

Geospatial Data Analysis using geopandas and Folium: A business case-study

SONIYA RANGNANI

DATA SCIENTIST, PRICELABS



Geospatial data

Data involving location information

Helps in understanding relationships between geographic attributes and any other metrics data, e.g., e.g. how does revenue vary from one country to another?

Applications:

- Visualizing the area
- Performing trade area analysis
- for a brand, Selecting locations for opening new stores
- Planning telecommunication/transportation network
- Risk assessment due to destructive weather, etc

Agenda of talk

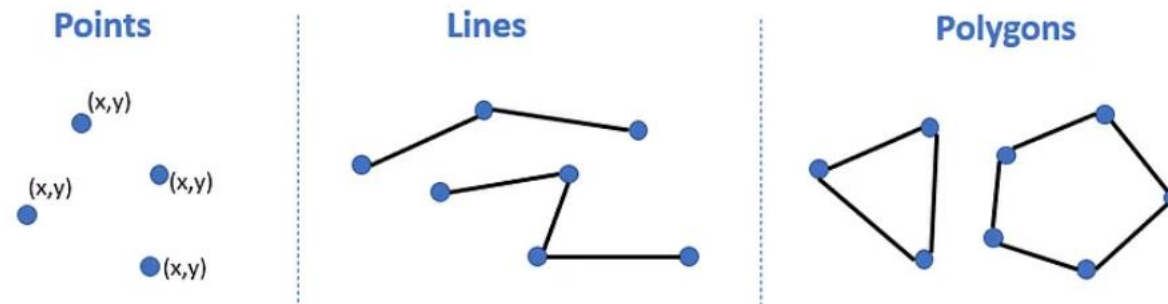
- Basics of geospatial data format
- Creating a point geometry from latitude and longitude
- Creating buffer area around a point for trade analysis
- Visualizing geospatial data using folium
- Join operation between two spatial dataframes for intersecting points

Working with Geospatial Data

Vector data represent geometries in the world.

The road network, the buildings, the restaurants, and ATMs are all vectors

Vector data is simply a collection of discrete locations ((x, y) values) called “vertices” that define one of three shapes:



Shapes of different vector data

Vector data is commonly stored in a “shapefile” format. A shapefile is composed of three required files with the same prefix (here, ‘spatial-data’) but different extensions:

- *spatial-data.shp*: main file that stores records of each shape geometries
- *spatial-data.shx*: index of how the geometries in the main file relate to one-another
- *spatial-data.dbf*: attributes of each record

Geopandas



The diagram illustrates the data format of a typical geopandas dataframe. It consists of a table with four columns: 'index' (blue), 'data' (green), 'data' (green), and 'geometry' (yellow). The first row is a header row, and the subsequent five rows represent data rows. A black redaction bar is positioned above the 'data' column headers.

index	data	data	geometry

Data format of a typical geopandas dataframe

Geo json

geopandas to work with vector data in python.

Geopandas extends the pandas capabilities to geospatial data and leverages the capabilities of shapely to perform geometric operations on spatial data.

Depends on

fiona for file access and

matplotlib for plotting.

GeoSeries and GeoDataFrame

Geometry column contains different geometries like points (latitudes and longitudes), lines, polygons, etc., as shapely objects.


US Fast Food Case Study

[Kaggle](#) Dataset

10,000 fast-food restaurants in US.

Objective.

Determine how many Burger King restaurants (a competitor of McDonald's) are in the vicinity of corresponding McDonald's restaurants.



```
## Importing Libraries
import pandas as pd
import geopandas as gpd
import folium
from pyproj import CRS
from shapely.geometry import Point
from geopy.distance import geodesic

## Loading and Formatting Dataset
df_FFR = pd.read_csv('FastFoodRestaurants.csv')
df_FFR = df_FFR[['name', 'address', 'city', 'province', 'latitude', 'longitude']]
df_FFR.rename(columns={'province': 'state'}, inplace=True)

## Creating Data Subsets
# McDonalds in NY
df_MD_NY = df_FFR.loc[(df_FFR.state == 'NY') & (df_FFR.name.isin(['McDonald\'s', 'Mcdonald\'s', 'McDonalds']))]
df_MD_NY.reset_index(drop=True, inplace=True)
# Burger King in NY
df_BK_NY = df_FFR.loc[(df_FFR.state == 'NY') & (df_FFR.name.isin(['Burger King']))]
df_BK_NY.reset_index(drop=True, inplace=True)
```


