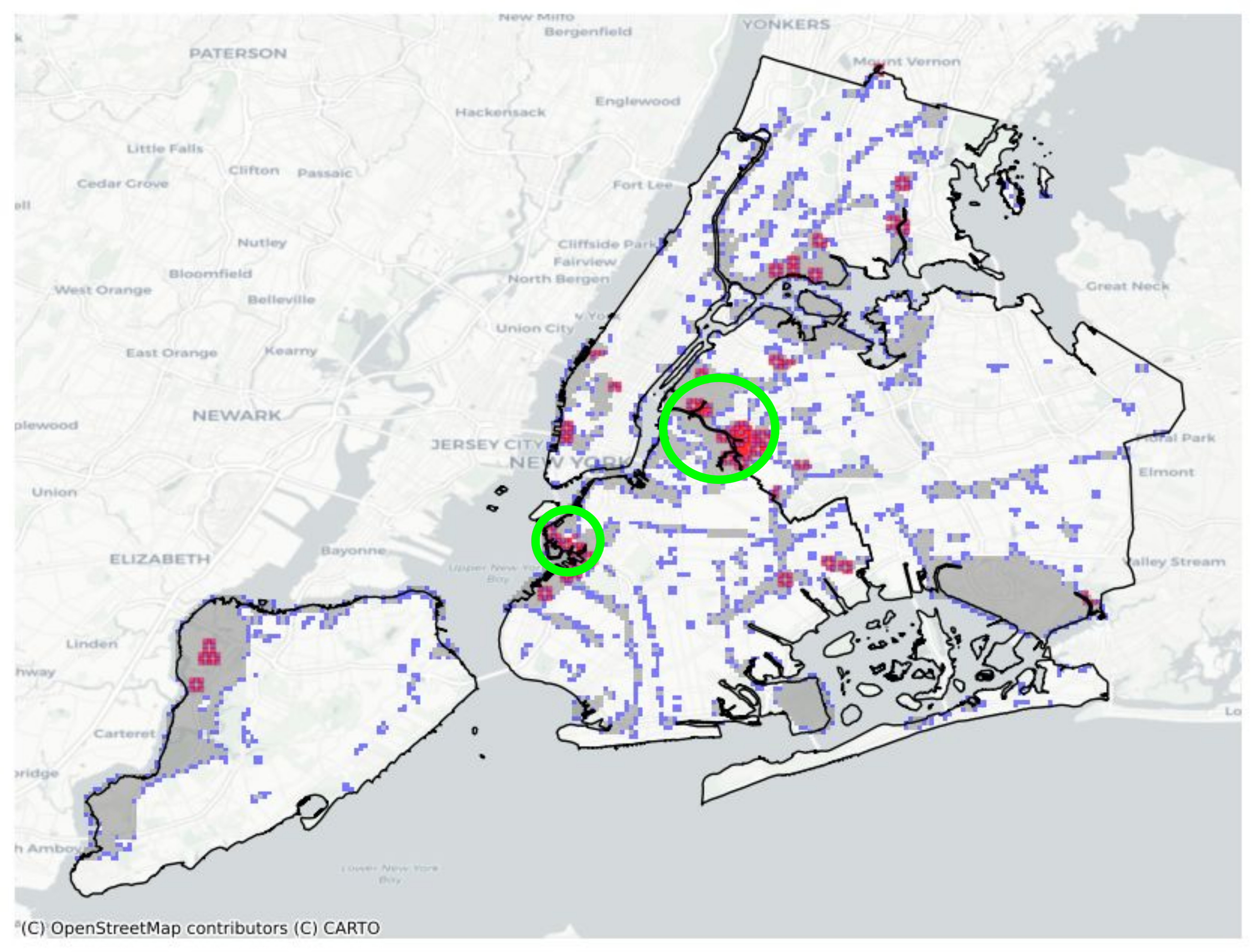
H_0: There is no spatial autocorrelation.

H_a: There is spatial autocorrelation.

