02-COVID-19Acc-Original

February 25, 2023

0.1 Title: RP- Spatial Accessibility of COVID-19 Healthcare Resources in Illinois

0.1.1 Reproduction

Reproduction of: Rapidly measuring spatial accessibility of COVID-19 healthcare resources: a case study of Illinois, USA

Original study *by* Kang, J. Y., A. Michels, F. Lyu, Shaohua Wang, N. Agbodo, V. L. Freeman, and Shaowen Wang. 2020. Rapidly measuring spatial accessibility of COVID-19 healthcare resources: a case study of Illinois, USA. International Journal of Health Geographics 19 (1):1–17. DOI:10.1186/s12942-020-00229-x.

Reproduction Authors: Joe Holler, Kufre Udoh, Derrick Burt, Drew An-Pham, & Spring '21 Middlebury Geog 0323.

Reproduction Materials Available at: github.com/HEGSRR/RPr-Kang-2020

Created: `8 Jun 2021` Revised: `24 Feb 2023`

0.1.2 Data

To perform the ESFCA method, three types of data are required, as follows: (1) road network, (2) population, and (3) hospital information. The road network can be obtained from the OpenStreetMap Python Library, called OSMNX. The population data is available on the American Community Survey. Lastly, hosptial information is also publically available on the Homelanad Infrastructure Foundation-Level Data.

0.1.3 Code

Import necessary libraries to run this model. See `requirements.txt` for the library versions used for this analysis.

[1]:
```
# Import modules
import numpy as np
import pandas as pd
import geopandas as gpd
import networkx as nx
import osmnx as ox
import re
from shapely.geometry import Point, LineString, Polygon
import matplotlib.pyplot as plt
```

```python
from tqdm import tqdm
import multiprocessing as mp
import folium
import itertools
import os
import time
import warnings
from scipy import stats
from IPython.display import display, clear_output

warnings.filterwarnings("ignore")
print('\n'.join(f'{m.__name__}=={m.__version__}' for m in globals().values() if␣
 ↪getattr(m, '__version__', None)))
```

```
numpy==1.22.0
pandas==1.3.5
geopandas==0.10.2
networkx==2.6.3
osmnx==1.1.2
re==2.2.1
folium==0.12.1.post1
```

0.1.4 Check Directories

Because we have restructured the repository for replication, we need to check our working directory and make necessary adjustments.

[2]:
```python
# Check working directory
os.getcwd()
```

[2]: '/home/jovyan/work/RPr-Kang-2020/procedure/code'

[3]:
```python
# Use to set work directory properly
if os.path.basename(os.getcwd()) == 'code':
    # This line moves the directory up two folders
    os.chdir('../../')
os.getcwd()
```

[3]: '/home/jovyan/work/RPr-Kang-2020'

0.2 Load and Visualize Data

0.2.1 Population and COVID-19 Cases Data by County

[4]:
```python
# Load data for at risk population
atrisk_data = gpd.read_file('./data/raw/public/PopData/Chicago_Tract.shp')
atrisk_data.head()
```

[4]:
```
          GEOID STATEFP COUNTYFP TRACTCE         NAMELSAD   Pop  Unnamed_ 0  \
0  17031010400      17      031  010400  Census Tract 104  5153        1538
1  17031010600      17      031  010600  Census Tract 106  6271         438
2  17031030200      17      031  030200  Census Tract 302  5444        2075
3  17031030300      17      031  030300  Census Tract 303  3464         516
4  17031030400      17      031  030400  Census Tract 304  2582        1520

                                     NAME  OverFifty  TotalPop  \
0  Census Tract 104, Cook County, Illinois       1103      5153
1  Census Tract 106, Cook County, Illinois       1469      6271
2  Census Tract 302, Cook County, Illinois       2018      5444
3  Census Tract 303, Cook County, Illinois       1097      3464
4  Census Tract 304, Cook County, Illinois        860      2582

                                            geometry
0  POLYGON ((-87.66125 42.01288, -87.66125 42.012…
1  POLYGON ((-87.67059 42.00537, -87.67046 42.005…
2  POLYGON ((-87.67062 41.99808, -87.67045 41.998…
3  POLYGON ((-87.67501 41.99799, -87.67473 41.998…
4  POLYGON ((-87.67471 41.99076, -87.67440 41.990…
```

[5]:
```
# Load data for covid cases
covid_data = gpd.read_file('./data/raw/public/PopData/Chicago_ZIPCODE.shp')
covid_data['cases'] = covid_data['cases']
covid_data.head()
```

[5]:
```
  ZCTA5CE10       County State            Join  ZONE       ZONENAME  FIPS  \
0     60660  Cook County    IL  Cook County IL  IL_E  Illinois East  1201
1     60640  Cook County    IL  Cook County IL  IL_E  Illinois East  1201
2     60614  Cook County    IL  Cook County IL  IL_E  Illinois East  1201
3     60712  Cook County    IL  Cook County IL  IL_E  Illinois East  1201
4     60076  Cook County    IL  Cook County IL  IL_E  Illinois East  1201

     pop  cases                                           geometry
0  43242     78  POLYGON ((-87.65049 41.99735, -87.65029 41.996…
1  69715    117  POLYGON ((-87.64645 41.97965, -87.64565 41.978…
2  71308    134  MULTIPOLYGON (((-87.67703 41.91845, -87.67705 …
3  12539     42  MULTIPOLYGON (((-87.76181 42.00465, -87.76156 …
4  31867    114  MULTIPOLYGON (((-87.74782 42.01540, -87.74526 …
```

0.2.2 Hospital Data

Note that 999 is treated as a “NULL”/“NA” so these hospitals are filtered out. This data contains the number of ICU beds and ventilators at each hospital.

[6]:

```
# Load data for hospitals
hospitals = gpd.read_file('./data/raw/public/HospitalData/Chicago_Hospital_Info.
↪shp')
hospitals.head()
```

[6]:

```
   FID                                        Hospital      City  ZIP_Code  \
0    2                    Methodist Hospital of Chicago   Chicago     60640
1    4                   Advocate Christ Medical Center  Oak Lawn     60453
2   13                                Evanston Hospital  Evanston     60201
3   24  AMITA Health Adventist Medical Center Hinsdale  Hinsdale     60521
4   25                              Holy Cross Hospital   Chicago     60629

           X          Y  Total_Bed  Adult ICU  Total Vent  \
0 -87.671079  41.972800        145         36          12
1 -87.732483  41.720281        785        196          64
2 -87.683288  42.065393        354         89          29
3 -87.920116  41.805613        261         65          21
4 -87.690841  41.770001        264         66          21

                         geometry
0  MULTIPOINT (-87.67108 41.97280)
1  MULTIPOINT (-87.73248 41.72028)
2  MULTIPOINT (-87.68329 42.06539)
3  MULTIPOINT (-87.92012 41.80561)
4  MULTIPOINT (-87.69084 41.77000)
```

