```
pip install geopandas osmnx shapely
```

```
Requirement already satisfied: geopandas in /usr/local/lib/python3.11/dist-packages (1.0.1)
Requirement already satisfied: osmnx in /usr/local/lib/python3.11/dist-packages (2.0.3)
Requirement already satisfied: shapely in /usr/local/lib/python3.11/dist-packages (2.1.0)
Requirement already satisfied: numpy>=1.22 in /usr/local/lib/python3.11/dist-packages (from geopandas) (2.0.2)
Requirement already satisfied: pyogrio>=0.7.2 in /usr/local/lib/python3.11/dist-packages (from geopandas) (0.10.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from geopandas) (24.2)
Requirement already satisfied: pandas>=1.4.0 in /usr/local/lib/python3.11/dist-packages (from geopandas) (2.2.2)
Requirement already satisfied: pyproj>=3.3.0 in /usr/local/lib/python3.11/dist-packages (from geopandas) (3.7.1)
Requirement already satisfied: networkx>=2.5 in /usr/local/lib/python3.11/dist-packages (from osmnx) (3.4.2)
Requirement already satisfied: requests>=2.27 in /usr/local/lib/python3.11/dist-packages (from osmnx) (2.32.3)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.4.0->geopandas) (2.9.0.
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.4.0->geopandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.4.0->geopandas) (2025.2)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from pyogrio>=0.7.2->geopandas) (2025.4.26)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.27->osmnx) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.27->osmnx) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.27->osmnx) (2.4.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas>=1.4.0->geopanda
```

```
import geopandas as gpd
import numpy as np
from shapely.geometry import Point

# Load the relevant datasets
wards = gpd.read_file("2023_boundary.shp")
schools_m = gpd.read_file("schools.shp")  # Assuming you have a school shapefile
gang_boundaries = gpd.read_file("gang_boundaries.shp")  # Assuming you have gang boundary shapefile

# Create regular grid points within city bounds
minx, miny, maxx, maxy = wards.total_bounds
x_coords = np.linspace(minx, maxx, 100)  # Adjust grid size here (e.g., 100 points in x direction)
y_coords = np.linspace(miny, maxy, 100)  # Adjust grid size here (e.g., 100 points in y direction)
grid_points = [Point(x, y) for x in x_coords for y in y_coords]

# Create GeoDataFrame for grid points and filter by ward boundaries
grid_gdf = gpd.GeoDataFrame(geometry=grid_points, crs=wards.crs)
grid_gdf = grid_gdf[grid_gdf.geometry.within(wards.unary_union)]

# 1. Add Schools Covariate (e.g., number of schools within 500m of each grid cell)
grid_gdf["Schools_500m"] = grid_gdf.geometry.apply(
    lambda point: schools_m[schools_m.geometry.distance(point) <= 500].shape[0]
)

# 2. Add Gang Territory Covariate (e.g., 1 if inside gang boundary, 0 if not)
grid_gdf["Inside_Gang"] = grid_gdf.geometry.apply(
    lambda point: int(gang_boundaries.geometry.intersects(point).any())
)

# 3. Add Temporal Covariates (e.g., assume a fixed year for simplicity)
grid_gdf["Year"] = 2022  # Example: you could get this from the shooting data or set as constant
grid_gdf["Month"] = 5  # Example: a fixed month, or extract this from your data
grid_gdf["DayOfWeek"] = 2  # Example: a fixed day of the week (0=Monday, 6=Sunday)

# Now grid_gdf contains both spatial and temporal covariates
```

```
---------------------------------------------------------------------------
DataSourceError                           Traceback (most recent call last)
<ipython-input-9-0328fc375614> in <cell line: 0>()
      5 # Load the relevant datasets
      6 wards = gpd.read_file("2023_boundary.shp")
----> 7 schools_m = gpd.read_file("schools.shp")  # Assuming you have a school shapefile
      8 gang_boundaries = gpd.read_file("gang_boundaries.shp")  # Assuming you have gang boundary shapefile
      9

                                 3 frames
/usr/local/lib/python3.11/dist-packages/pyogrio/raw.py in read(path_or_buffer, layer, encoding, columns, read_geometry, force_2d,
skip_features, max_features, where, bbox, mask, fids, sql, sql_dialect, return_fids, datetime_as_string, **kwargs)
    196     dataset_kwargs = _preprocess_options_key_value(kwargs) if kwargs else {}
    197
--> 198     return ogr_read(
    199         get_vsi_path_or_buffer(path_or_buffer),
    200         layer=layer,

pyogrio/_io.pyx in pyogrio._io.ogr_read()

pyogrio/_io.pyx in pyogrio._io.ogr_open()

DataSourceError: schools.shp: No such file or directory
```

Data set up for IPPP model

```
import geopandas as gpd
import numpy as np
from shapely.geometry import Point
import numpy as np
import geopandas as gpd
from shapely.geometry import Point
import matplotlib.pyplot as plt


# Load ward boundaries (make sure it's in EPSG:4326 or projected CRS)
wards = gpd.read_file("2023_boundary.shp")

# Create regular grid points within city bounds
minx, miny, maxx, maxy = wards.total_bounds
x_coords = np.linspace(minx, maxx, 100)
y_coords = np.linspace(miny, maxy, 100)
grid_points = [Point(x, y) for x in x_coords for y in y_coords]

# Create GeoDataFrame and keep only points inside the ward boundaries
grid_gdf = gpd.GeoDataFrame(geometry=grid_points, crs=wards.crs)
grid_gdf = grid_gdf[grid_gdf.geometry.within(wards.unary_union)]
```

```
<ipython-input-10-da4213a21748>:21: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  grid_gdf = grid_gdf[grid_gdf.geometry.within(wards.unary_union)]
```

```
grid_gdf
```

	geometry
182	POINT (-87.93573 41.95804)
183	POINT (-87.93573 41.96187)
184	POINT (-87.93573 41.96569)
185	POINT (-87.93573 41.96951)
186	POINT (-87.93573 41.97334)
...	...
9818	POINT (-87.52834 41.71336)
9819	POINT (-87.52834 41.71718)
9820	POINT (-87.52834 41.72101)
9821	POINT (-87.52834 41.72483)
9822	POINT (-87.52834 41.72865)

4042 rows × 1 columns

Viewing shapefile

```
wards
```

	ward	objectid	date_edit_	time_edit_	ward_id	globalid	st_area_sh	st_length_	geometry
0	1.0	51.0	2022-06-01	00:00:00.000	1.0	{DB2A2A7D-FAF1-42A4-B061-AE18C31A80BB}	6.589346e+07	61878.821587	POLYGON ((-87.68777 41.92858, -87.68776 41.928...
1	2.0	52.0	2022-06-01	00:00:00.000	2.0	{88F300F6-D6DF-4337-8DE3-0C2D27A5B338}	3.128511e+07	74175.949239	POLYGON ((-87.62517 41.904, -87.625 41.904, -8...
2	3.0	53.0	2022-06-01	00:00:00.000	3.0	{4F169974-0F93-4C35-98C1-20A8415C72F5}	9.777341e+07	85625.597620	POLYGON ((-87.61992 41.86577, -87.61985 41.865...
3	4.0	54.0	2022-06-01	00:00:00.000	4.0	{02DC7BBB-BD1E-413C-994E-0A3154FD3D54}	1.158091e+08	103594.867171	POLYGON ((-87.62429 41.877, -87.62429 41.87696...
4	5.0	55.0	2022-06-01	00:00:00.000	5.0	{0A109A41-9DED-47D7-934E-1EA1CC7EE025}	1.120803e+08	88207.690241	POLYGON ((-87.5603 41.76636, -87.56058 41.7663...
5	6.0	56.0	2022-06-01	00:00:00.000	6.0	{FD74A999-4BBA-4CE3-BEBB-CC76423037E8}	1.392022e+08	80779.851890	POLYGON ((-87.61794 41.77292, -87.61786 41.772...
6	7.0	57.0	2022-06-01	00:00:00.000	7.0	{279FCBD9-EA0D-4FFC-A8CA-EDA2676C0721}	1.414924e+08	98906.567862	POLYGON ((-87.54393 41.7603, -87.5437 41.76009...
7	8.0	58.0	2022-06-01	00:00:00.000	8.0	{2B5945BD-349C-4088-9B25-D99D4FB6C01D}	1.799926e+08	77255.568224	POLYGON ((-87.58587 41.76424, -87.58597 41.764...
8	9.0	59.0	2022-06-01	00:00:00.000	9.0	{B75B90E6-CB95-4890-BBC3-D2F313E2EE80}	2.344895e+08	102227.294779	POLYGON ((-87.61414 41.72189, -87.61414 41.721...
9	10.0	60.0	2022-06-01	00:00:00.000	10.0	{19FF2026-6DDD-45DA-8DF6-B44B0C15834C}	5.773395e+08	147455.404991	POLYGON ((-87.53018 41.7431, -87.53008 41.7424...
10	11.0	61.0	2022-06-01	00:00:00.000	11.0	{C7397C87-C4AE-4C48-A00E-50D2CD1100CF}	1.273484e+08	64972.869726	POLYGON ((-87.636 41.81635, -87.63602 41.81635...
11	12.0	62.0	2022-06-01	00:00:00.000	12.0	{EAD019CB-AA66-48C4-9338-795B4E083B59}	1.334616e+08	76139.484398	POLYGON ((-87.66588 41.84493, -87.66581 41.844...
12	13.0	63.0	2022-06-01	00:00:00.000	13.0	{883D9536-9DDE-4FB8-B340-E9A644FD1851}	1.480001e+08	59975.600111	POLYGON ((-87.74205 41.77113, -87.74206 41.771...
13	14.0	64.0	2022-06-01	00:00:00.000	14.0	{612295DC-E128-468C-B0C9-A1ADE1CB1C84}	1.213109e+08	72296.250663	POLYGON ((-87.69421 41.80828, -87.6942 41.8080...
14	15.0	65.0	2022-06-01	00:00:00.000	15.0	{51F05987-1A65-4707-B17E-98C8709061B1}	8.307862e+07	107657.780595	POLYGON ((-87.63584 41.80905, -87.63585 41.809...
15	16.0	66.0	2022-06-01	00:00:00.000	16.0	{B54690B6-21DF-44F5-B898-D73EF9976BDC}	1.258524e+08	90382.506038	POLYGON ((-87.64077 41.77616, -87.64082 41.776...
16	17.0	67.0	2022-06-01	00:00:00.000	17.0	{A0475DAB-473E-447B-9F82-AEC0CA483C93}	1.301704e+08	90154.781522	POLYGON ((-87.65421 41.76869, -87.65421 41.768...
17	18.0	68.0	2022-06-01	00:00:00.000	18.0	{E22B6995-91F8-4C49-A9F3-4085D503F2A4}	1.933789e+08	79337.751930	POLYGON ((-87.6583 41.7395, -87.65834 41.7395,...
18	19.0	69.0	2022-06-01	00:00:00.000	19.0	{91001AF5-B4E0-4F45-A1CB-7C22070BA22D}	2.199899e+08	101405.125517	POLYGON ((-87.66674 41.73022, -87.66677 41.730...
19	20.0	70.0	2022-06-01	00:00:00.000	20.0	{1EA93731-81AC-42D6-9711-283C3A3CA2BE}	1.453499e+08	100221.251179	POLYGON ((-87.62082 41.79383, -87.6205 41.7938...
20	21.0	71.0	2022-06-01	00:00:00.000	21.0	{BA7C81E8-7DEB-46DF-8E71-95F9AE112FF3}	2.118767e+08	92642.006031	POLYGON ((-87.62405 41.73014, -87.6241 41.7301...
21	22.0	72.0	2022-06-01	00:00:00.000	22.0	{0705D4DE-80BF-473A-B3B7-078D63AD0298}	1.055096e+08	60413.987154	POLYGON ((-87.70761 41.84995, -87.70761

									41.849...
22	23.0	73.0	2022-06-01	00:00:00.000	23.0	{51BB999E-306A-4091-8F7E-4241AACA3340}	1.168305e+08	94602.226753	POLYGON ((-87.71024 41.77156, -87.71039 41.771...
23	24.0	74.0	2022-06-01	00:00:00.000	24.0	{9ED052F7-F703-4A2D-A829-363FB60472C4}	1.120000e+08	68816.597232	POLYGON ((-87.6907 41.85635, -87.69077 41.8563...