Moran I

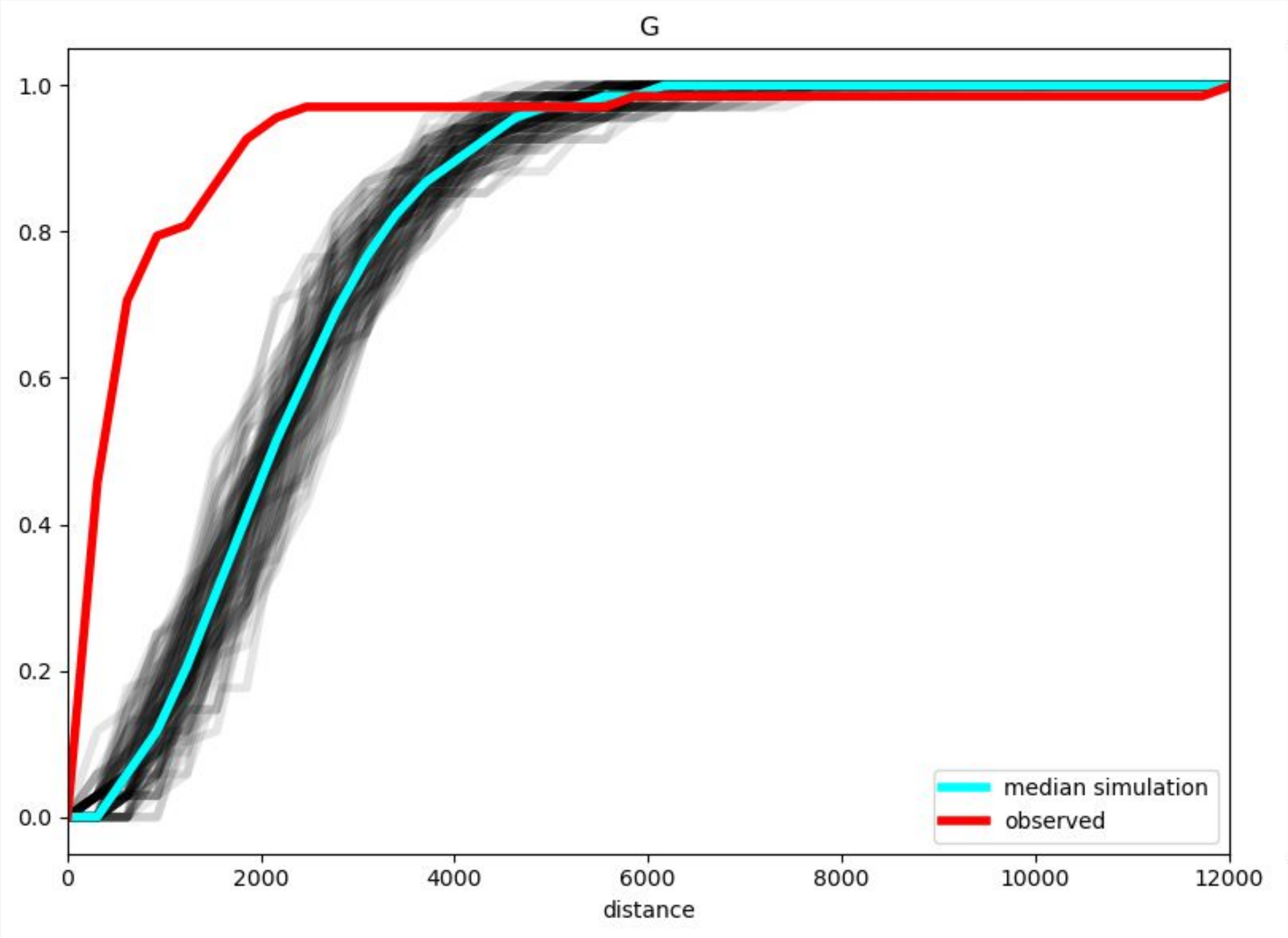
Moran I's statistic = 0.11195
p-value = 0.0001

P-value < 0.05: reject the null hypothesis that there is no spatial autocorrelation and conclude that spatial correlation is present.