0.2.3 Generate and Plot Map of Hospitals

[7]:

```
# Plot hospitals
m = folium.Map(location=[41.85, -87.65], tiles='cartodbpositron', zoom_start=10)

for i in range(0, len(hospitals)):
    folium.CircleMarker(
      location=[hospitals.iloc[i]['Y'], hospitals.iloc[i]['X']],
      popup="{}{}\n{}{}\n{}{}".format('Hospital Name: ',hospitals.
↪iloc[i]['Hospital'],
                                      'ICU Beds: ',hospitals.iloc[i]['Adult␣
↪ICU'],
                                      'Ventilators: ', hospitals.iloc[i]['Total␣
↪Vent']),
      radius=5,
      color='grey',
      fill=True,
      fill_opacity=0.6,
      legend_name = 'Hospitals'
    ).add_to(m)
```

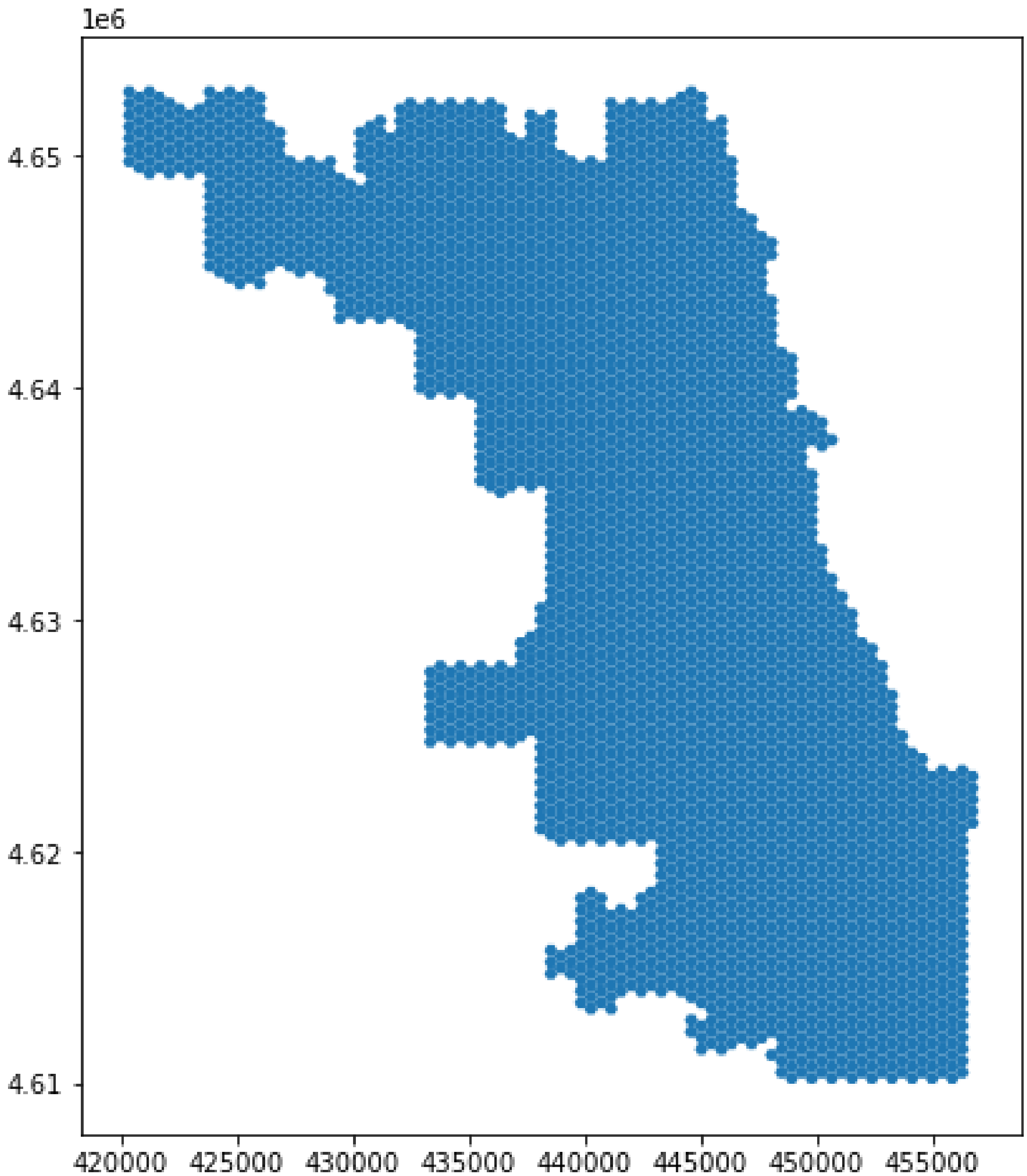
```
legend_html =   '''<div style="position: fixed; width: 20%; heigh: auto;
                        bottom: 10px; left: 10px;
                        solid grey; z-index:9999; font-size:14px;
                        ">  Legend<br>'''

m
```

[7]: <folium.folium.Map at 0x7f0d043bcc40>

0.2.4 Load and Plot Hexagon Grids (500-meter resolution)

[8]:
```
# Load grid file and plot
grid_file = gpd.read_file('./data/raw/public/GridFile/Chicago_Grid.shp')
grid_file.plot(figsize=(8,8))
```

[8]: <AxesSubplot:>



0.2.5 Load the Original Street Network

Note: Skip the next cell unless you want to simulate errors caused by using a street network constained to Chicago, Illinois.

[15]:
```
%%time
print("Loading Chicago road network from data/raw/public/Chicago_Network.
 ↪graphml. Please wait...", flush=True)
G = ox.load_graphml('./data/raw/public/Chicago_Network.graphml')
```

```
print("Data loaded.")
```

```
Loading Chicago road network from data/raw/public/Chicago_Network.graphml.
Please wait…
Data loaded.
CPU times: user 8.19 s, sys: 191 ms, total: 8.38 s
Wall time: 8.38 s
```

0.2.6 Load the Buffered Street Network

Note: Do not run this cell if you want to simulate errors caused by using a street network constained to Chicago, Illinois.