Each point assigned to ward

```
# Ensure 'wards' has a column named 'WARDNO' (or similar)
# You can print to verify: print(wards.columns)

# Spatial join to assign each grid point its ward
grid_gdf = gpd.sjoin(grid_gdf, wards[['ward', 'geometry']], how="left", predicate="within")

# Check how many got assigned successfully
print("Unassigned points:", grid_gdf["ward"].isna().sum())
```

```
Unassigned points: 0
```

```
# Compute distance to nearest gang boundary
grid_gdf["dist_to_gang"] = grid_gdf.geometry.apply(lambda point: gang_boundaries.geometry.distance(point).min())

# Compute distance to nearest school (combine public and private schools)
all_schools = pd.concat([private_school_gdf, public_school_gdf])
grid_gdf["dist_to_school"] = grid_gdf.geometry.apply(lambda point: all_schools.geometry.distance(point).min())
```

Show hidden output

```
import geopandas as gpd
import os
import pandas as pd
from shapely.geometry import Point



# Initialize empty list to collect GeoDataFrames
gang_boundaries = []

# Loop through each year
for year in range(2007, 2025):
    filename = os.path.join(f"{year}_Gang_Boundaries.shp")
    if os.path.exists(filename):
        try:
            gdf = gpd.read_file(filename)
            gdf["year"] = year  # Optionally track source year
            gang_boundaries.append(gdf)
            print(f"✅ Loaded: {filename}")
        except Exception as e:
            print(f"⚠️ Failed to load {filename}: {e}")
    else:
        print(f"⛔ File not found: {filename}")

# Concatenate all into one GeoDataFrame
if gang_boundaries:
    gang_boundaries = gpd.GeoDataFrame(pd.concat(gang_boundaries, ignore_index=True), crs=gang_boundaries[0].crs)
    print(f"\n✅ Combined gang boundaries from {len(gang_boundaries)} years.")
else:
    print("❌ No shapefiles loaded.")
```

```
✅ Loaded: 2007_Gang_Boundaries.shp
✅ Loaded: 2008_Gang_Boundaries.shp
✅ Loaded: 2009_Gang_Boundaries.shp
✅ Loaded: 2010_Gang_Boundaries.shp
✅ Loaded: 2011_Gang_Boundaries.shp
✅ Loaded: 2012_Gang_Boundaries.shp
⛔ File not found: 2013_Gang_Boundaries.shp
✅ Loaded: 2014_Gang_Boundaries.shp
✅ Loaded: 2015_Gang_Boundaries.shp
✅ Loaded: 2016_Gang_Boundaries.shp
✅ Loaded: 2017_Gang_Boundaries.shp
```

```
✅ Loaded: 2018_Gang_Boundaries.shp
✅ Loaded: 2019_Gang_Boundaries.shp
✅ Loaded: 2020_Gang_Boundaries.shp
✅ Loaded: 2021_Gang_Boundaries.shp
✅ Loaded: 2022_Gang_Boundaries.shp
✅ Loaded: 2023_Gang_Boundaries.shp
✅ Loaded: 2024_Gang_Boundaries.shp

✅ Combined gang boundaries from 943 years.
```

gang_boundaries

	OBJECTID	GANG_NAME	SHAPE__Are	SHAPE__Len	geometry	year	Shape__Are	Shape__Len	FID	SHAPE_AREA	SHAPE_LEN
0	1.0	LATIN COUNTS	3.330409e+06	20109.113362	MULTIPOLYGON (((-9758875.461 5134610.943, -975...	2007	NaN	NaN	NaN	NaN	NaN
1	2.0	LATIN DRAGONS	1.738342e+06	13610.756236	MULTIPOLYGON (((-9743422.639 5109241.779, -974...	2007	NaN	NaN	NaN	NaN	NaN
2	3.0	LATIN EAGLES	3.177308e+05	2867.456277	POLYGON ((-9766764.606 5148587.921, -9766763.9...	2007	NaN	NaN	NaN	NaN	NaN
3	4.0	LATIN JIVERS	7.516058e+04	1641.799169	MULTIPOLYGON (((-9762338.21 5146450.809, -9762...	2007	NaN	NaN	NaN	NaN	NaN
					MULTIPOLYGON						

```
# Compute distance to nearest gang boundary
grid_gdf["dist_to_gang"] = grid_gdf.geometry.apply(lambda point: gang_boundaries.geometry.distance(point).min())

# Compute distance to nearest school (combine public and private schools)
all_schools = pd.concat([private_school_gdf, public_school_gdf])
grid_gdf["dist_to_school"] = grid_gdf.geometry.apply(lambda point: all_schools.geometry.distance(point).min())
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-20-5852a267cfa2> in <cell line: 0>()
      1 # Compute distance to nearest gang boundary
----> 2 grid_gdf["dist_to_gang"] = grid_gdf.geometry.apply(lambda point: gang_boundaries.geometry.distance(point).min())
      3
      4 # Compute distance to nearest school (combine public and private schools)
      5 all_schools = pd.concat([private_school_gdf, public_school_gdf])

                                  14 frames
lib.pyx in pandas._libs.lib.map_infer()

/usr/local/lib/python3.11/dist-packages/shapely/measurement.py in distance(a, b, **kwargs)
     79
     80     """
---> 81     return lib.distance(a, b, **kwargs)
     82
     83

KeyboardInterrupt:
```

```
import geopandas as gpd
from shapely.geometry import Point
import pandas as pd

# --- Load and Filter School Datasets ---
private_school_df = pd.read_csv("Private_schools.csv")
private_school_df = private_school_df[private_school_df["CITY"].str.strip().str.upper() == "CHICAGO"]
chicago_school_df = pd.read_csv("Chicago_schools.csv")

# Convert to GeoDataFrames
private_school_gdf = gpd.GeoDataFrame(
    private_school_df,
    geometry=[Point(xy) for xy in zip(private_school_df["LON"], private_school_df["LAT"])],
```

```
    crs="EPSG:4326"
)

public_school_gdf = gpd.GeoDataFrame(
    chicago_school_df,
    geometry=[Point(xy) for xy in zip(chicago_school_df["Long"], chicago_school_df["Lat"])],
    crs="EPSG:4326"
)

# Add school type labels
private_school_gdf["School_Type"] = "Private"
public_school_gdf["School_Type"] = "Public"

# Combine into one GeoDataFrame
all_schools = pd.concat([private_school_gdf, public_school_gdf], ignore_index=True)



import geopandas as gpd
import pandas as pd
from shapely.geometry import Point

# Load the gun violence dataset
gun_df = pd.read_csv("chicago_shooting.csv")

# Drop rows with missing coordinates
gun_df = gun_df.dropna(subset=["LONGITUDE", "LATITUDE"])

# Convert to GeoDataFrame
all_gun_points = gpd.GeoDataFrame(
    gun_df,
    geometry=gpd.points_from_xy(gun_df["LONGITUDE"], gun_df["LATITUDE"]),
    crs="EPSG:4326"  # Assuming WGS84, adjust if needed
)

# Reproject to match grid_gdf (meters, for distance calculation)
all_gun_points = all_gun_points.to_crs(grid_gdf.crs)
```

```
<ipython-input-21-55db9539993c>:6: DtypeWarning: Columns (25,26,27) have mixed types. Specify dtype option on import or set low_memory=F
  gun_df = pd.read_csv("chicago_shooting.csv")
```

```
import geopandas as gpd
import numpy as np
from shapely.geometry import Point

# Load the relevant datasets
wards = gpd.read_file("2023_boundary.shp")



# Create regular grid points within city bounds
minx, miny, maxx, maxy = wards.total_bounds
x_coords = np.linspace(minx, maxx, 100)  # Adjust grid size here (e.g., 100 points in x direction)
y_coords = np.linspace(miny, maxy, 100)  # Adjust grid size here (e.g., 100 points in y direction)
grid_points = [Point(x, y) for x in x_coords for y in y_coords]

# Create GeoDataFrame for grid points and filter by ward boundaries
grid_gdf = gpd.GeoDataFrame(geometry=grid_points, crs=wards.crs)
grid_gdf = grid_gdf[grid_gdf.geometry.within(wards.unary_union)]

# 1. Add Schools Covariate (e.g., number of schools within 500m of each grid cell)
grid_gdf["Schools_500m"] = grid_gdf.geometry.apply(
    lambda point: schools_m[schools_m.geometry.distance(point) <= 500].shape[0]
)

# 2. Add Gang Territory Covariate (e.g., 1 if inside gang boundary, 0 if not)
grid_gdf["Inside_Gang"] = grid_gdf.geometry.apply(
    lambda point: int(gang_boundaries.geometry.intersects(point).any())
)

# 3. Add Temporal Covariates (e.g., assume a fixed year for simplicity)
grid_gdf["Year"] = 2022  # Example: you could get this from the shooting data or set as constant
grid_gdf["Month"] = 5  # Example: a fixed month, or extract this from your data
grid_gdf["DayOfWeek"] = 2  # Example: a fixed day of the week (0=Monday, 6=Sunday)
```

```
# Now grid_gdf contains both spatial and temporal covariates
```

```
<ipython-input-24-abd00a7d791f>:18: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  grid_gdf = grid_gdf[grid_gdf.geometry.within(wards.unary_union)]
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-24-abd00a7d791f> in <cell line: 0>()
     19
     20 # 1. Add Schools Covariate (e.g., number of schools within 500m of each grid cell)
---> 21 grid_gdf["Schools_500m"] = grid_gdf.geometry.apply(
     22     lambda point: schools_m[schools_m.geometry.distance(point) <= 500].shape[0]
     23 )

 ---------------------------------------- 7 frames ----------------------------------------
lib.pyx in pandas._libs.lib.map_infer()

<ipython-input-24-abd00a7d791f> in <lambda>(point)
     20 # 1. Add Schools Covariate (e.g., number of schools within 500m of each grid cell)
     21 grid_gdf["Schools_500m"] = grid_gdf.geometry.apply(
---> 22     lambda point: schools_m[schools_m.geometry.distance(point) <= 500].shape[0]
     23 )
     24

NameError: name 'schools_m' is not defined
```

```
# Count gun incidents near each grid point (within 500m)
#grid_gdf["incident_count"] = grid_gdf.geometry.apply(
    ###lambda pt: all_gun_points.geometry.distance(pt).lt(500).sum()
#)
grid_gdf["incident_count"] = grid_gdf.geometry.apply(
    lambda pt: all_gun_points.geometry.distance(pt).lt(750).sum()
)
```

```
<ipython-input-25-d3e7e072c257>:6: UserWarning: Geometry is in a geographic CRS. Results from 'distance' are likely incorrect. Use 'GeoS

  lambda pt: all_gun_points.geometry.distance(pt).lt(750).sum()
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-25-d3e7e072c257> in <cell line: 0>()
      3     ###lambda pt: all_gun_points.geometry.distance(pt).lt(500).sum()
      4 #)
----> 5 grid_gdf["incident_count"] = grid_gdf.geometry.apply(
      6     lambda pt: all_gun_points.geometry.distance(pt).lt(750).sum()
      7 )

 ---------------------------------------- 18 frames ----------------------------------------
lib.pyx in pandas._libs.lib.map_infer()

/usr/lib/python3.11/inspect.py in getmodule(object, _filename)
    986     # Copy sys.modules in order to cope with changes while iterating
    987     for modname, module in sys.modules.copy().items():
--> 988         if ismodule(module) and hasattr(module, '__file__'):
    989             f = module.__file__
    990             if f == _filesbymodname.get(modname, None):

KeyboardInterrupt:
```

```
import statsmodels.api as sm
import statsmodels.formula.api as smf

# Drop missing values
model_data = grid_gdf.dropna(subset=["incident_count", "dist_to_gang", "dist_to_school", "ward"])

# Fit the IPPP-like model
model = smf.glm(
    formula="incident_count ~ dist_to_school + dist_to_gang + C(ward)",
    data=model_data,
    family=sm.families.Poisson()
) fit()

print(model.summary())
```

```
  File "<ipython-input-26-75f6467c06f9>", line 12
    ) fit()
      ^
SyntaxError: invalid syntax
```

```
grid_gdf["incident_count"].value_counts()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3804         try:
-> 3805             return self._engine.get_loc(casted_key)
   3806         except KeyError as err:

index.pyx in pandas._libs.index.IndexEngine.get_loc()

index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'incident_count'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
[... 3 frames]
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3810             ):
   3811                 raise InvalidIndexError(key)
-> 3812             raise KeyError(key) from err
   3813         except TypeError:
   3814             # If we have a listlike key, _check_indexing_error will raise

KeyError: 'incident_count'
```

```
grid_gdf = grid_gdf.to_crs("EPSG:26971")
all_schools = all_schools.to_crs("EPSG:26971")
gang_boundaries = gang_boundaries.to_crs("EPSG:26971")

grid_gdf["dist_to_school"] = grid_gdf.geometry.apply(lambda point: all_schools.geometry.distance(point).min())
grid_gdf["dist_to_gang"] = grid_gdf.geometry.apply(lambda point: gang_boundaries.geometry.distance(point).min())
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-28-dd253a48d3d3> in <cell line: 0>()
      4
      5 grid_gdf["dist_to_school"] = grid_gdf.geometry.apply(lambda point: all_schools.geometry.distance(point).min())
----> 6 grid_gdf["dist_to_gang"] = grid_gdf.geometry.apply(lambda point: gang_boundaries.geometry.distance(point).min())

[... 14 frames]
lib.pyx in pandas._libs.lib.map_infer()

/usr/local/lib/python3.11/dist-packages/shapely/measurement.py in distance(a, b, **kwargs)
     79
     80     """
---> 81     return lib.distance(a, b, **kwargs)
     82
     83

KeyboardInterrupt:
```

```
grid_gdf[["dist_to_school", "dist_to_gang"]].corr()
```

```
model = smf.glm(
    formula="incident_count ~ dist_to_gang + C(ward)",
    data=grid_gdf.dropna(),
    family=sm.families.Poisson()
).fit()
```

```
import statsmodels.api as sm
import statsmodels.formula.api as smf

# Drop missing values
model_data = grid_gdf.dropna(subset=["incident_count", "dist_to_gang", "dist_to_school", "ward"])

# Fit the IPPP-like model
model = smf.glm(
    formula="incident_count ~ dist_to_school + dist_to_gang + C(ward)",
    data=model_data,
    family=sm.families.Poisson()
).fit()

print(model.summary())
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-29-5e1caf54fe64> in <cell line: 0>()
      3
      4 # Drop missing values
----> 5 model_data = grid_gdf.dropna(subset=["incident_count", "dist_to_gang", "dist_to_school", "ward"])
      6
      7 # Fit the IPPP-like model

/usr/local/lib/python3.11/dist-packages/pandas/core/frame.py in dropna(self, axis, how, thresh, subset, inplace, ignore_index)
   6668             check = indices == -1
   6669             if check.any():
-> 6670                 raise KeyError(np.array(subset)[check].tolist())
   6671             agg_obj = self.take(indices, axis=agg_axis)
   6672

KeyError: ['incident_count', 'dist_to_gang', 'ward']
```

```
grid_gdf[["incident_count", "dist_to_school", "dist_to_gang"]].describe()
grid_gdf[["incident_count", "dist_to_school", "dist_to_gang"]].isna().sum()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-30-c67820d4bf98> in <cell line: 0>()
----> 1 grid_gdf[["incident_count", "dist_to_school", "dist_to_gang"]].describe()
      2 grid_gdf[["incident_count", "dist_to_school", "dist_to_gang"]].isna().sum()

                                  3 frames
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in _raise_if_missing(self, key, indexer, axis_name)
   6250
   6251             not_found = list(ensure_index(key)[missing_mask.nonzero()[0]].unique())
-> 6252             raise KeyError(f"{not_found} not in index")
   6253
   6254     @overload

KeyError: "['incident_count', 'dist_to_gang'] not in index"
```

```
grid_gdf["school_gang_index"] = (grid_gdf["dist_to_school"] + grid_gdf["dist_to_gang"]) / 2

model = smf.glm(
    formula="incident_count ~ school_gang_index + C(ward)",
    data=grid_gdf,
    family=sm.families.Poisson()
).fit()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3804         try:
-> 3805             return self._engine.get_loc(casted_key)
   3806         except KeyError as err:

index.pyx in pandas._libs.index.IndexEngine.get_loc()

index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'dist_to_gang'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
                                  3 frames
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3810             ):
   3811                 raise InvalidIndexError(key)
-> 3812             raise KeyError(key) from err
   3813         except TypeError:
   3814             # If we have a listlike key, _check_indexing_error will raise

KeyError: 'dist_to_gang'
```

```
from shapely.geometry import Point
import numpy as np

minx, miny, maxx, maxy = wards.total_bounds
x_coords = np.linspace(minx, maxx, 100)
y_coords = np.linspace(miny, maxy, 100)
grid_points = [Point(x, y) for x in x_coords for y in y_coords]
grid_gdf = gpd.GeoDataFrame(geometry=grid_points, crs=wards.crs)
grid_gdf = grid_gdf[grid_gdf.geometry.within(wards.unary_union)]
```

```
<ipython-input-50-2583640b8e86>:9: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  grid_gdf = grid_gdf[grid_gdf.geometry.within(wards.unary_union)]
```

```
grid_gdf["in_gang"] = grid_gdf.geometry.within(gang_boundaries.unary_union).astype(int)
```

```
<ipython-input-32-77fb441528da>:1: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  grid_gdf["in_gang"] = grid_gdf.geometry.within(gang_boundaries.unary_union).astype(int)
```