Creating Point and Buffer Area

Convert these into geodataframes by creating a point geometry object from latitude and longitude.

```
#create_point - combines lat/long to a single point
#Function to create a point based on the latitude and longitude columns in a dataframe. Returns original dataframe with point geometry as
a column
def create_point (df, Long, Lat):
    crs = CRS("epsg:4326")
    geometry = [Point(xy) for xy in zip(df[Long], df[Lat])]
    df = gpd.GeoDataFrame(df, geometry=geometry).set_crs(crs, allow_override=True)
    df.rename(columns={'geometry': 'Centroid'}, inplace=True)
    return df

df_MD_NY = create_point(df_MD_NY, 'longitude', 'latitude')
df_BK_NY = create_point(df_BK_NY, 'longitude', 'latitude')
```

As it can be seen below, a new column 'Centroid' has been created which is a Point geometry data type and the class of the dataframe is converted to GeoDataFrame.

```
df_MD_NY.head()
```

	name	address	city	state	latitude	longitude	Centroid
0	McDonald's	324 Main St	Massena	NY	44.921300	-74.89021	POINT (-74.89021 44.92130)
1	McDonald's	6098 State Highway 37	Massena	NY	44.950080	-74.84553	POINT (-74.84553 44.95008)
2	McDonald's	603 E Main St	Endicott	NY	42.095910	-76.05453	POINT (-76.05453 42.09591)
3	McDonald's	1134 Paterson St	Ogdensburg	NY	44.690205	-75.47740	POINT (-75.47740 44.69020)
4	McDonald's	51-67 E 161st St	Bronx	NY	40.828053	-73.92529	POINT (-73.92529 40.82805)

```
df_MD_NY.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 70 entries, 0 to 69