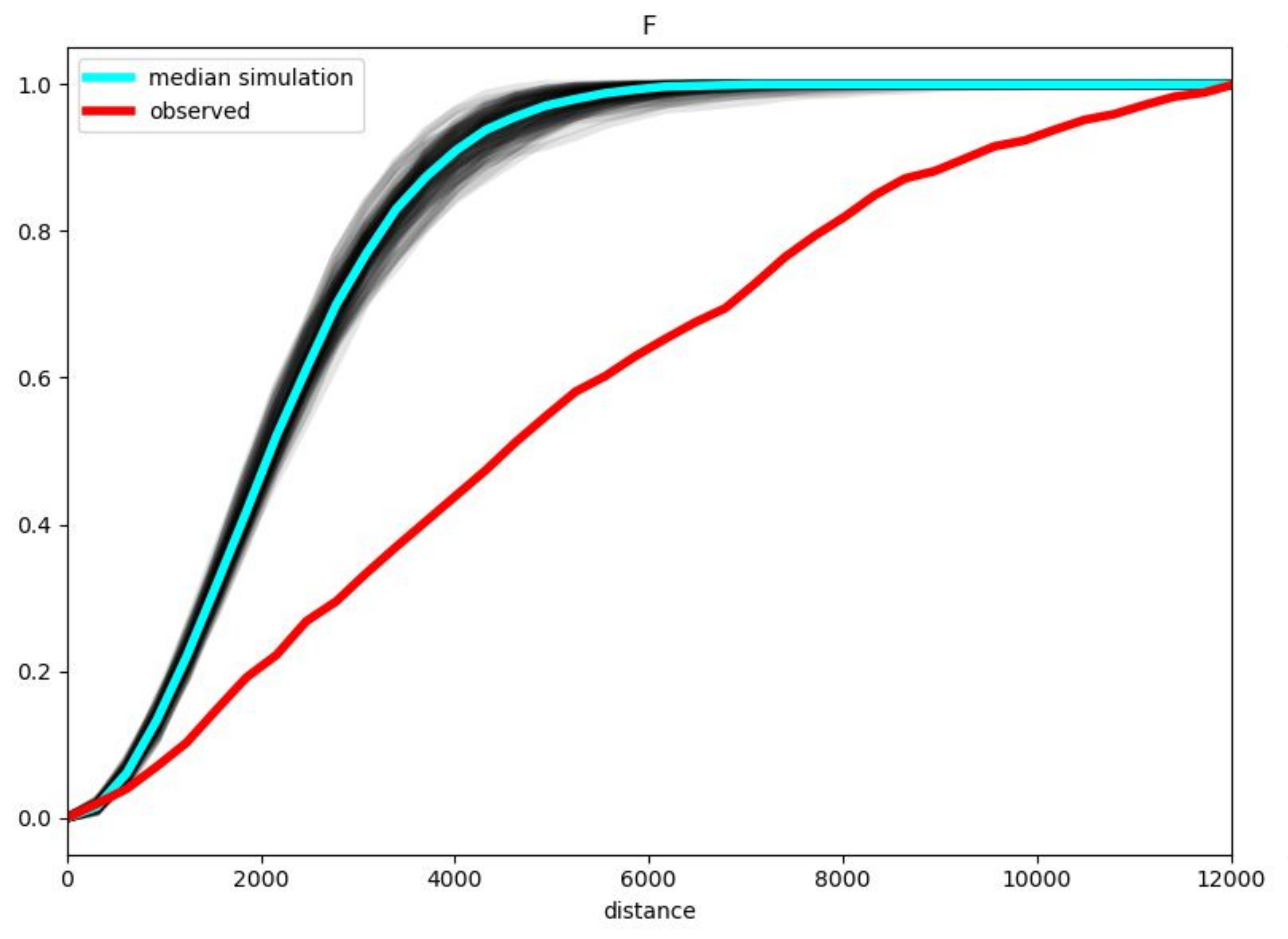
Ripley's G Test

- In this plot, we see that the **red** observed function rises much faster than simulated completely spatially random patterns
- This means that points in the observed pattern are ***closer to their nearest neighbors*** than would be expected from a completely spatially random pattern
- This pattern is **clustered.**