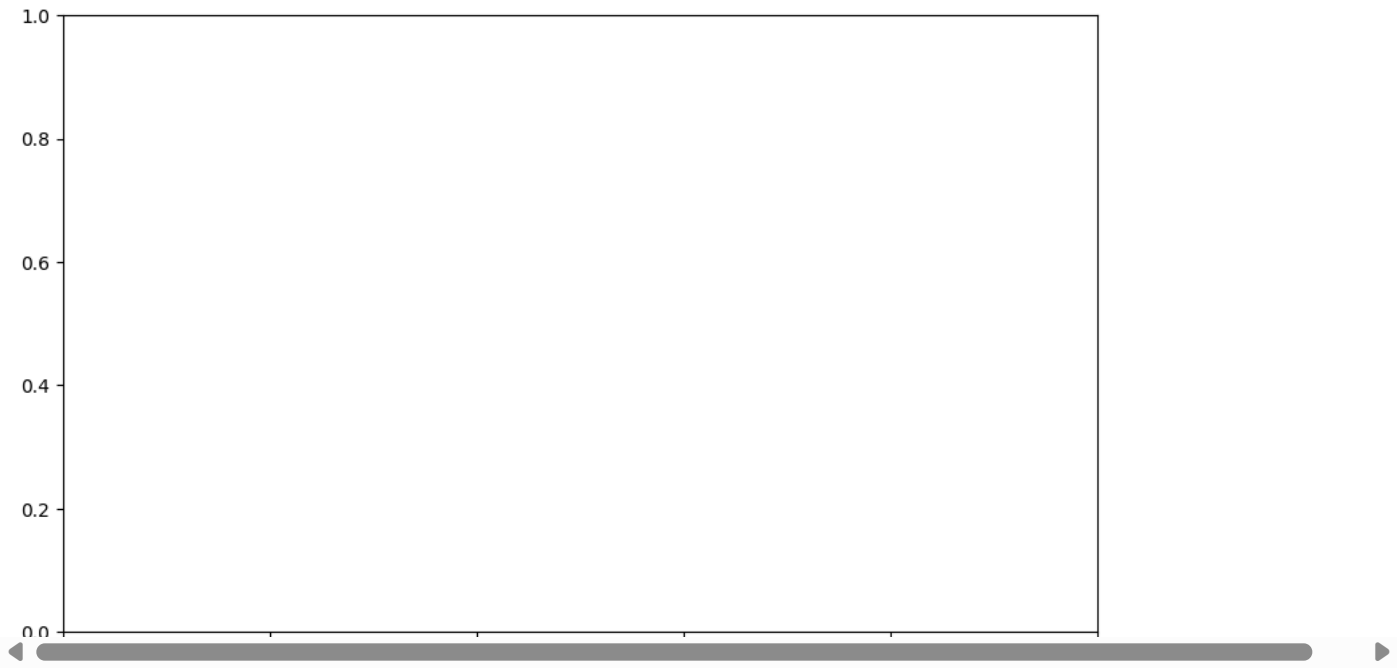
```
grid_gdf["incident_count"] = grid_gdf.geometry.apply(
    lambda pt: all_gun_points.geometry.distance(pt).lt(50).sum()
)
grid_gdf
print(grid_gdf.columns)
```

```
<ipython-input-33-8351f785fe3d>:2: UserWarning: Geometry is in a geographic CRS. Results from 'distance' are likely incorrect. Use 'GeoS

  lambda pt: all_gun_points.geometry.distance(pt).lt(50).sum()
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-33-8351f785fe3d> in <cell line: 0>()
----> 1 grid_gdf["incident_count"] = grid_gdf.geometry.apply(
      2     lambda pt: all_gun_points.geometry.distance(pt).lt(50).sum()
      3 )
      4 grid_gdf
      5 print(grid_gdf.columns)

                                 14 frames
lib.pyx in pandas._libs.lib.map_infer()

/usr/local/lib/python3.11/dist-packages/shapely/measurement.py in distance(a, b, **kwargs)
     79
     80     """
---> 81     return lib.distance(a, b, **kwargs)
     82
     83

KeyboardInterrupt:
```

```
import statsmodels.api as sm
import statsmodels.formula.api as smf

model = smf.glm(
    formula="incident_count ~ in_gang",
    data=grid_gdf,
    family=sm.families.Poisson()
).fit()

print(model.summary())
```

```
import geopandas as gpd
import pandas as pd
import matplotlib.pyplot as plt

# Load and prepare the gun violence data
gun_df = pd.read_csv("chicago_shooting.csv").dropna(subset=["LONGITUDE", "LATITUDE"])
all_gun_points = gpd.GeoDataFrame(
    gun_df,
    geometry=gpd.points_from_xy(gun_df["LONGITUDE"], gun_df["LATITUDE"]),
    crs="EPSG:4326"
)

# Reproject to match gang boundaries
all_gun_points = all_gun_points.to_crs(gang_boundaries.crs)

# Check if each point is inside a gang boundary
all_gun_points["Inside_Gang"] = all_gun_points.geometry.within(gang_boundaries.unary_union)

# Split into inside and outside
inside_counts = all_gun_points[all_gun_points["Inside_Gang"] == True]
outside_counts = all_gun_points[all_gun_points["Inside_Gang"] == False]

# Plot histogram (same bin size for comparison)
plt.figure(figsize=(10, 6))
plt.hist([len(inside_counts), len(outside_counts)],
         bins=[0, 1, 2],
         label=["Inside Gang Territory", "Outside Gang Territory"],
         align='left',
         color=["red", "blue"])

plt.xticks([0, 1], ["Inside", "Outside"])
plt.ylabel("Number of Shootings")
plt.title("Shootings Inside vs Outside Gang Territory")
plt.legend()
plt.tight_layout()
plt.show()
```

```
<ipython-input-34-0bdbcacac905>:6: DtypeWarning: Columns (25,26,27) have mixed types. Specify dtype option on import or set low_memory=F
  gun_df = pd.read_csv("chicago_shooting.csv").dropna(subset=["LONGITUDE", "LATITUDE"])
<ipython-input-34-0bdbcacac905>:17: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  all_gun_points["Inside_Gang"] = all_gun_points.geometry.within(gang_boundaries.unary_union)
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-34-0bdbcacac905> in <cell line: 0>()
     23 # Plot histogram (same bin size for comparison)
     24 plt.figure(figsize=(10, 6))
---> 25 plt.hist([len(inside_counts), len(outside_counts)],
     26          bins=[0, 1, 2],
     27          label=["Inside Gang Territory", "Outside Gang Territory"],

                                 4 frames
/usr/local/lib/python3.11/dist-packages/matplotlib/axes/_axes.py in hist(self, x, bins, range, density, weights, cumulative, bottom,
histtype, align, orientation, rwidth, log, color, label, stacked, **kwargs)
   7047             colors = mcolors.to_rgba_array(color)
   7048             if len(colors) != nx:
-> 7049                 raise ValueError(f"The 'color' keyword argument must have one "
   7050                                  f"color per dataset, but {nx} datasets and "
   7051                                  f"{len(colors)} colors were provided")

ValueError: The 'color' keyword argument must have one color per dataset, but 1 datasets and 2 colors were provided
```




```
print(gun_df.columns)
```

gang_boundaries

	OBJECTID	GANG_NAME	SHAPE__Are	SHAPE__Len	geometry	year	Shape__Are	Shape__Len	FID	SHAPE_AREA	SHAPE_LEN
0	1.0	LATIN COUNTS	3.330409e+06	20109.113362	MULTIPOLYGON (((355484.203 572739.731, 355484....	2007	NaN	NaN	NaN	NaN	NaN
1	2.0	LATIN DRAGONS	1.738342e+06	13610.756236	MULTIPOLYGON (((367193.685 553951.515, 367234....	2007	NaN	NaN	NaN	NaN	NaN
2	3.0	LATIN EAGLES	3.177308e+05	2867.456277	POLYGON ((349524.328 583080.973, 349524.912 58...	2007	NaN	NaN	NaN	NaN	NaN
3	4.0	LATIN JIVERS	7.516058e+04	1641.799169	MULTIPOLYGON (((352834.707 581517.845, 352809....	2007	NaN	NaN	NaN	NaN	NaN
					MULTIPOLYGON						

```
years = list(range(2007, 2025))

for year in years:
    gang_this_year = gang_boundaries[gang_boundaries["year"] == year]

    if not gang_this_year.empty:
        # Generate treatment flag: 1 if grid point is within 300m of any gang zone
        buffer = gang_this_year.geometry.buffer(0).unary_union
        grid_gdf[f"treated_{year}"] = grid_gdf.geometry.within(buffer).astype(int)
```

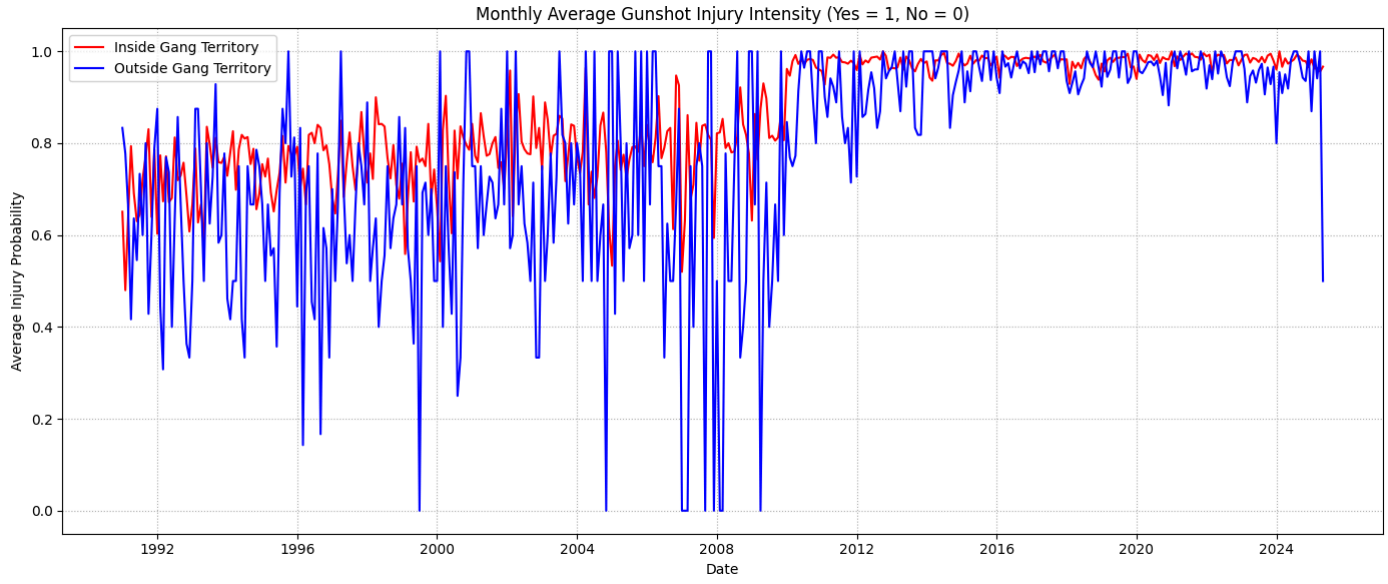
```
<ipython-input-35-002e01f955cb>:8: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  buffer = gang_this_year.geometry.buffer(0).unary_union
<ipython-input-35-002e01f955cb>:8: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  buffer = gang_this_year.geometry.buffer(0).unary_union
<ipython-input-35-002e01f955cb>:8: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  buffer = gang_this_year.geometry.buffer(0).unary_union
<ipython-input-35-002e01f955cb>:8: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  buffer = gang_this_year.geometry.buffer(0).unary_union
<ipython-input-35-002e01f955cb>:8: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  buffer = gang_this_year.geometry.buffer(0).unary_union
<ipython-input-35-002e01f955cb>:8: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  buffer = gang_this_year.geometry.buffer(0).unary_union
<ipython-input-35-002e01f955cb>:8: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  buffer = gang_this_year.geometry.buffer(0).unary_union
<ipython-input-35-002e01f955cb>:8: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  buffer = gang_this_year.geometry.buffer(0).unary_union
<ipython-input-35-002e01f955cb>:8: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  buffer = gang_this_year.geometry.buffer(0).unary_union
<ipython-input-35-002e01f955cb>:8: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  buffer = gang_this_year.geometry.buffer(0).unary_union
<ipython-input-35-002e01f955cb>:8: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  buffer = gang_this_year.geometry.buffer(0).unary_union
<ipython-input-35-002e01f955cb>:8: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  buffer = gang_this_year.geometry.buffer(0).unary_union
<ipython-input-35-002e01f955cb>:8: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  buffer = gang_this_year.geometry.buffer(0).unary_union
<ipython-input-35-002e01f955cb>:8: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  buffer = gang_this_year.geometry.buffer(0).unary_union
<ipython-input-35-002e01f955cb>:8: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  buffer = gang_this_year.geometry.buffer(0).unary_union
<ipython-input-35-002e01f955cb>:8: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  buffer = gang_this_year.geometry.buffer(0).unary_union
<ipython-input-35-002e01f955cb>:8: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  buffer = gang_this_year.geometry.buffer(0).unary_union
```

```
grid_gdf["incident_count_2015"]  # for example
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3804         try:
-> 3805             return self._engine.get_loc(casted_key)
   3806         except KeyError as err:

index.pyx in pandas._libs.index.IndexEngine.get_loc()

index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'incident_count_2015'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
                              3 frames
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3810             ):
   3811                 raise InvalidIndexError(key)
-> 3812             raise KeyError(key) from err
   3813         except TypeError:
   3814             # If we have a listlike key, _check_indexing_error will raise

KeyError: 'incident_count_2015'
```

```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt

# --- Prepare gun violence data ---
gun_df = pd.read_csv("chicago_shooting.csv")
gun_df = gun_df.dropna(subset=["LONGITUDE", "LATITUDE", "DATE", "GUNSHOT_INJURY_I"])
gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
gun_df["GUNSHOT_INJURY_I"] = gun_df["GUNSHOT_INJURY_I"].str.strip().str.upper().map({"YES": 1, "NO": 0})

# --- Convert to GeoDataFrame ---
all_gun_points = gpd.GeoDataFrame(
    gun_df,
    geometry=gpd.points_from_xy(gun_df["LONGITUDE"], gun_df["LATITUDE"]),
    crs="EPSG:4326"
)
all_gun_points = all_gun_points.to_crs(gang_boundaries.crs)

# --- Classify as inside or outside gang boundaries ---
all_gun_points["Inside_Gang"] = all_gun_points.geometry.within(gang_boundaries.unary_union)

# --- Group by date and gang territory status ---
summary = all_gun_points.groupby([all_gun_points["DATE"].dt.to_period("M"), "Inside_Gang"])["GUNSHOT_INJURY_I"].mean().unstack(fill_value=0)
summary.index = summary.index.to_timestamp()

# --- Plot ---
plt.figure(figsize=(14, 6))
plt.plot(summary.index, summary[True], label="Inside Gang Territory", color="red")
plt.plot(summary.index, summary[False], label="Outside Gang Territory", color="blue")
plt.title("Monthly Average Gunshot Injury Intensity (Yes = 1, No = 0)")
plt.xlabel("Date")
plt.ylabel("Average Injury Probability")
plt.legend()
plt.grid(True, linestyle=":")
plt.tight_layout()
plt.show()
```

```
<ipython-input-37-747bd70114b0>:6: DtypeWarning: Columns (25,26,27) have mixed types. Specify dtype option on import or set low_memory=F
  gun_df = pd.read_csv("chicago_shooting.csv")
<ipython-input-37-747bd70114b0>:8: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `da
  gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
<ipython-input-37-747bd70114b0>:20: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  all_gun_points["Inside_Gang"] = all_gun_points.geometry.within(gang_boundaries.unary_union)
```



```
# --- Count total incidents by gang territory status ---
counts = all_gun_points["Inside_Gang"].value_counts()
print("Incident Counts:")
print(counts.rename({True: "Inside Gang Territory", False: "Outside Gang Territory"}))
```

```
Incident Counts:
Inside_Gang
Inside Gang Territory     54545
Outside Gang Territory     6893
Name: count, dtype: int64
```

```
# --- Classify as inside or outside gang boundaries ---
all_gun_points["Inside_Gang"]
```