[16]:
```
%%time
# Read in Chicago street network (pull from OSMNX drive if it doesn't already␣
↪exist)
if not os.path.exists("./data/raw/private/Chicago_Network_Buffer.graphml"):
    print("Loading Chicago road network from OpenStreetMap. Please wait... last␣
↪runtime was 9min 20s...", flush=True)
    G = ox.graph_from_place('Chicago', network_type='drive', buffer_dist=24140.
↪2)
    print("Saving Chicago road network to raw/private/Chicago_Network_Buffer.
↪graphml. Please wait...", flush=True)
    ox.save_graphml(G, './data/raw/private/Chicago_Network_Buffer.graphml')
    print("Data saved.")
else:
    print("Loading Chicago road network from raw/private/Chicago_Network_Buffer.
↪graphml. Please wait...", flush=True)
    G = ox.load_graphml('./data/raw/private/Chicago_Network_Buffer.graphml')
    print("Data loaded.")
```

```
Loading Chicago road network from raw/private/Chicago_Network_Buffer.graphml.
Please wait…
Data loaded.
CPU times: user 36.4 s, sys: 1.36 s, total: 37.7 s
Wall time: 37.7 s
```

0.2.7 Plot the Road Network

[17]:
```
%%time
ox.plot_graph(G)
```

```
CPU times: user 49 s, sys: 303 ms, total: 49.3 s
Wall time: 49 s
```

[17]: (<Figure size 576x576 with 1 Axes>, <AxesSubplot:>)

[18]:
```
%%time
# Turn nodes and edges into geodataframes
nodes, edges = ox.graph_to_gdfs(G, nodes=True, edges=True)

# Get unique counts of road segments for each speed limit
print(edges['maxspeed'].value_counts())
```

```python
print(len(edges))
```

```
25 mph                         4793
30 mph                         3555
35 mph                         3364
40 mph                         2093
45 mph                         1418
20 mph                         1155
55 mph                          614
60 mph                          279
50 mph                          191
40                               79
15 mph                           76
70 mph                           71
65 mph                           54
10 mph                           38
[40 mph, 45 mph]                 27
[30 mph, 35 mph]                 26
45,30                            24
[40 mph, 35 mph]                 22
70                               21
25                               20
[55 mph, 45 mph]                 16
25, east                         14
[45 mph, 35 mph]                 13
[30 mph, 25 mph]                 10
[45 mph, 50 mph]                  8
50                                8
[40 mph, 30 mph]                  7
[35 mph, 25 mph]                  6
[55 mph, 60 mph]                  5
20                                4
[70 mph, 60 mph]                  3
[65 mph, 60 mph]                  3
[40 mph, 45 mph, 35 mph]          3
[70 mph, 65 mph]                  2
[70 mph, 45 mph, 5 mph]           2
[40, 45 mph]                      2
[35 mph, 50 mph]                  2
35                                2
[55 mph, 65 mph]                  2
[40 mph, 50 mph]                  2
[15 mph, 25 mph]                  2
[40 mph, 35 mph, 25 mph]          2
[15 mph, 40 mph, 30 mph]          2
[20 mph, 25 mph]                  2
[30 mph, 25, east]                2
```

```
[65 mph, 55 mph]               2
[20 mph, 35 mph]               2
[55 mph, 55]                   2
55                             2
[15 mph, 30 mph]               2
[45 mph, 30 mph]               2
[15 mph, 45 mph]               2
[55 mph, 45, east, 50 mph]     2
[20 mph, 30 mph]               1
[5 mph, 45 mph, 35 mph]        1
[55 mph, 35 mph]               1
[5 mph, 35 mph]                1
[55 mph, 50 mph]               1
Name: maxspeed, dtype: int64
384240
CPU times: user 30.7 s, sys: 52.5 ms, total: 30.8 s
Wall time: 30.8 s
```

0.3 "Helper" Functions

The functions below are needed for our analysis later, let's take a look!

0.3.1 network_setting

Cleans the OSMNX network to work better with drive-time analysis.

First, we remove all nodes with 0 outdegree because any hospital assigned to such a node would be unreachable from everywhere. Next, we remove small (under 10 node) *strongly connected components* to reduce erroneously small ego-centric networks. Lastly, we ensure that the max speed is set and in the correct units before calculating time.

Args:

- network: OSMNX network for the spatial extent of interest

Returns:

- OSMNX network: cleaned OSMNX network for the spatial extent

[19]:
```
def network_setting(network):
    _nodes_removed = len([n for (n, deg) in network.out_degree() if deg ==0])
    network.remove_nodes_from([n for (n, deg) in network.out_degree() if deg␣
 ↪==0])
    for component in list(nx.strongly_connected_components(network)):
        if len(component)<10:
            for node in component:
                _nodes_removed+=1
                network.remove_node(node)
    for u, v, k, data in tqdm(G.edges(data=True, keys=True),position=0):
        if 'maxspeed' in data.keys():
            speed_type = type(data['maxspeed'])
```

```
            if (speed_type==str):
                # Add in try/except blocks to catch maxspeed formats that don't␣
 ↪fit Kang et al's cases
                try:
                    if len(data['maxspeed'].split(','))==2:
                        data['maxspeed_fix']=float(data['maxspeed'].
 ↪split(',')[0])
                    elif data['maxspeed']=='signals':
                        data['maxspeed_fix']=35.0 # Drive speed setting as 35␣
 ↪miles
                    else:
                        data['maxspeed_fix']=float(data['maxspeed'].split()[0])
                except:
                    data['maxspeed_fix']=35.0 # Miles
            else:
                try:
                    data['maxspeed_fix']=float(data['maxspeed'][0].split()[0])
                except:
                    data['maxspeed_fix']=35.0 # Miles
        else:
            data['maxspeed_fix']= 35.0 # Miles
        data['maxspeed_meters'] = data['maxspeed_fix']*26.8223 # Convert mile␣
 ↪to meter
        data['time'] = float(data['length'])/ data['maxspeed_meters']
    print("Removed {} nodes ({:2.4f}%) from the OSMNX network".
 ↪format(_nodes_removed, _nodes_removed/float(network.number_of_nodes())))
    print("Number of nodes: {}".format(network.number_of_nodes()))
    print("Number of edges: {}".format(network.number_of_edges()))
    return(network)
```

Pre Process Street network

[20]:
```
%%time
# G, hospitals, grid_file, pop_data = file_import (population_dropdown.value,␣
 ↪place_dropdown.value)
G = network_setting(G)
# Create point geometries for each node in the graph, to make constructing␣
 ↪catchment area polygons easier
for node, data in G.nodes(data=True):
    data['geometry']=Point(data['x'], data['y'])
# Modify code to react to processor dropdown (got rid of file_import function)
```

```
100%|      | 383911/383911 [00:00<00:00, 400496.20it/s]

Removed 274 nodes (0.0019%) from the OSMNX network
Number of nodes: 142044
Number of edges: 383911
CPU times: user 5.55 s, sys: 282 ms, total: 5.84 s
```

```
Wall time: 5.81 s
```

[21]:
```
%%time
## Get unique counts for each road network
# Turn nodes and edges in geodataframes
nodes, edges = ox.graph_to_gdfs(G, nodes=True, edges=True)

# Count
print(edges['maxspeed_fix'].value_counts())
print(len(edges))
```

```
35.0    369255
25.0      4827
30.0      3593
40.0      2237
45.0      1465
20.0      1161
55.0       637
60.0       277
50.0       199
70.0        89
15.0        84
65.0        47
10.0        38
5.0          2
Name: maxspeed_fix, dtype: int64
383911
CPU times: user 31 s, sys: 140 ms, total: 31.2 s
Wall time: 31.1 s
```

0.3.2 hospital_setting

Finds the nearest OSMNX node for each hospital.