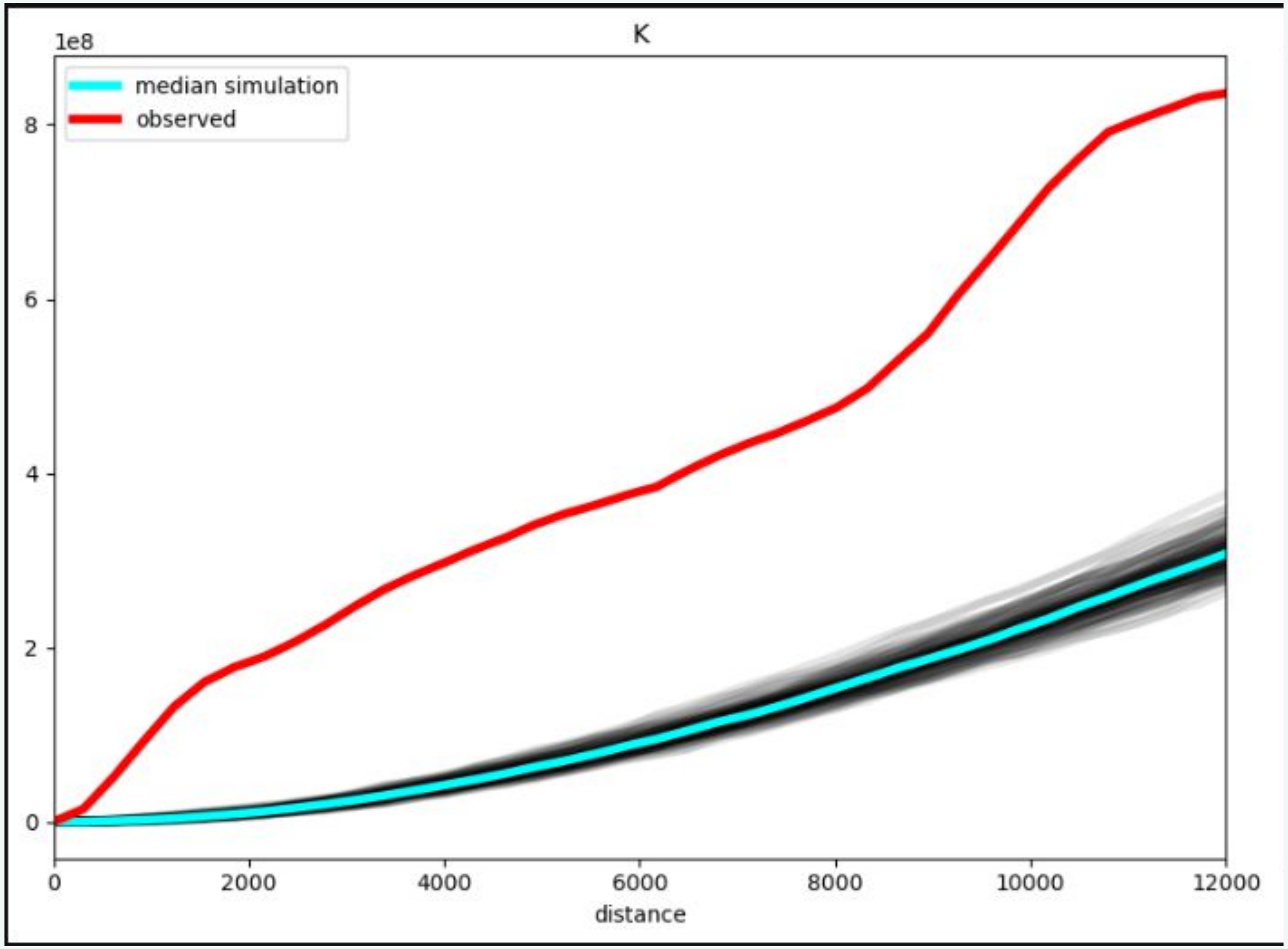
Ripley's F Test

- In this plot, we see that the **red** observed function increases more slowly than simulated completely spatially random patterns
- This means that there are more empty gaps in the pattern than would be expected from a completely spatially random pattern
- This pattern is **clustered.**



Ripley's G Test

- In this plot, we see that the **red** observed function increases more quickly than simulated completely spatially random patterns
- This means that there are more events within each distance threshold than would be expected under CSR
- This pattern is **clustered.**



Thank you.

Questions?