	Inside_Gang
0	True
1	True
2	True
3	True
4	True
...	...
61437	False
61438	False
61439	True
61440	False
61441	True

61438 rows × 1 columns

dtype: bool

```
gun_df.columns
```

```
Index(['CASE_NUMBER', 'DATE', 'BLOCK', 'VICTIMIZATION_PRIMARY',
       'INCIDENT_PRIMARY', 'GUNSHOT_INJURY_I', 'UNIQUE_ID', 'ZIP_CODE', 'WARD',
       'COMMUNITY_AREA', 'STREET_OUTREACH_ORGANIZATION', 'AREA', 'DISTRICT',
       'BEAT', 'AGE', 'SEX', 'RACE', 'VICTIMIZATION_FBI_CD', 'INCIDENT_FBI_CD',
       'VICTIMIZATION_FBI_DESCR', 'INCIDENT_FBI_DESCR',
       'VICTIMIZATION_IUCR_CD', 'INCIDENT_IUCR_CD',
       'VICTIMIZATION_IUCR_SECONDARY', 'INCIDENT_IUCR_SECONDARY',
       'HOMICIDE_VICTIM_FIRST_NAME', 'HOMICIDE_VICTIM_MI',
       'HOMICIDE_VICTIM_LAST_NAME', 'MONTH', 'DAY_OF_WEEK', 'HOUR',
       'LOCATION_DESCRIPTION', 'STATE_HOUSE_DISTRICT', 'STATE_SENATE_DISTRICT',
       'UPDATED', 'LATITUDE', 'LONGITUDE', 'LOCATION'],
      dtype='object')
```

```
gun_df['INCIDENT_PRIMARY'].unique()
```

```
array(['HOMICIDE', 'BATTERY', 'ROBBERY', 'NON-FATAL',
       'CRIMINAL SEXUAL ASSAULT'], dtype=object)
```

```
import pandas as pd

# List of crimes ordered by severity
crime_ranking = {
    'SECOND DEGREE MURDER': 1,
    'AGGRAVATED POLICE OFFICER - HANDGUN': 2,
    'AGGRAVATED POLICE OFFICER - OTHER DANGEROUS WEAPON': 3,
    'AGGRAVATED DOMESTIC BATTERY - HANDGUN': 4,
    'AGGRAVATED PROTECTED EMPLOYEE - HANDGUN': 5,
    'AGGRAVATED PROTECTED EMPLOYEE - OTHER FIREARM': 6,
    'AGGRAVATED VEHICULAR HIJACKING': 7,
    'AGGRAVATED - OTHER FIREARM': 8,
    'AGGRAVATED - OTHER DANGEROUS WEAPON': 9,
    'ARMED - OTHER FIREARM': 10,
```

```
    'ARMED - OTHER DANGEROUS WEAPON': 11,
    'ATTEMPT AGGRAVATED': 12,
    'ATTEMPT ARMED - OTHER FIREARM': 13,
    'ATTEMPT ARMED - OTHER DANGEROUS WEAPON': 14,
    'AGGRAVATED - KNIFE / CUTTING INSTRUMENT': 15,
    'AGGRAVATED DOMESTIC BATTERY - OTHER FIREARM': 16,
    'AGGRAVATED DOMESTIC BATTERY - OTHER DANGEROUS WEAPON': 17,
    'VEHICULAR HIJACKING': 18,
    'RECKLESS CONDUCT': 19,
    'SIMPLE': 20,
    'DOMESTIC BATTERY SIMPLE': 21
}

# Create a dataframe with the crimes
crimes = ['AGGRAVATED DOMESTIC BATTERY - HANDGUN', 'AGGRAVATED POLICE OFFICER - OTHER DANGEROUS WEAPON',
          'AGGRAVATED VEHICULAR HIJACKING', 'ATTEMPT AGGRAVATED', 'AGGRAVATED - OTHER FIREARM',
          'ATTEMPT ARMED - OTHER FIREARM', 'AGGRAVATED POLICE OFFICER - HANDGUN',
          'AGGRAVATED POLICE OFFICER - OTHER FIREARM', 'AGGRAVATED', 'SECOND DEGREE MURDER',
          'ARMED - OTHER FIREARM', 'AGGRAVATED PROTECTED EMPLOYEE - OTHER FIREARM', 'SIMPLE',
          'AGGRAVATED DOMESTIC BATTERY - OTHER FIREARM', 'AGGRAVATED PROTECTED EMPLOYEE - HANDGUN',
          'AGGRAVATED - KNIFE / CUTTING INSTRUMENT', 'AGGRAVATED DOMESTIC BATTERY - OTHER DANGEROUS WEAPON',
          'VEHICULAR HIJACKING', 'ATTEMPT ARMED - OTHER DANGEROUS WEAPON', 'AGGRAVATED - OTHER DANGEROUS WEAPON',
          'ARMED - OTHER DANGEROUS WEAPON', 'RECKLESS CONDUCT', 'DOMESTIC BATTERY SIMPLE']

# Map the crimes to their ranks
crime_ranking_df = pd.DataFrame(crimes, columns=['Crime'])
crime_ranking_df['Rank'] = crime_ranking_df['Crime'].map(crime_ranking)

# Display the dataframe
print(crime_ranking_df)
```

```
                                           Crime  Rank
0              AGGRAVATED DOMESTIC BATTERY - HANDGUN   4.0
1   AGGRAVATED POLICE OFFICER - OTHER DANGEROUS WE...   3.0
2                     AGGRAVATED VEHICULAR HIJACKING   7.0
3                                ATTEMPT AGGRAVATED  12.0
4                        AGGRAVATED - OTHER FIREARM   8.0
5                     ATTEMPT ARMED - OTHER FIREARM  13.0
6                AGGRAVATED POLICE OFFICER - HANDGUN   2.0
7           AGGRAVATED POLICE OFFICER - OTHER FIREARM   NaN
8                                        AGGRAVATED   NaN
9                              SECOND DEGREE MURDER   1.0
10                            ARMED - OTHER FIREARM  10.0
11     AGGRAVATED PROTECTED EMPLOYEE - OTHER FIREARM   6.0
12                                          SIMPLE  20.0
13       AGGRAVATED DOMESTIC BATTERY - OTHER FIREARM  16.0
14           AGGRAVATED PROTECTED EMPLOYEE - HANDGUN   5.0
15           AGGRAVATED - KNIFE / CUTTING INSTRUMENT  15.0
16  AGGRAVATED DOMESTIC BATTERY - OTHER DANGEROUS ...  17.0
17                               VEHICULAR HIJACKING  18.0
18            ATTEMPT ARMED - OTHER DANGEROUS WEAPON  14.0
19               AGGRAVATED - OTHER DANGEROUS WEAPON   9.0
20                    ARMED - OTHER DANGEROUS WEAPON  11.0
21                                 RECKLESS CONDUCT  19.0
22                           DOMESTIC BATTERY SIMPLE  21.0
```

```
# --- Group by date and gang territory status ---
summary = all_gun_points.groupby([all_gun_points["DATE"].dt.to_period("M"), "Inside_Gang"])["GUNSHOT_INJURY_I"].mean().unstack(fill_value=0)
summary.index = summary.index.to_timestamp()
summary
```

Inside_Gang	0	1
DATE		
1991-01-01	0.833333	0.650794
1991-02-01	0.777778	0.480000
1991-03-01	0.666667	0.633333
1991-04-01	0.416667	0.793651
1991-05-01	0.636364	0.690909
...	...	...
2025-01-01	0.869565	0.982609
2025-02-01	1.000000	0.962500
2025-03-01	0.941176	0.963303
2025-04-01	1.000000	0.956140
2025-05-01	0.500000	0.966667

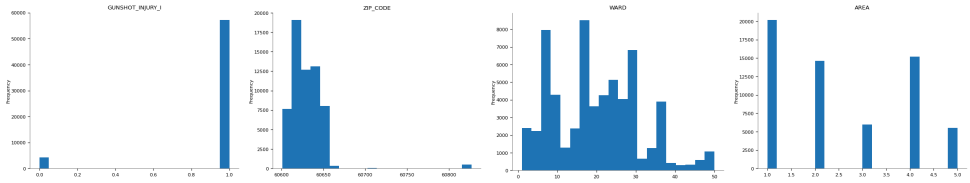
413 rows × 2 columns

```
gun_df
```

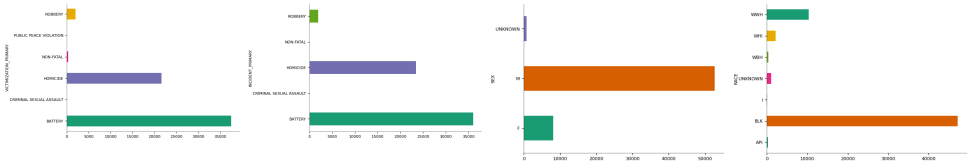
	CASE_NUMBER	DATE	BLOCK	VICTIMIZATION_PRIMARY	INCIDENT_PRIMARY	GUNSHOT_INJURY_I	UNIQUE_ID	ZIP_CODE	WARD	COMMU
0	JF167335	2022-03-08 15:27:00	6000 N KENMORE AVE	HOMICIDE	HOMICIDE	0	HOM-JF167335-#1	60660.0	48.0	ED
1	JG148375	2023-02-11 02:30:00	8400 S WABASH AVE	HOMICIDE	HOMICIDE	1	HOM-JG148375-#1	60619.0	6.0	
2	JD438266	2020-11-21 22:15:00	7900 S BRANDON AVE	HOMICIDE	HOMICIDE	1	HOM-JD438266-#1	60617.0	7.0	
3	JH317789	2024-06-23 08:11:00	12300 S HALSTED ST	HOMICIDE	HOMICIDE	1	HOM-JH317789-#1	60628.0	9.0	
4	JH317789	2024-06-23 08:11:00	12300 S HALSTED ST	HOMICIDE	HOMICIDE	1	HOM-JH317789-#2	60628.0	9.0	
...	...	...	...	...	...	...	...	...	...	
61437	JJ130497	2025-01-28 18:36:00	5300 S WABASH AVE	HOMICIDE	HOMICIDE	1	HOM-JJ130497-#1	60615.0	3.0	WAS
61438	JJ130497	2025-01-28 18:36:00	5300 S WABASH AVE	HOMICIDE	HOMICIDE	1	HOM-JJ130497-#2	60615.0	3.0	WAS
61439	JJ129588	2025-01-28 04:50:00	7700 S OGLESBY AVE	BATTERY	HOMICIDE	1	SHOOT-JJ129588-#3	60649.0	7.0	SOU
61440	JJ129772	2025-01-28 09:11:00	2000 N NEWCASTLE AVE	HOMICIDE	HOMICIDE	1	HOM-JJ129772-#1	60707.0	29.0	
61441	JJ128407	2025-01-27 05:16:00	1700 W GARFIELD BLVD	BATTERY	BATTERY	1	SHOOT-JJ128407-#1	60609.0	16.0	