Args:

- hospital: GeoDataFrame of hospitals
- G: OSMNX network

Returns:

- GeoDataFrame of hospitals with info on nearest OSMNX node

[22]:
```
def hospital_setting(hospitals, G):
    # Create an empty column
    hospitals['nearest_osm']=None
    # Append the neaerest osm column with each hospitals neaerest osm node
    for i in tqdm(hospitals.index, desc="Find the nearest osm from hospitals",␣
 ↪position=0):
```

```
        hospitals['nearest_osm'][i] = ox.get_nearest_node(G,␣
↪[hospitals['Y'][i], hospitals['X'][i]], method='euclidean') # find the␣
↪nearest node from hospital location
    print ('hospital setting is done')
    return(hospitals)
```

0.3.3 pop_centroid

Converts geodata to centroids

Args:

- pop_data: a GeodataFrame
- pop_type: a string, either "pop" for general population or "covid" for COVID-19 case data

Returns:

- GeoDataFrame of centroids with population data

[23]:
```
# To estimate the centroids of census tract / county
def pop_centroid (pop_data, pop_type):
    pop_data = pop_data.to_crs({'init': 'epsg:4326'})
    # If pop is selected in dropdown, select at risk pop where population is␣
↪greater than 0
    if pop_type =="pop":
        pop_data=pop_data[pop_data['OverFifty']>=0]
    # If covid is selected in dropdown, select where covid cases are greater␣
↪than 0
    if pop_type =="covid":
        pop_data=pop_data[pop_data['cases']>=0]
    pop_cent = pop_data.centroid # it make the polygon to the point without any␣
↪other information
    # Convert to gdf
    pop_centroid = gpd.GeoDataFrame()
    i = 0
    for point in tqdm(pop_cent, desc='Pop Centroid File Setting', position=0):
        if pop_type== "pop":
            pop = pop_data.iloc[i]['OverFifty']
            code = pop_data.iloc[i]['GEOID']
        if pop_type =="covid":
            pop = pop_data.iloc[i]['cases']
            code = pop_data.iloc[i].ZCTA5CE10
        pop_centroid = pop_centroid.append({'code':code,'pop': pop,'geometry':␣
↪point}, ignore_index=True)
        i = i+1
    return(pop_centroid)
```

0.3.4 calculate_catchment_area

Calculates a catchment area of things within some distance of a point using a given metric.

Function first creates an ego-centric subgraph on the NetworkX road network starting with the nearest OSM node for the hospital and going out to a given distance as measured by the distance unit. We then calculate the convex hull around the nodes in the ego-centric subgraph and make it a GeoPandas object.

Args:

- G: OSMNX network
- nearest_osm: OSMNX road network node that is closest to the place of interest (hospital)
- distance: the max distance to include in the catchment area
- distance_unit: how we measure distance (used by ego_graph), we always use time

Returns:

- GeoDataFrame the catchment area.

[24]:
```
def calculate_catchment_area(G, nearest_osm, distance, distance_unit = "time"):
    # Consutrct an ego graph based on distance unit for an input node
    road_network = nx.ego_graph(G, nearest_osm, distance,␣
 ↪distance=distance_unit)
    # Create point geometries for all nodes in ego graph
    nodes = [Point((data['x'], data['y'])) for node, data in road_network.
 ↪nodes(data=True)]
    # Create a single part geometry of all nodes
    polygon = gpd.GeoSeries(nodes).unary_union.convex_hull ## to create convex␣
 ↪hull
    polygon = gpd.GeoDataFrame(gpd.GeoSeries(polygon)) ## change polygon to␣
 ↪geopandas
    polygon = polygon.rename(columns={0:'geometry'}).set_geometry('geometry')
    return polygon.copy(deep=True)
```

0.3.5 hospital_measure_acc

Measures the effect of a single hospital on the surrounding area. (Uses `calculate_catchment_area` or `djikstra_cca`)

Args:

- _thread_id: int used to keep track of which thread this is
- hospital: Geopandas dataframe with information on a hospital
- pop_data: Geopandas dataframe with population data
- distances: Distances in time to calculate accessibility for
- weights: how to weight the different travel distances

Returns:

- Tuple containing:
  - Int (_thread_id)
  - GeoDataFrame of catchment areas with key stats

```
[25]: def hospital_measure_acc (_thread_id, hospital, pop_data, distances, weights):
    # weights = 1, 0.68, 0.22
    # distances = 10 20 30
    # Apply catchment calculation for each distance (10, 20, and 30 min)
    polygons = []
    for distance in distances:
        # Append djikstra catchment calculation (uncomment to use)
        polygons.append(calculate_catchment_area(G,␣
↪hospital['nearest_osm'],distance))
    # Clip the overlapping distance ploygons (create two donuts + hole)
    for i in reversed(range(1, len(distances))):
        polygons[i] = gpd.overlay(polygons[i], polygons[i-1], how="difference")

    # Calculate accessibility measurements
    num_pops = []
    for j in pop_data.index:
        point = pop_data['geometry'][j]
        # Multiply polygons by weights
        for k in range(len(polygons)):
            if len(polygons[k]) > 0: # To exclude the weirdo (convex hull is␣
↪not polygon)
                if (point.within(polygons[k].iloc[0]["geometry"])):
                    num_pops.append(pop_data['pop'][j]*weights[k])
    total_pop = sum(num_pops)
    for i in range(len(distances)):
        polygons[i]['time']=distances[i]
        polygons[i]['total_pop']=total_pop
        polygons[i]['icu_beds'] = float(hospital['Adult ICU'])/
↪polygons[i]['total_pop'] # proportion of # of beds over pops in 10 mins
        polygons[i]['vents'] = float(hospital['Total Vent'])/
↪polygons[i]['total_pop'] # proportion of # of beds over pops in 10 mins
        polygons[i].crs = { 'init' : 'epsg:4326'}
        polygons[i] = polygons[i].to_crs({'init':'epsg:32616'})
    print('\rCatchment for hospital {:4.0f} complete'.format(_thread_id), end="␣
↪", flush=True)
    return(_thread_id, [ polygon.copy(deep=True) for polygon in polygons ])
```

0.3.6 measure_acc_par

Parallel implementation of accessibility measurement.

Args:

- hospitals: Geodataframe of hospitals
- pop_data: Geodataframe containing population data
- network: OSMNX street network
- distances: list of distances to calculate catchments for
- weights: list of floats to apply to different catchments

- num_proc: number of processors to use.