Data columns (total 7 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   name        70 non-null    object
1   address     70 non-null    object
2   city        70 non-null    object
3   state       70 non-null    object
4   latitude    70 non-null    float64
5   longitude   70 non-null    float64
6   Centroid    70 non-null    geometry
```

```
# create_buffer_area - takes a list of radii and creates a buffer area based on points in a dataframe
# Need to specify the geometry column in the dataframe when defining the function
# When crs is set to epsg:2272, this is commonly used for the measure of feet,
# which can be easily converted into miles (5280ft =1 mile)
# Reference for crs is https://epsg.io
def create_buffer_area(df, geometry_col, radius_list):
    gdf = gpd.GeoDataFrame(df, geometry=geometry_col)
    gdf.to_crs(epsg=2272, inplace=True)
    rdf = gpd.GeoDataFrame()
    for r in radius_list:
        tadf = pd.merge(df,
                        gpd.GeoDataFrame(gdf.geometry.buffer(r*5280)),
                        left_index=True,
                        right_index=True)
        tadf['Radius'], tadf['Radius_Type'] = r, 'Mile'
        rdf = pd.concat([rdf, tadf])
    rdf.rename(columns={0: 'Buffer_Area'}, inplace=True)
    rdf = gpd.GeoDataFrame(rdf, geometry=geometry_col)
    rdf.to_crs(epsg=4326, inplace=True)
    return rdf

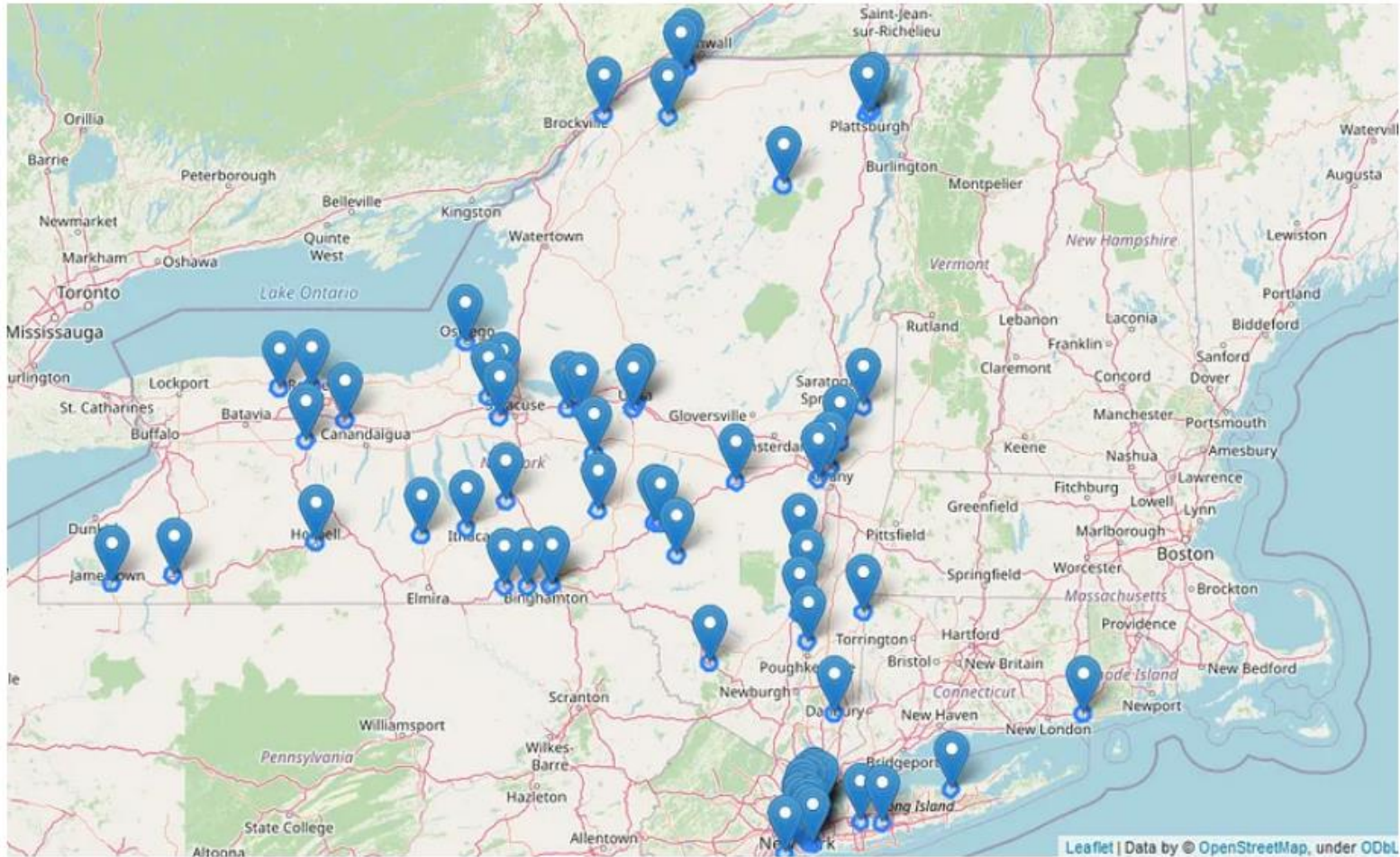
radii = [3] # list of trade area radius
df_MD_NY = create_buffer_area(df_MD_NY, 'Centroid', radii)
```