61438 rows × 38 columns

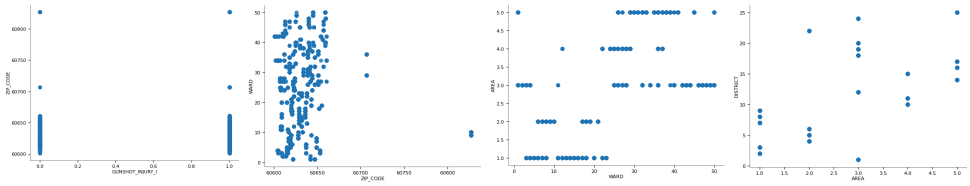
Distributions



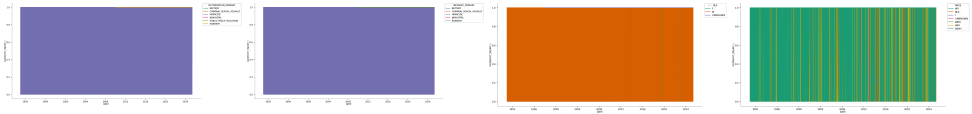
Categorical distributions



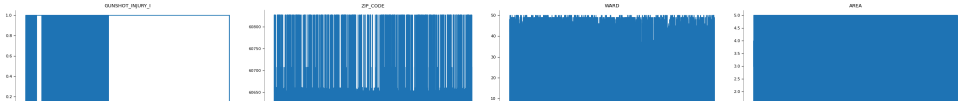
2-d distributions

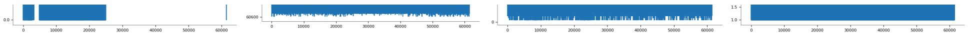


Time series

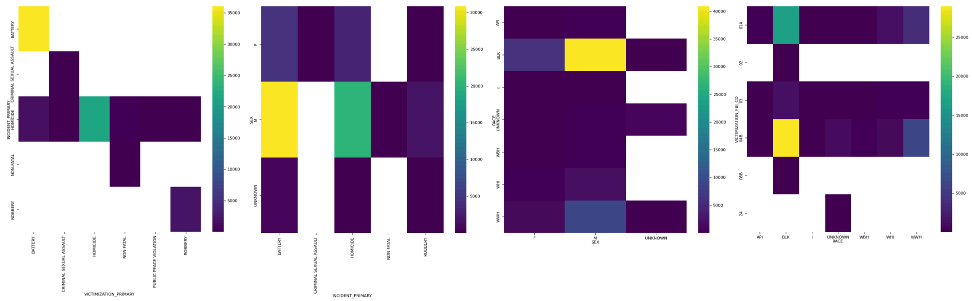


Values



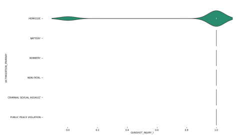


2-d categorical distributions

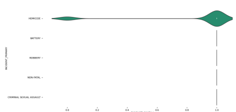


Faceted distributions

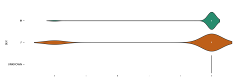
```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `leg
```



```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `leg
```



```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `leg
```



```
<string>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `leg
```

```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
import statsmodels.api as sm

# --- Load and prepare data ---
gun_df = pd.read_csv("chicago_shooting.csv")
gun_df = gun_df.dropna(subset=["LONGITUDE", "LATITUDE", "DATE", "GUNSHOT_INJURY_I"])
gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
gun_df["GUNSHOT_INJURY_I"] = gun_df["GUNSHOT_INJURY_I"].str.strip().str.upper().map({"YES": 1, "NO": 0})

# --- Convert to GeoDataFrame ---
all_gun_points = gpd.GeoDataFrame(
    gun_df,
    geometry=gpd.points_from_xy(gun_df["LONGITUDE"], gun_df["LATITUDE"]),
    crs="EPSG:4326"
)
all_gun_points = all_gun_points.to_crs(gang_boundaries.crs)

# --- Classify whether within gang territory ---
all_gun_points["Inside_Gang"] = all_gun_points.geometry.within(gang_boundaries.unary_union).astype(int)

# --- Optional: extract time features ---
all_gun_points["Year"] = all_gun_points["DATE"].dt.year
all_gun_points["Month"] = all_gun_points["DATE"].dt.month

# --- Create logistic regression model ---
X = all_gun_points[["Inside_Gang", "Year", "Month"]]
X = sm.add_constant(X)  # add intercept
y = all_gun_points["GUNSHOT_INJURY_I"]

model = sm.Logit(y, X)
result = model.fit()

# --- Output results ---
print(result.summary())
```

```
<ipython-input-42-92897d856fb5>:7: DtypeWarning: Columns (25,26,27) have mixed types. Specify dtype option on import or set low_memory=F
  gun_df = pd.read_csv("chicago_shooting.csv")
<ipython-input-42-92897d856fb5>:9: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `da
  gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
<ipython-input-42-92897d856fb5>:21: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  all_gun_points["Inside_Gang"] = all_gun_points.geometry.within(gang_boundaries.unary_union).astype(int)
Optimization terminated successfully.
         Current function value: 0.205431
         Iterations 8
                           Logit Regression Results
==============================================================================
Dep. Variable:       GUNSHOT_INJURY_I   No. Observations:                61438
Model:                          Logit   Df Residuals:                    61434
Method:                           MLE   Df Model:                            3
Date:                Sat, 10 May 2025   Pseudo R-squ.:                 0.1866
Time:                        18:32:02   Log-Likelihood:                -12621.
converged:                       True   LL-Null:                       -15517.
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const       -234.7803      3.346    -70.171      0.000    -241.338    -228.223
Inside_Gang    0.7050      0.045     15.647      0.000       0.617       0.793
Year           0.1178      0.002     70.583      0.000       0.115       0.121
Month          0.0196      0.005      3.701      0.000       0.009       0.030
==============================================================================
```

```
import pandas as pd
import geopandas as gpd
import statsmodels.api as sm

# --- Load and clean data ---
gun_df = pd.read_csv("chicago_shooting.csv")
gun_df = gun_df.dropna(subset=["LONGITUDE", "LATITUDE", "DATE", "GUNSHOT_INJURY_I"])
gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
gun_df["GUNSHOT_INJURY_I"] = gun_df["GUNSHOT_INJURY_I"].str.strip().str.upper().map({"YES": 1, "NO": 0})

# --- Time-based features from DATE ---
```

```
gun_df["Year"] = gun_df["DATE"].dt.year
gun_df["Month"] = gun_df["DATE"].dt.month
gun_df["DayOfWeek"] = gun_df["DATE"].dt.dayofweek  # optional

# --- Convert to GeoDataFrame ---
gun_gdf = gpd.GeoDataFrame(
    gun_df,
    geometry=gpd.points_from_xy(gun_df["LONGITUDE"], gun_df["LATITUDE"]),
    crs="EPSG:4326"
)



# --- Determine if incident was inside a gang territory ---
gun_gdf["Inside_Gang"] = gun_gdf.geometry.within(gang_boundaries.unary_union).astype(int)

# --- Define independent variables (X) and dependent variable (y) ---
X = gun_gdf[["Inside_Gang", "Year", "Month"]]  # Add "DayOfWeek" if you like
X = sm.add_constant(X)  # adds intercept term
y = gun_gdf["GUNSHOT_INJURY_I"]

# --- Fit logistic regression model ---
logit_model = sm.Logit(y, X).fit()
print(logit_model.summary())
```

```
<ipython-input-43-2b2baf2143f8>:6: DtypeWarning: Columns (25,26,27) have mixed types. Specify dtype option on import or set low_memory=F
  gun_df = pd.read_csv("chicago_shooting.csv")
<ipython-input-43-2b2baf2143f8>:8: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `da
  gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
<ipython-input-43-2b2baf2143f8>:26: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_gdf["Inside_Gang"] = gun_gdf.geometry.within(gang_boundaries.unary_union).astype(int)
Optimization terminated successfully.
         Current function value: 0.207265
         Iterations 8
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
<ipython-input-43-2b2baf2143f8> in <cell line: 0>()
     32
     33 # --- Fit logistic regression model ---
---> 34 logit_model = sm.Logit(y, X).fit()
     35 print(logit_model.summary())

                                 4 frames
/usr/local/lib/python3.11/dist-packages/numpy/linalg/_linalg.py in _raise_linalgerror_singular(err, flag)
    102
    103 def _raise_linalgerror_singular(err, flag):
--> 104     raise LinAlgError("Singular matrix")
    105
    106 def _raise_linalgerror_nonposdef(err, flag):

LinAlgError: Singular matrix
```

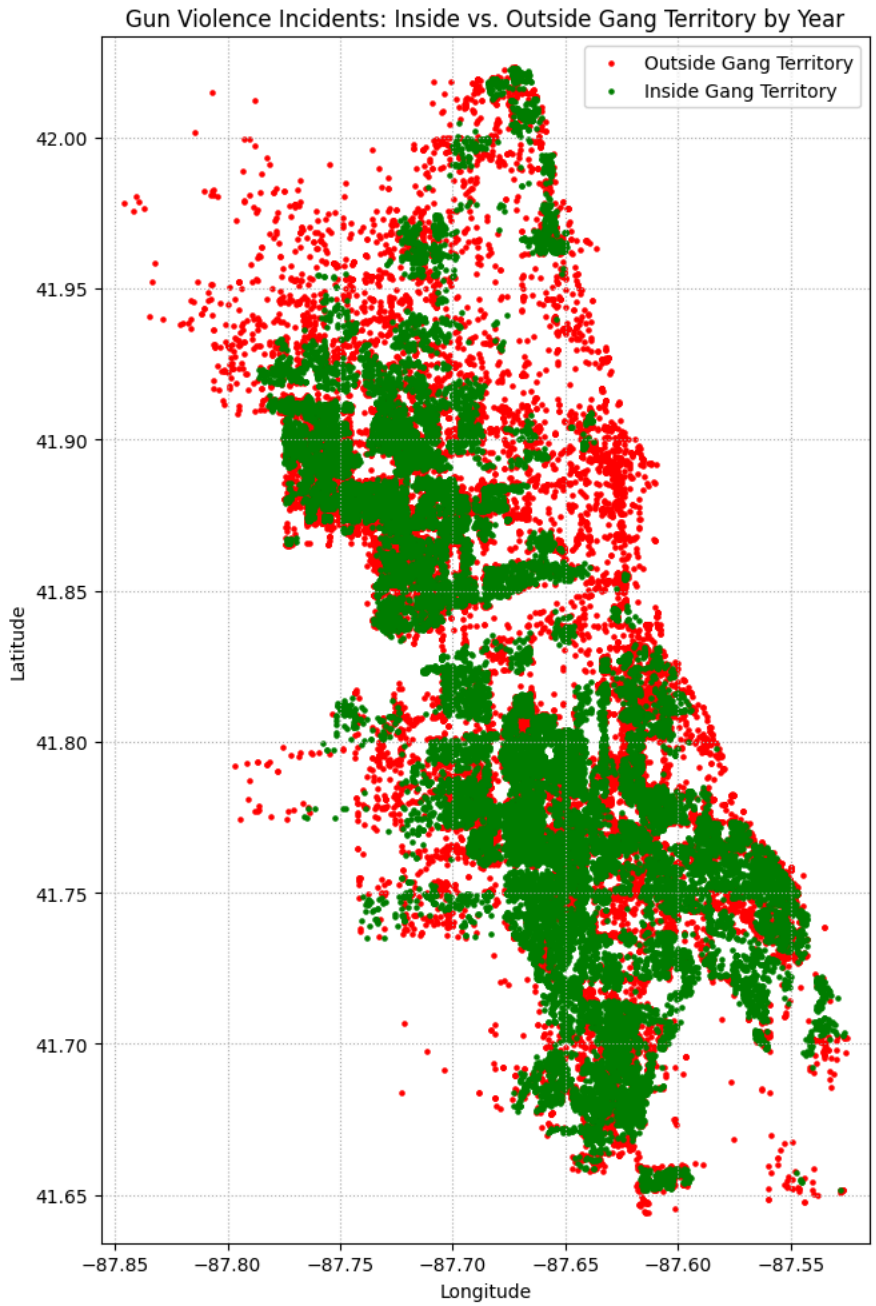
```
import pandas as pd
import geopandas as gpd
import os

# Load and clean gun violence data
gun_df = pd.read_csv("chicago_shooting.csv")
gun_df = gun_df.dropna(subset=["LONGITUDE", "LATITUDE", "DATE", "GUNSHOT_INJURY_I"])
gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
gun_df["GUNSHOT_INJURY_I"] = gun_df["GUNSHOT_INJURY_I"].str.strip().str.upper().map({"YES": 1, "NO": 0})
gun_df["Year"] = gun_df["DATE"].dt.year

# Convert to GeoDataFrame
gun_gdf = gpd.GeoDataFrame(
    gun_df,
    geometry=gpd.points_from_xy(gun_df["LONGITUDE"], gun_df["LATITUDE"]),
    crs="EPSG:4326"
)

# Container for matched data
matched_dfs = []

# Loop over all relevant years
for year in range(2007, 2025):
    gun_year = gun_gdf[gun_gdf["Year"] == year].copy()
```

```
        shapefile_path = f"{year}_Gang_Boundaries.shp"

        if os.path.exists(shapefile_path) and not gun_year.empty:
            try:
                gang_gdf = gpd.read_file(shapefile_path)
                gang_gdf = gang_gdf.to_crs(gun_year.crs)

                # Spatial join: attach gang info
                joined = gpd.sjoin(gun_year, gang_gdf, predicate="within", how="left")
                joined["Inside_Gang"] = ~joined["index_right"].isna()
                joined["Gang_Year"] = year
                matched_dfs.append(joined)
                print(f"✅ Year {year}: matched {len(joined)} incidents.")
            except Exception as e:
                print(f"⚠️ Error processing year {year}: {e}")
        else:
            print(f"⚠️ Missing data or no shootings for year {year}")

# Combine all matched years
final_df = pd.concat(matched_dfs, ignore_index=True)
final_df.to_csv("shootings_with_gang_matches.csv", index=False)
print("✅ Output saved to 'shootings_with_gang_matches.csv'")
```