Returns:

- Geodataframe of catchments with accessibility statistics calculated

[26]:
```
def hospital_acc_unpacker(args):
    return hospital_measure_acc(*args)

# Parallel implementation fo previous function
def measure_acc_par (hospitals, pop_data, network, distances, weights, num_proc␣
 ↪= 4):
    catchments = []
    for distance in distances:
        catchments.append(gpd.GeoDataFrame())
    pool = mp.Pool(processes = num_proc)
    hospital_list = [ hospitals.iloc[i] for i in range(len(hospitals)) ]
    results = pool.map(hospital_acc_unpacker, zip(range(len(hospital_list)),␣
 ↪hospital_list, itertools.repeat(pop_data), itertools.repeat(distances),␣
 ↪itertools.repeat(weights)))
    pool.close()
    results.sort()
    results = [ r[1] for r in results ]
    for i in range(len(results)):
        for j in range(len(distances)):
            catchments[j] = catchments[j].append(results[i][j], sort=False)
    return catchments
```

0.3.7 overlap_calc

Calculates and aggregates accessibility statistics for one catchment on our grid file.

Args:

- _id: thread ID
- poly: GeoDataFrame representing a catchment area
- grid_file: a GeoDataFrame representing our grids
- weight: the weight to applied for a given catchment
- service_type: the service we are calculating for: ICU beds or ventilators

Returns:

- Tuple containing:
  - thread ID
  - Counter object (dictionary for numbers) with aggregated stats by grid ID number

[27]:
```
from collections import Counter
def overlap_calc(_id, poly, grid_file, weight, service_type):
    value_dict = Counter()
    if type(poly.iloc[0][service_type])!=type(None):
```

```
        value = float(poly[service_type])*weight
        # Find polygons that overlap hex grids
        intersect = gpd.overlay(grid_file, poly, how='intersection')
        # Get the intersection's area
        intersect['overlapped']= intersect.area
        # Divide overlapping area by total area to get percent
        intersect['percent'] = intersect['overlapped']/intersect['area']
        # Only choose intersecting catchments that make up greater than 50% of␣
↪hexagon
        intersect=intersect[intersect['percent']>=0.5]
        # Pull id
        intersect_region = intersect['id']
        for intersect_id in intersect_region:
            try:
                value_dict[intersect_id] +=value
            except:
                value_dict[intersect_id] = value
    return(_id, value_dict)

def overlap_calc_unpacker(args):
    return overlap_calc(*args)
```

0.3.8 overlapping_function

Calculates how all catchment areas overlap with and affect the accessibility of each grid in our grid file.

Args:

- grid_file: GeoDataFrame of our grid
- catchments: GeoDataFrame of our catchments
- service_type: the kind of care being provided (ICU beds vs. ventilators)
- weights: the weight to apply to each service type
- num_proc: the number of processors

Returns:

- Geodataframe - grid_file with calculated stats

[28]:
```
def overlapping_function (grid_file, catchments, service_type, weights,␣
↪num_proc = 4):
    grid_file[service_type]=0
    pool = mp.Pool(processes = num_proc)
    acc_list = []
    for i in range(len(catchments)):
        acc_list.extend([ catchments[i][j:j+1] for j in␣
↪range(len(catchments[i])) ])
    acc_weights = []
    for i in range(len(catchments)):
```

```
        acc_weights.extend( [weights[i]]*len(catchments[i]) )
    results = pool.map(overlap_calc_unpacker, zip(range(len(acc_list)),␣
↪acc_list, itertools.repeat(grid_file), acc_weights, itertools.
↪repeat(service_type)))
    pool.close()
    results.sort()
    results = [ r[1] for r in results ]
    service_values = results[0]
    for result in results[1:]:
        service_values+=result
    for intersect_id, value in service_values.items():
        grid_file.loc[grid_file['id']==intersect_id, service_type] += value
    return(grid_file)
```

0.3.9 normalization

Normalizes our result (Geodataframe) for a given resource (res).

[29]:
```
def normalization (result, res):
    result[res]=(result[res]-min(result[res]))/
↪(max(result[res])-min(result[res]))
    return result
```

0.3.10 Output Map Functions

[30]:
```
def output_map(output_grid, base_map, hospitals, resource):
    ax=output_grid.plot(column=resource,
                        cmap='PuBuGn',
                        figsize=(18,12),
                        legend=True,
                        zorder=1)
    # Next two lines set bounds for our x- and y-axes because it looks like␣
↪there's a weird
    # Point at the bottom left of the map that's messing up our frame (Maja)
    ax.set_xlim([325000, 370000])
    ax.set_ylim([550000, 600000])
    hospitals.plot(ax=ax,
                   markersize=10,
                   zorder=1,
                   c='black',
                   legend=False)
```

[31]:
```
def output_map_classified(output_grid, base_map, hospitals, resource):
    ax=output_grid.plot(column=resource,
                        scheme='Equal_Interval',
                        k=5,
                        linewidth=0,
```

```
                        cmap='Blues',
                        figsize=(18,12),
                        legend=True,
                        label="Acc Measure",
                        zorder=1)
    # Next two lines set bounds for our x- and y-axes because it looks like␣
 ↪there's a weird
    # Point at the bottom left of the map that's messing up our frame (Maja)
    ax.set_xlim([325000, 370000])
    ax.set_ylim([550000, 600000])
    hospitals.plot(ax=ax,
                   markersize=10,
                   zorder=2,
                   c='black',
                   legend=False,
                   )
```

0.3.11 READ ME:

This final section of code requires running and re-running certain cells depending on your inputs in the dropdown menu below. There are step-by-step provided instructions in the text cells, but the general idea is to run the code for each population and resource option before doing the final section. So, run through the code cell below up until the "STOP HERE!" for each iteration, and then move on to the final section.

0.3.12 Run the model

Below you can customize the input of the model:

- Processor - the number of processors to use
- Population - the population to calculate the measure for
- Resource - the hospital resource of interest

[32]:
```
import ipywidgets
from IPython.display import display

processor_dropdown = ipywidgets.Dropdown( options=[("1", 1), ("2", 2), ("3",␣
 ↪3), ("4", 4)],
    value=4, description="Processor: ")

population_dropdown = ipywidgets.Dropdown( options=[("Population at Risk",␣
 ↪"pop"), ("COVID-19 Patients", "covid") ],
    value="pop", description="Population: ")

resource_dropdown = ipywidgets.Dropdown( options=[("ICU Beds", "icu_beds"),␣
 ↪("Ventilators", "vents") ],
    value="icu_beds", description="Resource: ")
```

```
display(processor_dropdown,population_dropdown,resource_dropdown)
```

```
Dropdown(description='Processor: ', index=3, options=(('1', 1), ('2', 2), ('3',␣
↪3), ('4', 4)), value=4)

Dropdown(description='Population: ', options=(('Population at Risk', 'pop'),␣
↪('COVID-19 Patients', 'covid')), …

Dropdown(description='Resource: ', options=(('ICU Beds', 'icu_beds'),␣
↪('Ventilators', 'vents')), value='icu_be…
```