	name	address	city	state	latitude	longitude	Centroid	Buffer_Area	Radius	Radius_Type
0	McDonald's	324 Main St	Massena	NY	44.921300	-74.89021	POINT (-74.89021 44.92130)	POLYGON ((2727277.630 2050026.623, 2727201.356...	3	Mile
1	McDonald's	6098 State Highway 37	Massena	NY	44.950080	-74.84553	POINT (-74.84553 44.95008)	POLYGON ((2738534.665 2060925.400, 2738458.391...	3	Mile
2	McDonald's	603 E Main St	Endicott	NY	42.095910	-76.05453	POINT (-76.05453 42.09591)	POLYGON ((2444652.894 1010979.684, 2444576.620...	3	Mile
3	McDonald's	1134 Paterson St	Ogdensburg	NY	44.690205	-75.47740	POINT (-75.47740 44.69020)	POLYGON ((2576945.348 1961120.964, 2576869.074...	3	Mile
4	McDonald's	51-67 E 161st St	Bronx	NY	40.828053	-73.92529	POINT (-73.92529 40.82805)	POLYGON ((3042476.231 567435.029, 3042399.957 ...	3	Mile

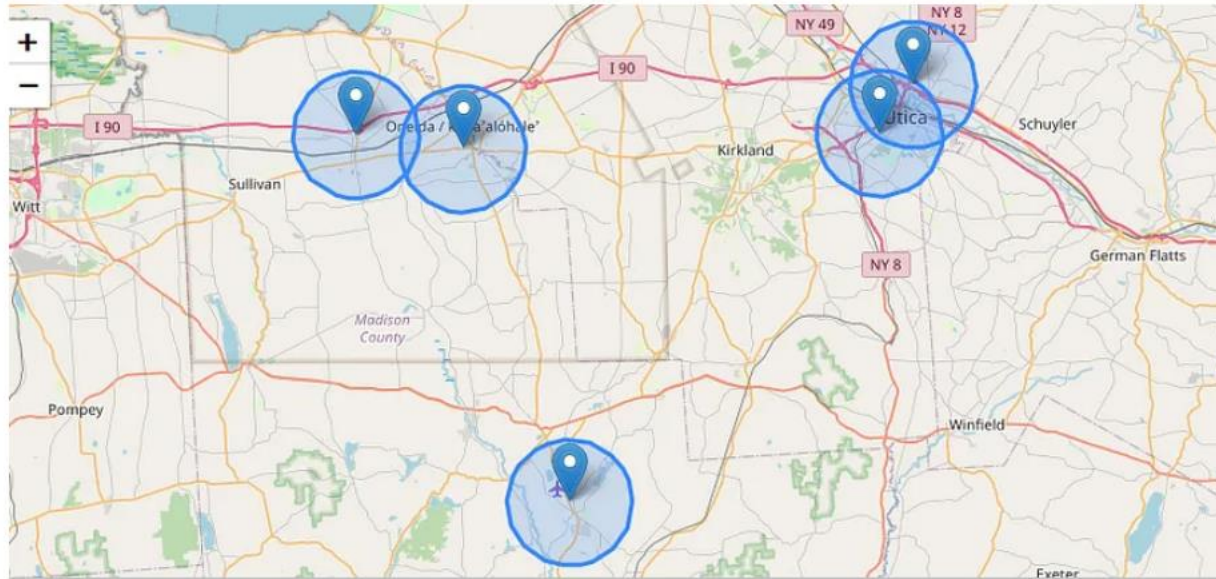
Visualizing Geospatial Data

```
#plot_point_polygon - plots a latitude, longitudes, and trade area from a dataframe onto a map
def plot_point_polygon (df, latitude, longitude, trade_area):
    #Create base map and centers the Map
    middle_long = np.median(df[longitude])
    middle_lat = np.median(df[latitude])
    m = folium.Map(location=[middle_lat, middle_long], width='80%', height='80%', tiles='OpenStreetMap', zoom_start=7)
    #m = folium.Map(location=[39.828395, -98.579480], zoom_start=5)
    #Plot Latitude & Longitude
    for idx, r in df.iterrows():
        folium.Marker([r[latitude], r[longitude]]).add_to(m)
    #Plot polygons
    polys = df[trade_area]
    folium.GeoJson(polys.to_crs(epsg=4326)).add_to(m)
    return m

ffr_map = plot_point_polygon(df_MD_NY, 'latitude', 'longitude', 'Buffer_Area')
ffr_map
```

McDonald's in New York State



Polygons are drawn around 3 miles from all the McDonald's

Spatial Join

In this section, we will find the Burger King restaurant points that fall within corresponding buffer areas of the McDonalds' using Geopandas' spatial join function. Just like Pandas' join operation, this one also involves joining two geodataframes both having at least one geometry type variable. However, there are a few additional things that are noteworthy to mention:

- CRS units for geometry objects from both dataframes should match.
- The geometry objects in each data frame to be joined should both be named 'geometry' or 'set_geometry' options should be used to denote the primary geometry object in case there are multiple geometry columns in the dataframe.
- After performing the inner spatial join operation, only the geometry object data from the left data frame is retained and the other one is discarded.


```

#setgeometry - quickly create a geometry column and set the crs
def setgeometry(df, geometry_variable, crs_type):
    df['geometry'] = df[geometry_variable]
    df = df.set_geometry("geometry")
    df.to_crs(epsg=crs_type, inplace=True)
    return df

#spatial_match will find points and/or radii that fall within or overlap with another radius
def spatial_match(df1,geometry1_col,df2,geometry2_col,crs_type=2272,how_type='inner'):
    df1 = setgeometry(df1,geometry1_col,crs_type)
    df2 = setgeometry(df2,geometry2_col,crs_type)
    new_df = gpd.sjoin(df1,df2,how=how_type, op='intersects')
    return new_df

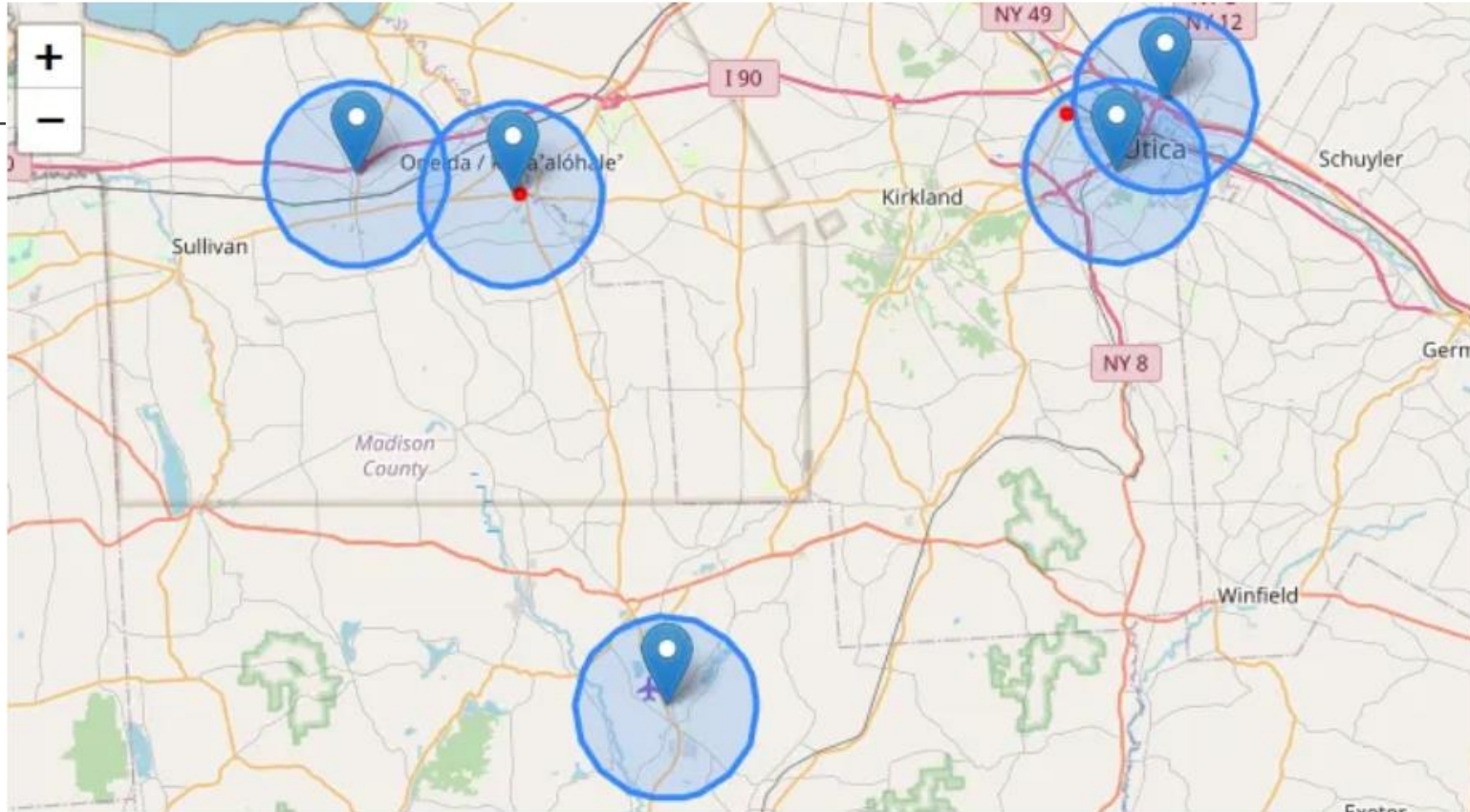
# Renaming columns before spatial join to avoid ambiguity
df_MD_NY.columns = ['MD_' + col for col in df_MD_NY.columns]
df_BK_NY.columns = ['BK_' + col for col in df_BK_NY.columns]

# Use spatial_match to find the Burger Kings that are located within 3 miles of McDonald's in New York.
BK_near_MD_NY = spatial_match(df_MD_NY,'MD_Buffer_Area',df_BK_NY,'BK_Centroid')
BK_near_MD_NY = BK_near_MD_NY[['MD_latitude','MD_longitude','MD_Centroid','MD_Buffer_Area','BK_latitude','BK_longitude','BK_Centroid']]
BK_near_MD_NY.reset_index(drop=True,inplace=True)

# Marking the intersecting Burger King points as red points in the map
for idx, r in BK_near_MD_NY.iterrows():
    folium.CircleMarker([r['BK_latitude'], r['BK_longitude']],radius=2,color='red').add_to(ffr_map)

ffr_map

```



Red points indicate the locations of Burger Kings

Final words

We introduces the concept of geospatial analysis, geopandas and other resourceful open-source python spatial libraries. We have covered common spatial operations like creating geometry points, creating buffer areas, spatial joins, and visualizing geospatial data on maps