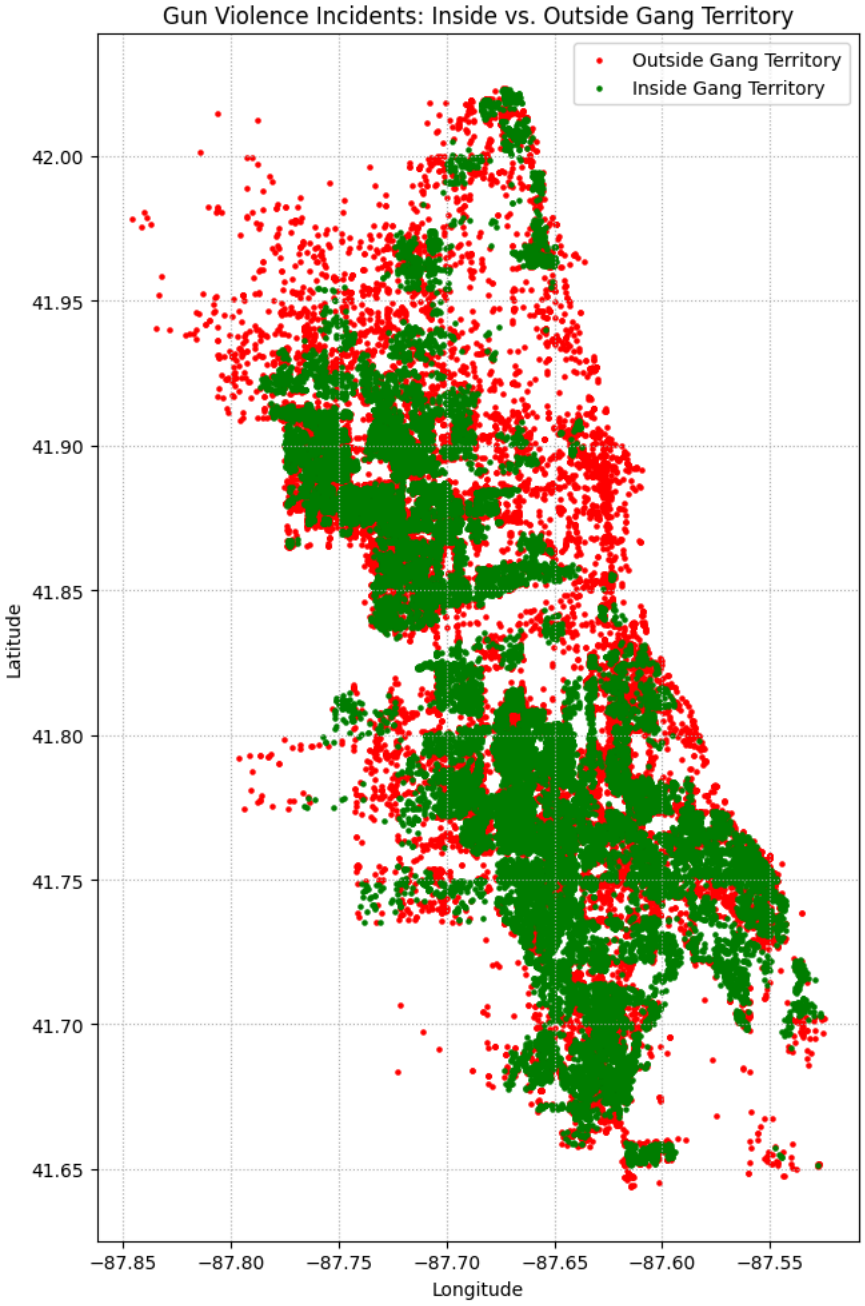
```
<ipython-input-44-ff7045ba753a>:6: DtypeWarning: Columns (25,26,27) have mixed types. Specify dtype option on import or set low_memory=F
  gun_df = pd.read_csv("chicago_shooting.csv")
<ipython-input-44-ff7045ba753a>:8: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `da
  gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
✅ Year 2007: matched 448 incidents.
✅ Year 2008: matched 518 incidents.
✅ Year 2009: matched 461 incidents.
✅ Year 2010: matched 2884 incidents.
✅ Year 2011: matched 2740 incidents.
✅ Year 2012: matched 3037 incidents.
⚠️ Missing data or no shootings for year 2013
✅ Year 2014: matched 2649 incidents.
✅ Year 2015: matched 3021 incidents.
✅ Year 2016: matched 4414 incidents.
✅ Year 2017: matched 3460 incidents.
✅ Year 2018: matched 3004 incidents.
✅ Year 2019: matched 2710 incidents.
✅ Year 2020: matched 4206 incidents.
✅ Year 2021: matched 4521 incidents.
✅ Year 2022: matched 3595 incidents.
✅ Year 2023: matched 3030 incidents.
✅ Year 2024: matched 2911 incidents.
✅ Output saved to 'shootings_with_gang_matches.csv'
```

```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# Load the processed file from the previous step
df = pd.read_csv("shootings_with_gang_matches.csv")
gdf = gpd.GeoDataFrame(
    df, geometry=gpd.points_from_xy(df["LONGITUDE"], df["LATITUDE"]), crs="EPSG:4326"
)

# Create a new column to extract the year from the date column (assuming a 'DATE' column exists in the CSV)
gdf["YEAR"] = pd.to_datetime(gdf["DATE"]).dt.year

# Create the figure and axis for plotting
fig, ax = plt.subplots(figsize=(12, 10))

# Set up the base map (Outside Gang Territory)
base = gdf[gdf["Inside_Gang"] == False].plot(ax=ax, markersize=5, color="red", label="Outside Gang Territory")
gdf[gdf["Inside_Gang"] == True].plot(ax=base, markersize=5, color="green", label="Inside Gang Territory")

# Set the plot limits based on the data's bounds
ax.set_xlim(gdf.geometry.x.min() - 0.01, gdf.geometry.x.max() + 0.01)
ax.set_ylim(gdf.geometry.y.min() - 0.01, gdf.geometry.y.max() + 0.01)

# Set up title, labels, and legend
ax.set_title("Gun Violence Incidents: Inside vs. Outside Gang Territory by Year")
ax.set_xlabel("Longitude")
ax.set_ylabel("Latitude")
```

```
    ax.legend()
    ax.grid(True, linestyle=":")

    # Function to update the scatter plot for each year
    def update(year):
        ax.clear()  # Clear the previous frame
        base = gdf[gdf["Inside_Gang"] == False].plot(ax=ax, markersize=5, color="red", label="Outside Gang Territory")
        gdf[gdf["Inside_Gang"] == True].plot(ax=base, markersize=5, color="green", label="Inside Gang Territory")

        # Filter incidents for the given year
        year_data = gdf[gdf["YEAR"] == year]

        # Plot the points for this year
        year_data[year_data["Inside_Gang"] == True].plot(ax=ax, markersize=5, color="green", alpha=0.6)
        year_data[year_data["Inside_Gang"] == False].plot(ax=ax, markersize=5, color="red", alpha=0.6)

        # Update title for current year
        ax.set_title(f"Gun Violence Incidents: Inside vs. Outside Gang Territory ({year})")

        # Set the plot limits to remain constant
        ax.set_xlim(gdf.geometry.x.min() - 0.01, gdf.geometry.x.max() + 0.01)
        ax.set_ylim(gdf.geometry.y.min() - 0.01, gdf.geometry.y.max() + 0.01)

    # Create the animation
    years = sorted(gdf["YEAR"].unique())
    ani = FuncAnimation(fig, update, frames=years, repeat=False, interval=1000)

    plt.tight_layout()
    plt.show()
```

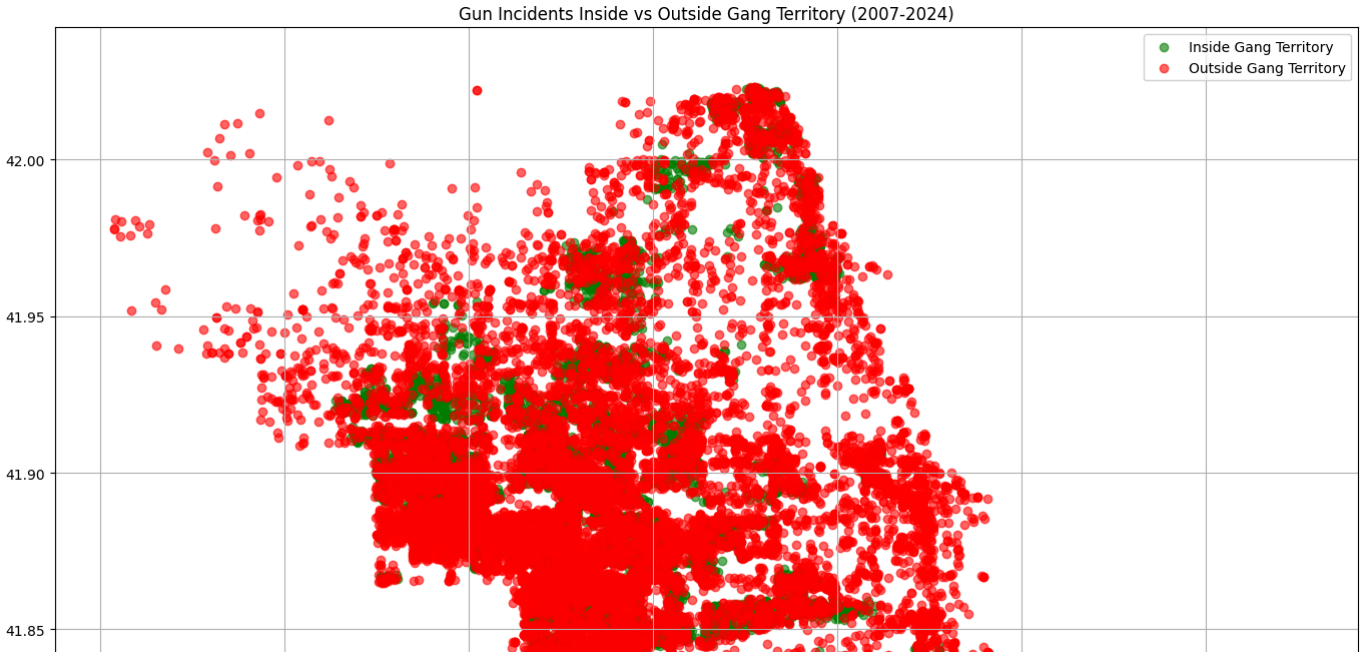

Gun Violence Incidents: Inside vs. Outside Gang Territory by Year
Outside Gang Territory
Inside Gang Territory
42.00
41.95
41.90
41.85
41.80
41.75
41.70
41.65
Latitude
−87.85
−87.80
−87.75
−87.70
−87.65
−87.60
−87.55
Longitude