[63]:
```
%%time
G = network_setting (G)
# Modify code to select pop valuee based on dropdown menu choice
if population_dropdown.value == "pop":
    pop_data = pop_centroid(atrisk_data, population_dropdown.value)
elif population_dropdown.value == "covid":
    pop_data = pop_centroid(covid_data, population_dropdown.value)
hospitals = hospital_setting(hospitals, G)
distances=[10,20,30] # Distances in travel time
weights=[1.0, 0.68, 0.22] # Weights where weights[0] is applied to distances[0]
resources = ["icu_beds", "vents"] # resources
```

```
100%|      | 383911/383911 [00:07<00:00, 53023.96it/s]

Removed 0 nodes (0.0000%) from the OSMNX network
Number of nodes: 142044
Number of edges: 383911

Pop Centroid File Setting: 100%|      | 86/86 [00:00<00:00, 187.16it/s]
Find the nearest osm from hospitals: 100%|      | 66/66 [01:49<00:00,
1.66s/it]

hospital setting is done
CPU times: user 1min 28s, sys: 2.28 s, total: 1min 31s
Wall time: 2min 7s
```

[64]:
```
%%time
catchments = measure_acc_par(hospitals, pop_data, G, distances, weights,␣
↪num_proc=processor_dropdown.value)
```

```
Catchment for hospital   64 complete
CPU times: user 2.26 s, sys: 547 ms, total: 2.81 s
Wall time: 5min 39s
```

[66]:
```
%%time
for j in range(len(catchments)):
```

```
    catchments[j] = catchments[j][catchments[j][resource_dropdown.value]!
 ↪=float('inf')]
result = overlapping_function(grid_file, catchments, resource_dropdown.value,␣
 ↪weights, num_proc=processor_dropdown.value)
```

```
CPU times: user 5.31 s, sys: 364 ms, total: 5.67 s
Wall time: 17.3 s
```

[68]:
```
result.head()
```

[68]:
```
             left           top          right        bottom    id        area  \
0  440843.416087  4.638515e+06  441420.766356  4.638015e+06  4158  216661.173
1  440843.416087  4.638015e+06  441420.766356  4.637515e+06  4159  216661.168
2  440843.416087  4.639515e+06  441420.766356  4.639015e+06  4156  216661.169
3  440843.416087  4.639015e+06  441420.766356  4.638515e+06  4157  216661.171
4  440843.416087  4.640515e+06  441420.766356  4.640015e+06  4154  216661.171

                                            geometry  icu_beds     vents
0  POLYGON ((440843.416 4638265.403, 440987.754 4...  0.895678  0.201396
1  POLYGON ((440843.416 4637765.403, 440987.754 4...  0.891970  0.200602
2  POLYGON ((440843.416 4639265.403, 440987.754 4...  0.912949  0.205079
3  POLYGON ((440843.416 4638765.403, 440987.754 4...  0.895678  0.201396
4  POLYGON ((440843.416 4640265.403, 440987.754 4...  0.910950  0.204671
```

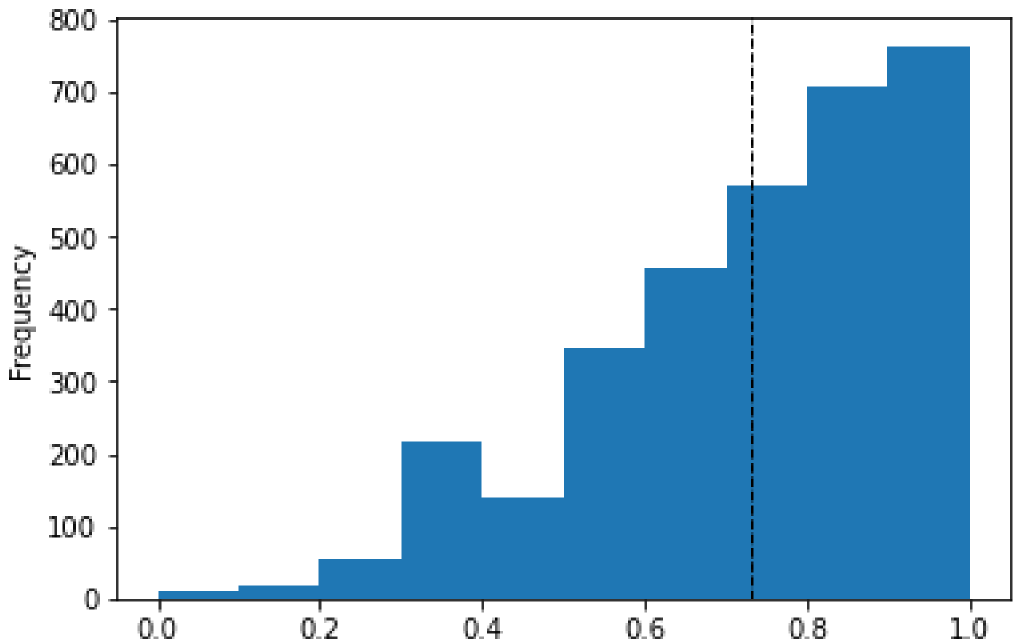
[69]:
```
result = normalization (result, resource_dropdown.value)
result.head()
```

[69]:
```
             left           top          right        bottom    id        area  \
0  440843.416087  4.638515e+06  441420.766356  4.638015e+06  4158  216661.173
1  440843.416087  4.638015e+06  441420.766356  4.637515e+06  4159  216661.168
2  440843.416087  4.639515e+06  441420.766356  4.639015e+06  4156  216661.169
3  440843.416087  4.639015e+06  441420.766356  4.638515e+06  4157  216661.171
4  440843.416087  4.640515e+06  441420.766356  4.640015e+06  4154  216661.171

                                            geometry  icu_beds     vents
0  POLYGON ((440843.416 4638265.403, 440987.754 4...  0.895678  0.895446
1  POLYGON ((440843.416 4637765.403, 440987.754 4...  0.891970  0.891812
2  POLYGON ((440843.416 4639265.403, 440987.754 4...  0.912949  0.912288
3  POLYGON ((440843.416 4638765.403, 440987.754 4...  0.895678  0.895446
4  POLYGON ((440843.416 4640265.403, 440987.754 4...  0.910950  0.910422
```

[70]:
```
# Save output to geopackage -- will name the layer according the dropdown␣
 ↪parameters
result.to_file('data/derived/public/results.gpkg',
                layer='{}_{}'.format(population_dropdown.
 ↪value,resource_dropdown.value),
                driver='GPKG')
```

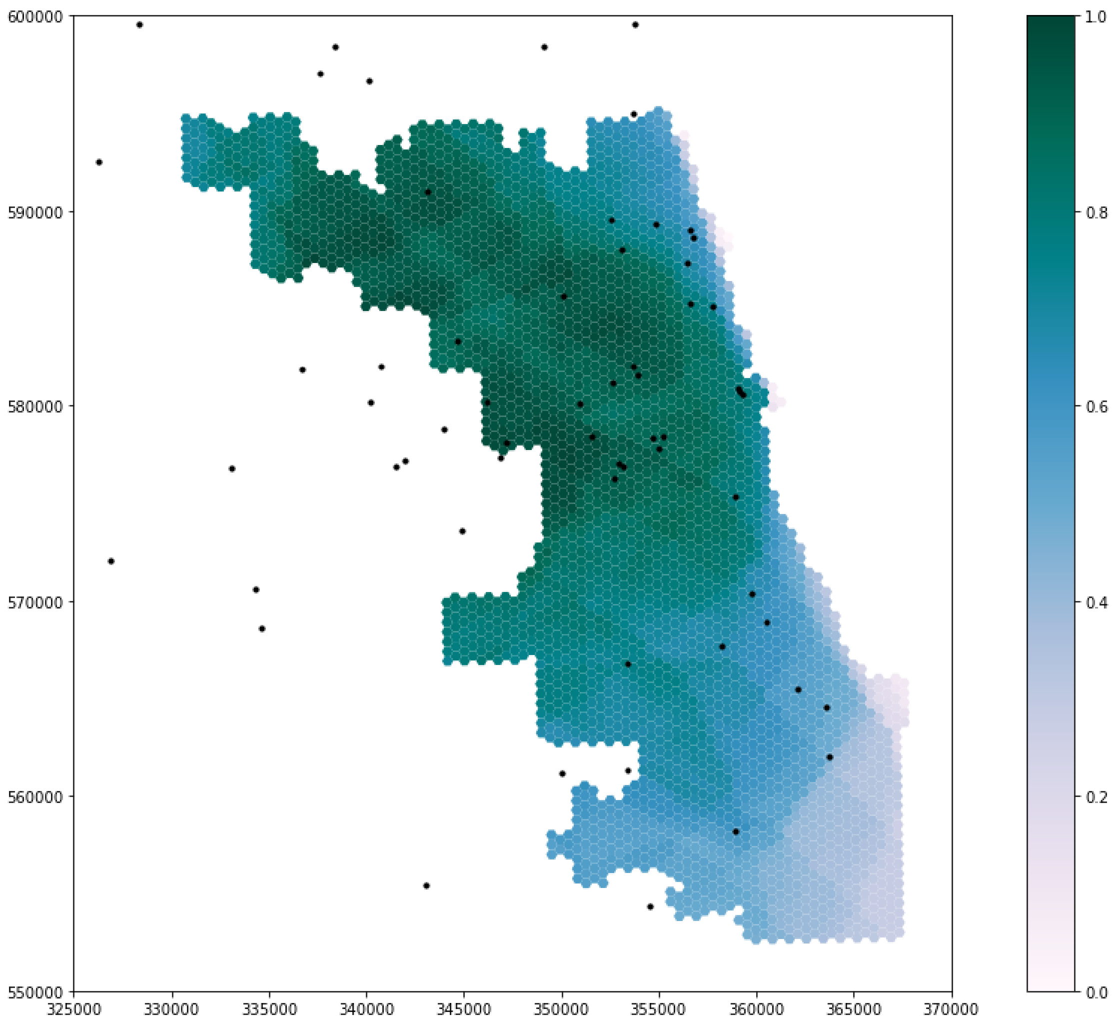
0.3.13 Plot distribution of results

[71]:
```
# If the 'Hospital ICU Beds' selection of the population dropdown has been run,␣
↪make a histogram
if hasattr(result, resource_dropdown.value):
    result['icu_beds'].plot.hist(bins=10)
    plt.axvline(result['icu_beds'].mean(), color='k', linestyle='dashed',␣
↪linewidth=1)
    plt.savefig('./results/figures/reproduction/{}_{}_histogram.png'.
↪format(population_dropdown.value,resource_dropdown.value))
else:
    print(resource_dropdown.value, "not calculated yet.\n")
```



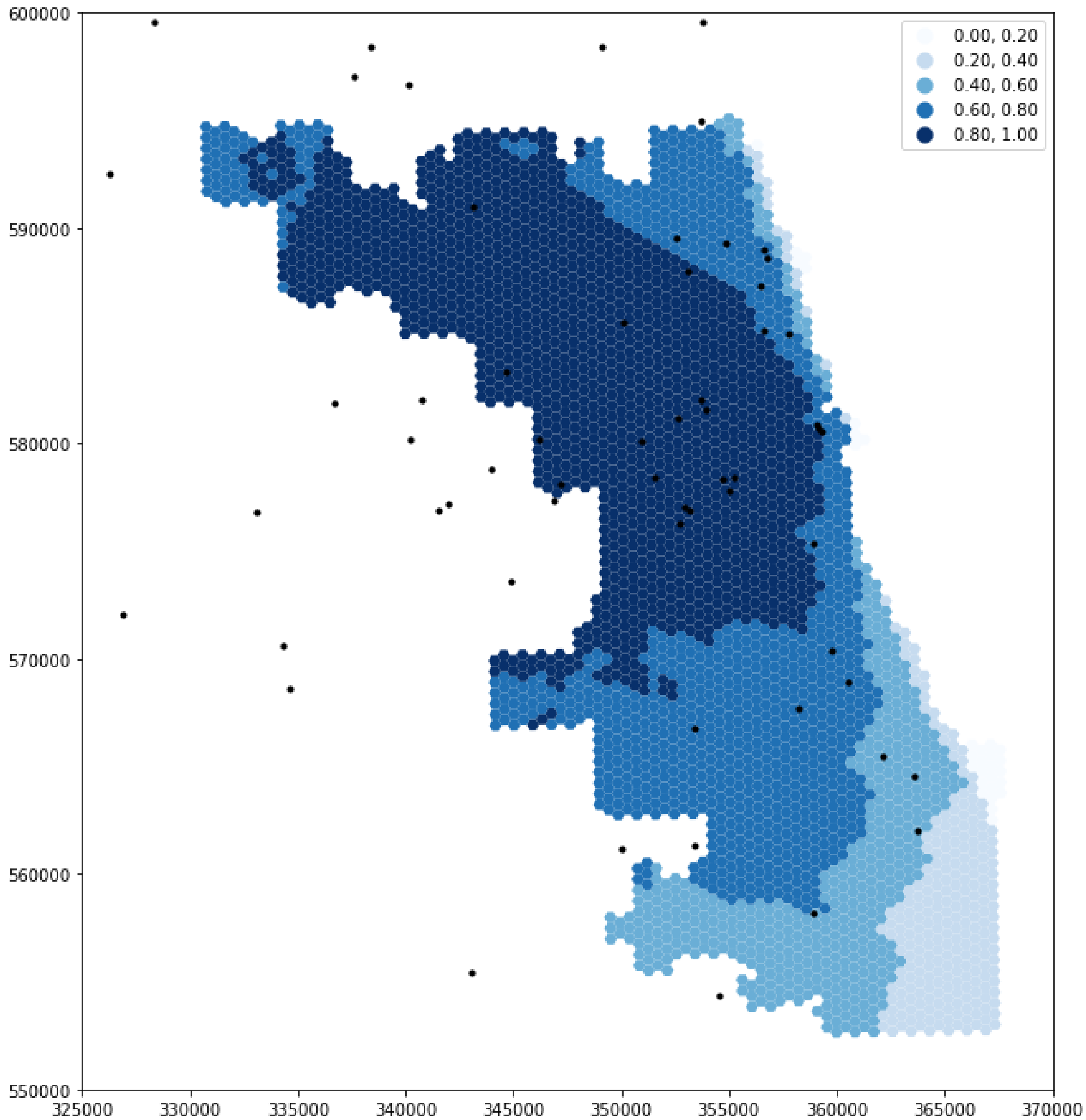
0.3.14 Unclassified Accessibility Map

[72]:
```
hospitals = hospitals.to_crs({'init': 'epsg:26971'})
result = result.to_crs({'init': 'epsg:26971'})
output_map(result, pop_data, hospitals, resource_dropdown.value)
plt.savefig('./results/figures/reproduction/{}_{}_continuous.png'.
↪format(population_dropdown.value, resource_dropdown.value))
```