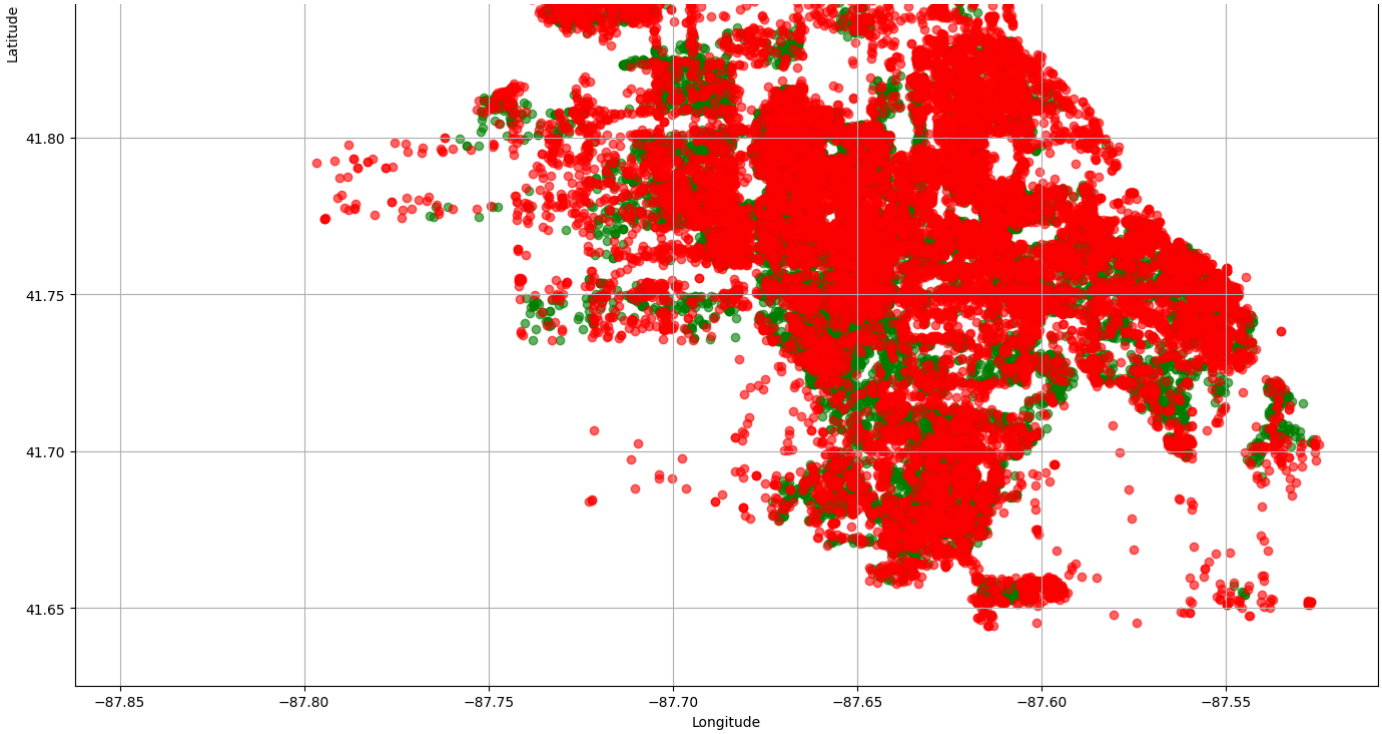
```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt

# Load the processed file from the previous step
df = pd.read_csv("shootings_with_gang_matches.csv")
gdf = gpd.GeoDataFrame(
    df, geometry=gpd.points_from_xy(df["LONGITUDE"], df["LATITUDE"]), crs="EPSG:4326"
)

# Plot
fig, ax = plt.subplots(figsize=(12, 10))
base = gdf[gdf["Inside_Gang"] == False].plot(ax=ax, markersize=5, color="red", label="Outside Gang Territory")
gdf[gdf["Inside_Gang"] == True].plot(ax=base, markersize=5, color="green", label="Inside Gang Territory")

plt.title("Gun Violence Incidents: Inside vs. Outside Gang Territory")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.legend()
plt.grid(True, linestyle=":")
plt.tight_layout()
plt.show()
```



```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt

# --- Prepare gun violence data ---
gun_df = pd.read_csv("chicago_shooting.csv")
gun_df = gun_df.dropna(subset=["LONGITUDE", "LATITUDE", "DATE", "GUNSHOT_INJURY_I"])
gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
gun_df["GUNSHOT_INJURY_I"] = gun_df["GUNSHOT_INJURY_I"].str.strip().str.upper().map({"YES": 1, "NO": 0})

# --- Convert to GeoDataFrame ---
all_gun_points = gpd.GeoDataFrame(
    gun_df,
    geometry=gpd.points_from_xy(gun_df["LONGITUDE"], gun_df["LATITUDE"]),
    crs="EPSG:4326"
)

# Initialize a new column to store if the incident is inside gang territory for the correct year
all_gun_points["Inside_Gang"] = False

# --- Loop through each year to apply the correct gang boundary ---
for year in range(2007, 2025):
    try:
        # Load the gang boundaries for the specific year
        gang_boundaries = gpd.read_file(f"{year}_Gang_Boundaries.shp").explode(index_parts=False)

        # Filter gun incidents for the current year
        gun_points_year = all_gun_points[all_gun_points["DATE"].dt.year == year]

        # Project gun incidents to the same CRS as the gang boundaries
        gun_points_year = gun_points_year.to_crs(gang_boundaries.crs)

        # Classify gun incidents as inside or outside gang territory
        gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)

        # Update the original dataframe with the inside/outside classification for this year
        all_gun_points.loc[all_gun_points["DATE"].dt.year == year, "Inside_Gang"] = gun_points_year["Inside_Gang"]

        print(f"Processed gang boundaries for {year}")
    except Exception as e:
        print(f"⚠️ Could not process year {year}: {e}")

# --- Plot the gun incidents categorized by gang territory status ---
plt.figure(figsize=(14, 14))
plt.scatter(all_gun_points[all_gun_points["Inside_Gang"] == True].geometry.x,
            all_gun_points[all_gun_points["Inside_Gang"] == True].geometry.y,
            color='green', label="Inside Gang Territory", alpha=0.6)
plt.scatter(all_gun_points[all_gun_points["Inside_Gang"] == False].geometry.x,
            all_gun_points[all_gun_points["Inside_Gang"] == False].geometry.y,
            color='red', label="Outside Gang Territory", alpha=0.6)

plt.title("Gun Incidents Inside vs Outside Gang Territory (2007-2024)")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

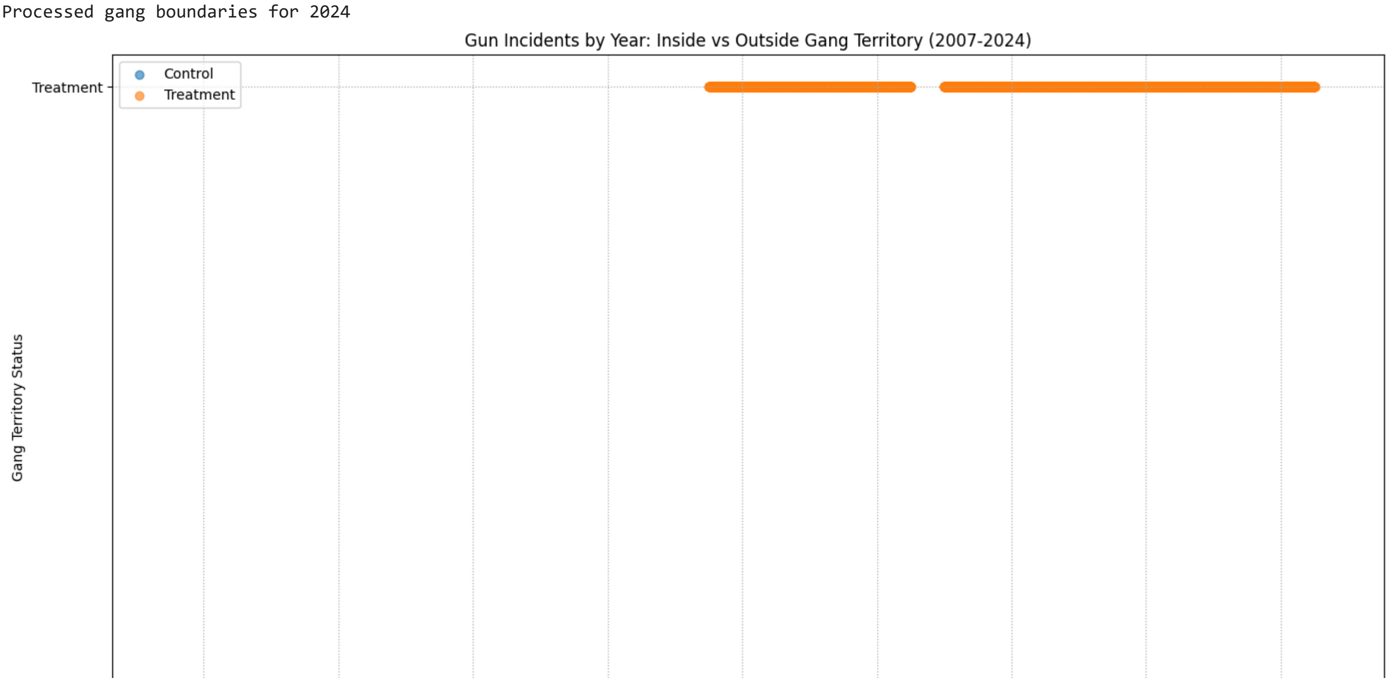
```
<ipython-input-47-a0a25c4af542>:6: DtypeWarning: Columns (25,26,27) have mixed types. Specify dtype option on import or set low_memory=F
  gun_df = pd.read_csv("chicago_shooting.csv")
<ipython-input-47-a0a25c4af542>:8: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `da
  gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
<ipython-input-47-a0a25c4af542>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2007
<ipython-input-47-a0a25c4af542>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2008
<ipython-input-47-a0a25c4af542>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2009
Processed gang boundaries for 2010
<ipython-input-47-a0a25c4af542>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-47-a0a25c4af542>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2011
Processed gang boundaries for 2012
⚠️ Could not process year 2013: 2013_Gang_Boundaries.shp: No such file or directory
<ipython-input-47-a0a25c4af542>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-47-a0a25c4af542>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2014
Processed gang boundaries for 2015
<ipython-input-47-a0a25c4af542>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-47-a0a25c4af542>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2016
Processed gang boundaries for 2017
<ipython-input-47-a0a25c4af542>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-47-a0a25c4af542>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2018
Processed gang boundaries for 2019
<ipython-input-47-a0a25c4af542>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-47-a0a25c4af542>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2020
Processed gang boundaries for 2021
<ipython-input-47-a0a25c4af542>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-47-a0a25c4af542>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2022
Processed gang boundaries for 2023
<ipython-input-47-a0a25c4af542>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-47-a0a25c4af542>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2024
```

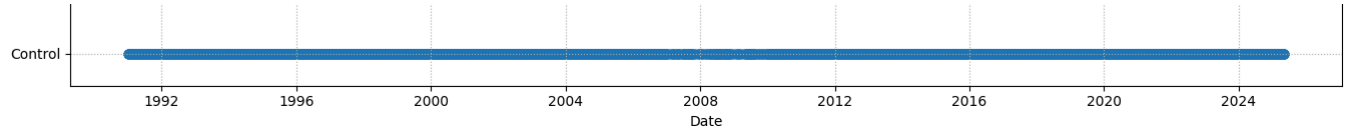



Latitude
41.80
41.75
41.70
41.65
−87.85
−87.80
−87.75
−87.70
−87.65
−87.60
−87.55
Longitude

```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt

# --- Prepare gun violence data ---
gun_df = pd.read_csv("chicago_shooting.csv")
gun_df = gun_df.dropna(subset=["LONGITUDE", "LATITUDE", "DATE", "GUNSHOT_INJURY_I"])
gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
gun_df["GUNSHOT_INJURY_I"] = gun_df["GUNSHOT_INJURY_I"].str.strip().str.upper().map({"YES": 1, "NO": 0})

# --- Convert to GeoDataFrame ---
all_gun_points = gpd.GeoDataFrame(
    gun_df,
    geometry=gpd.points_from_xy(gun_df["LONGITUDE"], gun_df["LATITUDE"]),
    crs="EPSG:4326"
)

# Initialize a new column to store if the incident is inside gang territory for the correct year
all_gun_points["Inside_Gang"] = False

# --- Loop through each year to apply the correct gang boundary ---
for year in range(2007, 2025):
    try:
        # Load the gang boundaries for the specific year
        gang_boundaries = gpd.read_file(f"{year}_Gang_Boundaries.shp").explode(index_parts=False)

        # Filter gun incidents for the current year
        gun_points_year = all_gun_points[all_gun_points["DATE"].dt.year == year]

        # Project gun incidents to the same CRS as the gang boundaries
        gun_points_year = gun_points_year.to_crs(gang_boundaries.crs)

        # Classify gun incidents as inside or outside gang territory
        gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)

        # Update the original dataframe with the inside/outside classification for this year
        all_gun_points.loc[all_gun_points["DATE"].dt.year == year, "Inside_Gang"] = gun_points_year["Inside_Gang"]

        print(f"Processed gang boundaries for {year}")
    except Exception as e:
        print(f"⚠️ Could not process year {year}: {e}")

# --- Map Inside_Gang to control and treatment ---
all_gun_points['Gang_Territory'] = all_gun_points['Inside_Gang'].map({True: 'Treatment', False: 'Control'})

# --- Plot Gun Incidents by Year, Categorized by Gang Territory Status ---
plt.figure(figsize=(14, 8))

# Plotting control and treatment on the y-axis with date on the x-axis
for territory in ['Control', 'Treatment']:
    subset = all_gun_points[all_gun_points['Gang_Territory'] == territory]
    plt.scatter(subset['DATE'], [territory] * len(subset), label=territory, alpha=0.6)

plt.title("Gun Incidents by Year: Inside vs Outside Gang Territory (2007-2024)")
plt.xlabel("Date")
plt.ylabel("Gang Territory Status")
plt.legend()
plt.grid(True, linestyle=":")
plt.tight_layout()
plt.show()
```

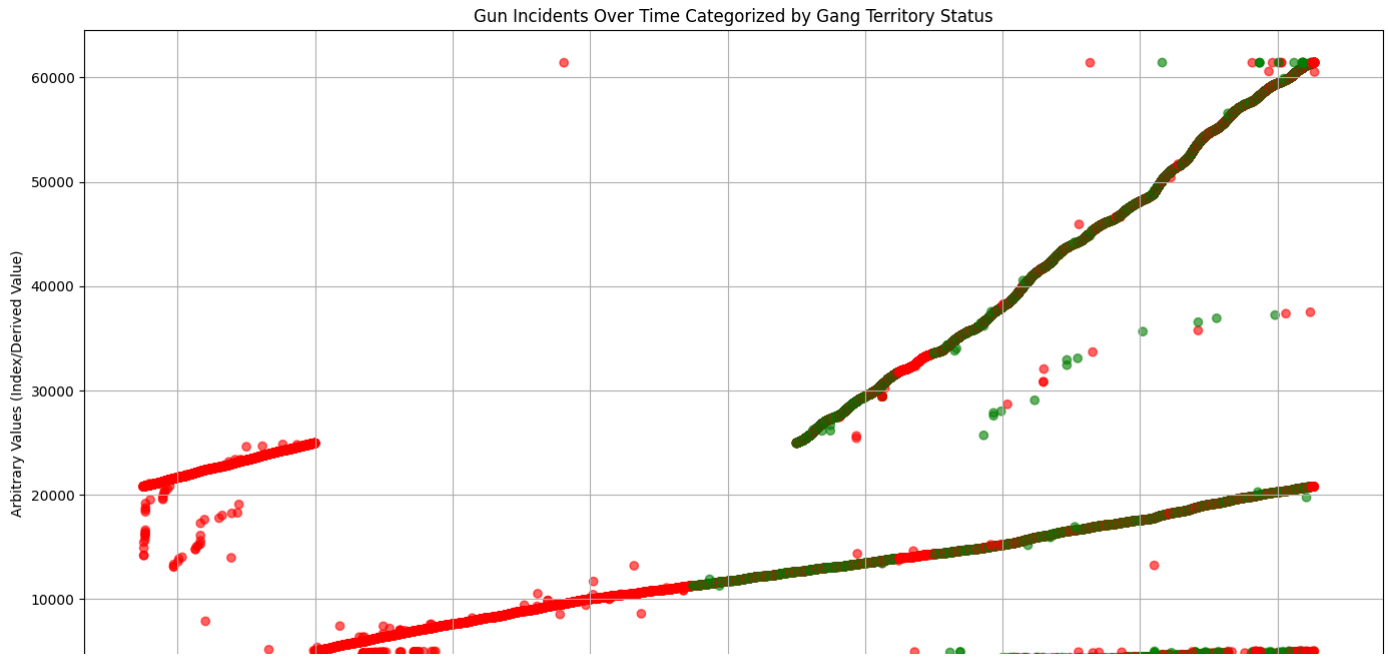
```
<ipython-input-48-99a5f69de673>:6: DtypeWarning: Columns (25,26,27) have mixed types. Specify dtype option on import or set low_memory=F
  gun_df = pd.read_csv("chicago_shooting.csv")
<ipython-input-48-99a5f69de673>:8: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `da
  gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
<ipython-input-48-99a5f69de673>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2007
<ipython-input-48-99a5f69de673>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2008
<ipython-input-48-99a5f69de673>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2009
Processed gang boundaries for 2010
<ipython-input-48-99a5f69de673>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-48-99a5f69de673>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2011
Processed gang boundaries for 2012
⚠️ Could not process year 2013: 2013_Gang_Boundaries.shp: No such file or directory
<ipython-input-48-99a5f69de673>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-48-99a5f69de673>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2014
Processed gang boundaries for 2015
<ipython-input-48-99a5f69de673>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-48-99a5f69de673>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-48-99a5f69de673>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2016
Processed gang boundaries for 2017
<ipython-input-48-99a5f69de673>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-48-99a5f69de673>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2018
Processed gang boundaries for 2019
<ipython-input-48-99a5f69de673>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-48-99a5f69de673>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2020
Processed gang boundaries for 2021
<ipython-input-48-99a5f69de673>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-48-99a5f69de673>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2022
Processed gang boundaries for 2023
<ipython-input-48-99a5f69de673>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2024
```

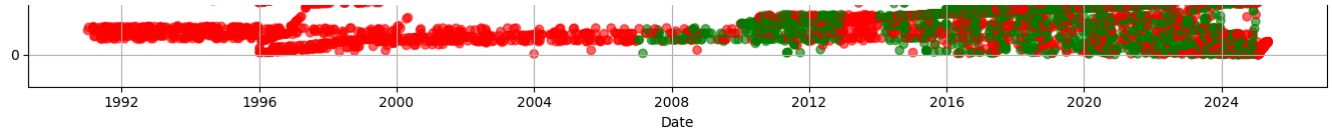