0.3.15 Classified Accessibility Map

[73]:
```
output_map_classified(result, pop_data, hospitals, resource_dropdown.value)
plt.savefig('./results/figures/reproduction/{}_{}_classified.png'.
 ↪format(population_dropdown.value, resource_dropdown.value))
```



0.3.16 STOP HERE!

If you have not run the model to calculate Ventilator and ICU accessibility scores for both COVID-19 and At Risk populations (i.e: if you have not run the model four times)... Do that before you try to run the following section.

0.3.17 Comparison with Original Results

[74]:
```
# Import study results to compare
# hospital_i assumed to be for ICU and hospital_v assumed to be for ventilator
# however it's unknown whether the population is the COVID-19 population or the␣
↪AT RISK population
```

```
fp = 'data/derived/public/Chicago_ACC.shp'
og_result = gpd.read_file(fp)
og_result.set_index("id")
og_result.head()
```

[74]:
```
     id  hospital_i  hospital_v  \
0  4158    0.844249    0.843439
1  4159    0.843600    0.843031
2  4156    0.906094    0.904699
3  4157    0.877197    0.876503
4  4154    0.911424    0.910002

                                            geometry
0  POLYGON ((-87.71312 41.89411, -87.71140 41.896…
1  POLYGON ((-87.71307 41.88961, -87.71135 41.891…
2  POLYGON ((-87.71322 41.90312, -87.71150 41.905…
3  POLYGON ((-87.71317 41.89861, -87.71145 41.900…
4  POLYGON ((-87.71332 41.91212, -87.71160 41.914…
```

[75]:
```
result.set_index("id")
result_compare = result.join(og_result[["hospital_i","hospital_v"]])
result_compare.head()
```

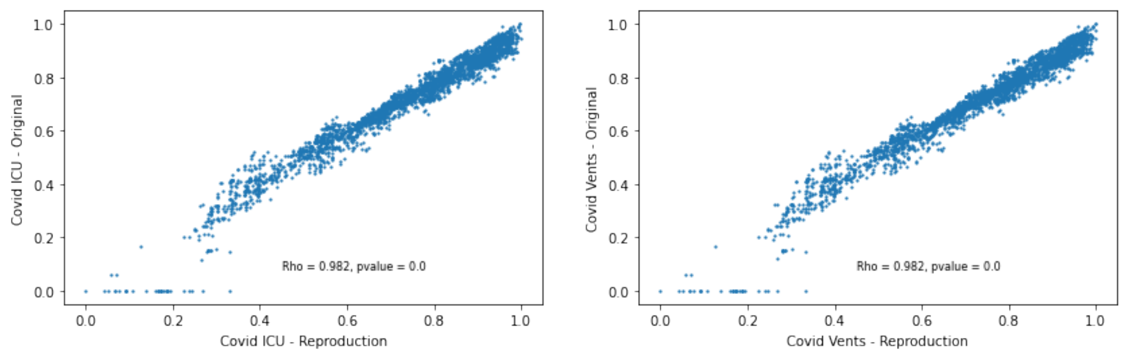
[75]:
```
            left           top          right        bottom    id        area  \
0  440843.416087  4.638515e+06  441420.766356  4.638015e+06  4158  216661.173
1  440843.416087  4.638015e+06  441420.766356  4.637515e+06  4159  216661.168
2  440843.416087  4.639515e+06  441420.766356  4.639015e+06  4156  216661.169
3  440843.416087  4.639015e+06  441420.766356  4.638515e+06  4157  216661.171
4  440843.416087  4.640515e+06  441420.766356  4.640015e+06  4154  216661.171

                                            geometry  icu_beds     vents  \
0  POLYGON ((351469.371 580527.566, 351609.858 58…  0.895678  0.895446
1  POLYGON ((351477.143 580027.445, 351617.630 58…  0.891970  0.891812
2  POLYGON ((351453.825 581527.810, 351594.311 58…  0.912949  0.912288
3  POLYGON ((351461.598 581027.688, 351602.085 58…  0.895678  0.895446
4  POLYGON ((351438.276 582528.054, 351578.761 58…  0.910950  0.910422

   hospital_i  hospital_v
0    0.844249    0.843439
1    0.843600    0.843031
2    0.906094    0.904699
3    0.877197    0.876503
4    0.911424    0.910002
```

[76]:
```
# Calculate spearman rho for ICU beds
icu_rho = stats.spearmanr(result_compare[["icu_beds", "hospital_i"]])
```

```
icu_rho = "Rho = " + str(round(icu_rho.correlation,3)) + ", pvalue = " +␣
↪str(icu_rho.pvalue)
# Calculate spearman rho for Ventilators
vents_rho = stats.spearmanr(result_compare[["vents", "hospital_v"]])
vents_rho = "Rho = " + str(round(vents_rho.correlation,3)) + ", pvalue = " +␣
↪str(vents_rho.pvalue)
print("ICU:", icu_rho,"\nVents:", vents_rho)
```

```
ICU: Rho = 0.982, pvalue = 0.0
Vents: Rho = 0.982, pvalue = 0.0
```

[77]:
```
# Create subplots
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(14,4));

axs[0].scatter(result_compare[["icu_beds"]], result_compare[["hospital_i"]],␣
↪s=1.5)
axs[0].set_xlabel("Covid ICU - Reproduction", labelpad=5)
axs[0].set_ylabel("Covid ICU - Original", labelpad=5)
axs[0].text(.45, .08, icu_rho, fontsize=8)
axs[1].scatter(result_compare[["vents"]], result_compare[["hospital_v"]], s=1.5)
axs[1].set_xlabel("Covid Vents - Reproduction", labelpad=5)
axs[1].set_ylabel("Covid Vents - Original", labelpad=5)
axs[1].text(.45, .08, vents_rho, fontsize=8)
plt.savefig("./results/figures/reproduction/rho_correlation_comparison.png")
```



0.3.18 References

Luo, W., & Qi, Y. (2009). An enhanced two-step floating catchment area (E2SFCA) method for measuring spatial accessibility to primary care physicians. Health & place, 15(4), 1100-1107.