```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt

# --- Prepare gun violence data ---
gun_df = pd.read_csv("chicago_shooting.csv")
gun_df = gun_df.dropna(subset=["LONGITUDE", "LATITUDE", "DATE", "GUNSHOT_INJURY_I"])
gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
gun_df["GUNSHOT_INJURY_I"] = gun_df["GUNSHOT_INJURY_I"].str.strip().str.upper().map({"YES": 1, "NO": 0})

# --- Convert to GeoDataFrame ---
all_gun_points = gpd.GeoDataFrame(
    gun_df,
    geometry=gpd.points_from_xy(gun_df["LONGITUDE"], gun_df["LATITUDE"]),
    crs="EPSG:4326"
)

# Initialize a new column to store if the incident is inside gang territory for the correct year
all_gun_points["Inside_Gang"] = False

# --- Loop through each year to apply the correct gang boundary ---
for year in range(2007, 2025):
    try:
        # Load the gang boundaries for the specific year
        gang_boundaries = gpd.read_file(f"{year}_Gang_Boundaries.shp").explode(index_parts=False)

        # Filter gun incidents for the current year
        gun_points_year = all_gun_points[all_gun_points["DATE"].dt.year == year]

        # Project gun incidents to the same CRS as the gang boundaries
        gun_points_year = gun_points_year.to_crs(gang_boundaries.crs)

        # Classify gun incidents as inside or outside gang territory
        gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)

        # Update the original dataframe with the inside/outside classification for this year
        all_gun_points.loc[all_gun_points["DATE"].dt.year == year, "Inside_Gang"] = gun_points_year["Inside_Gang"]

        print(f"Processed gang boundaries for {year}")
    except Exception as e:
        print(f"⚠️ Could not process year {year}: {e}")

# --- Plot the gun incidents categorized by gang territory status ---
# Use DATE as the x-axis, and use arbitrary values (for example, index or a derived value) for the y-axis
plt.figure(figsize=(14, 8))

# Arbitrary values for y-axis (you can replace with other values if needed)
y_values = range(len(all_gun_points))

# Plot gun incidents over time (DATE)
plt.scatter(all_gun_points["DATE"], y_values, c=all_gun_points["Inside_Gang"].map({True: 'green', False: 'red'}), alpha=0.6)

plt.title("Gun Incidents Over Time Categorized by Gang Territory Status")
plt.xlabel("Date")
plt.ylabel("Arbitrary Values (Index/Derived Value)")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
<ipython-input-49-9f5428c1fee9>:6: DtypeWarning: Columns (25,26,27) have mixed types. Specify dtype option on import or set low_memory=F
  gun_df = pd.read_csv("chicago_shooting.csv")
<ipython-input-49-9f5428c1fee9>:8: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `da
  gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
<ipython-input-49-9f5428c1fee9>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2007
<ipython-input-49-9f5428c1fee9>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2008
<ipython-input-49-9f5428c1fee9>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2009
<ipython-input-49-9f5428c1fee9>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2010
<ipython-input-49-9f5428c1fee9>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2011
Processed gang boundaries for 2012
⚠️ Could not process year 2013: 2013_Gang_Boundaries.shp: No such file or directory
<ipython-input-49-9f5428c1fee9>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-49-9f5428c1fee9>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2014
Processed gang boundaries for 2015
<ipython-input-49-9f5428c1fee9>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-49-9f5428c1fee9>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2016
Processed gang boundaries for 2017
<ipython-input-49-9f5428c1fee9>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-49-9f5428c1fee9>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2018
Processed gang boundaries for 2019
<ipython-input-49-9f5428c1fee9>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-49-9f5428c1fee9>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2020
Processed gang boundaries for 2021
<ipython-input-49-9f5428c1fee9>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-49-9f5428c1fee9>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2022
Processed gang boundaries for 2023
<ipython-input-49-9f5428c1fee9>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-49-9f5428c1fee9>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2024
```

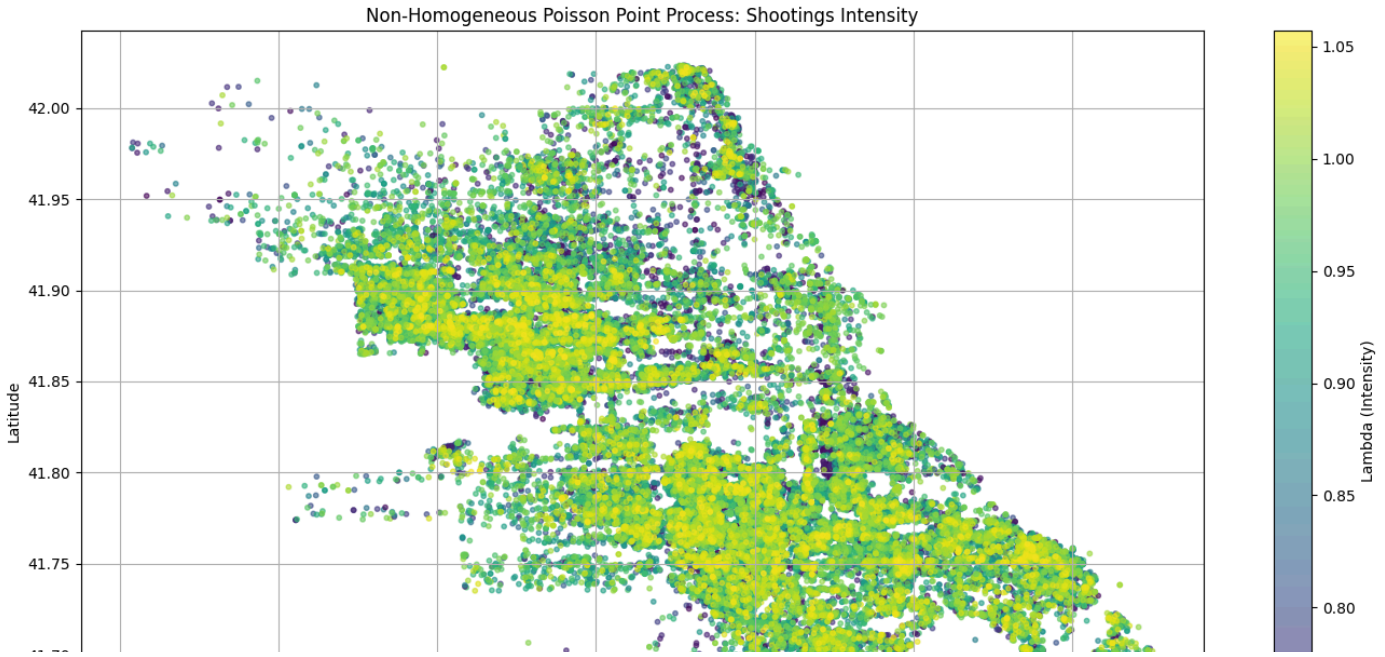


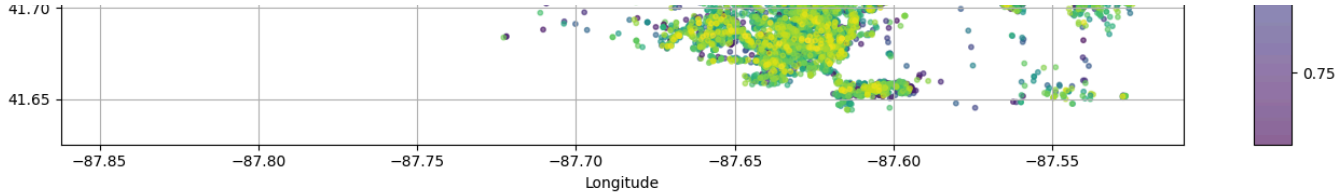



```
import pandas as pd
import geopandas as gpd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.preprocessing import MinMaxScaler

# --- Prepare gun violence data ---
gun_df = pd.read_csv("chicago_shooting.csv")
gun_df = gun_df.dropna(subset=["LONGITUDE", "LATITUDE", "DATE", "GUNSHOT_INJURY_I"])
gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
gun_df["GUNSHOT_INJURY_I"] = gun_df["GUNSHOT_INJURY_I"].str.strip().str.upper().map({"YES": 1, "NO": 0})

# --- Convert to GeoDataFrame ---
all_gun_points = gpd.GeoDataFrame(
    gun_df,
    geometry=gpd.points_from_xy(gun_df["LONGITUDE"], gun_df["LATITUDE"]),
    crs="EPSG:4326"
)

# Initialize a new column to store if the incident is inside gang territory for the correct year
all_gun_points["Inside_Gang"] = False

# --- Loop through each year to apply the correct gang boundary ---
for year in range(2007, 2025):
    try:
        # Load the gang boundaries for the specific year
        gang_boundaries = gpd.read_file(f"{year}_Gang_Boundaries.shp").explode(index_parts=False)

        # Filter gun incidents for the current year
        gun_points_year = all_gun_points[all_gun_points["DATE"].dt.year == year]

        # Project gun incidents to the same CRS as the gang boundaries
        gun_points_year = gun_points_year.to_crs(gang_boundaries.crs)

        # Classify gun incidents as inside or outside gang territory
        gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)

        # Update the original dataframe with the inside/outside classification for this year
        all_gun_points.loc[all_gun_points["DATE"].dt.year == year, "Inside_Gang"] = gun_points_year["Inside_Gang"]

        print(f"Processed gang boundaries for {year}")
    except Exception as e:
        print(f"⚠️ Could not process year {year}: {e}")

# --- Generate Covariates (Spatial and Temporal) ---
# Temporal Covariates
all_gun_points["Year"] = all_gun_points["DATE"].dt.year
all_gun_points["Month"] = all_gun_points["DATE"].dt.month
all_gun_points["DayOfWeek"] = all_gun_points["DATE"].dt.weekday

# Spatial Covariates (e.g., inside gang territory, number of schools in 500m, etc.)
# For simplicity, use "Inside_Gang" as spatial covariate here

# --- Prepare Data for Poisson Model ---
# Aggregating by grid or using a simpler method (e.g., by point)
df_grid = all_gun_points.copy()

# Normalize covariates (optional but recommended for scaling)
scaler = MinMaxScaler()
```

```
df_grid[['Inside_Gang', 'Month', 'Year', 'DayOfWeek']] = scaler.fit_transform(df_grid[['Inside_Gang', 'Month', 'Year', 'DayOfWeek']])

# Poisson regression model to estimate λ(x)
X = df_grid[['Inside_Gang', 'Month', 'Year', 'DayOfWeek']]
X = sm.add_constant(X)  # Add intercept
y = df_grid['GUNSHOT_INJURY_I']  # You can use the shooting count if needed, or injury as target

# Fit Poisson GLM
model = sm.GLM(y, X, family=sm.families.Poisson())
results = model.fit()

# --- Calculate Lambda (Intensity) for each grid cell/point ---
df_grid['Lambda'] = np.exp(
    results.params['const'] +
    results.params['Inside_Gang'] * df_grid['Inside_Gang'] +
    results.params['Month'] * df_grid['Month'] +
    results.params['Year'] * df_grid['Year'] +
    results.params['DayOfWeek'] * df_grid['DayOfWeek']
)

# --- Plot Conditional Intensity (Lambda) ---
# You can visualize the rate function (Lambda) as a heatmap or other plots

plt.figure(figsize=(14, 8))
plt.scatter(df_grid['LONGITUDE'], df_grid['LATITUDE'], c=df_grid['Lambda'], cmap='viridis', s=10, alpha=0.6)
plt.colorbar(label="Lambda (Intensity)")
plt.title("Non-Homogeneous Poisson Point Process: Shootings Intensity")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
<ipython-input-50-adca24ed8780>:9: DtypeWarning: Columns (25,26,27) have mixed types. Specify dtype option on import or set low_memory=F
  gun_df = pd.read_csv("chicago_shooting.csv")
<ipython-input-50-adca24ed8780>:11: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `d
  gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
<ipython-input-50-adca24ed8780>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2007
<ipython-input-50-adca24ed8780>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2008
<ipython-input-50-adca24ed8780>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2009
<ipython-input-50-adca24ed8780>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2010
<ipython-input-50-adca24ed8780>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2011
<ipython-input-50-adca24ed8780>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2012
⚠️ Could not process year 2013: 2013_Gang_Boundaries.shp: No such file or directory
<ipython-input-50-adca24ed8780>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2014
<ipython-input-50-adca24ed8780>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2015
<ipython-input-50-adca24ed8780>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2016
Processed gang boundaries for 2017
<ipython-input-50-adca24ed8780>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-50-adca24ed8780>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2018
Processed gang boundaries for 2019
<ipython-input-50-adca24ed8780>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-50-adca24ed8780>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2020
Processed gang boundaries for 2021
<ipython-input-50-adca24ed8780>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-50-adca24ed8780>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2022
Processed gang boundaries for 2023
<ipython-input-50-adca24ed8780>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-50-adca24ed8780>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2024
```

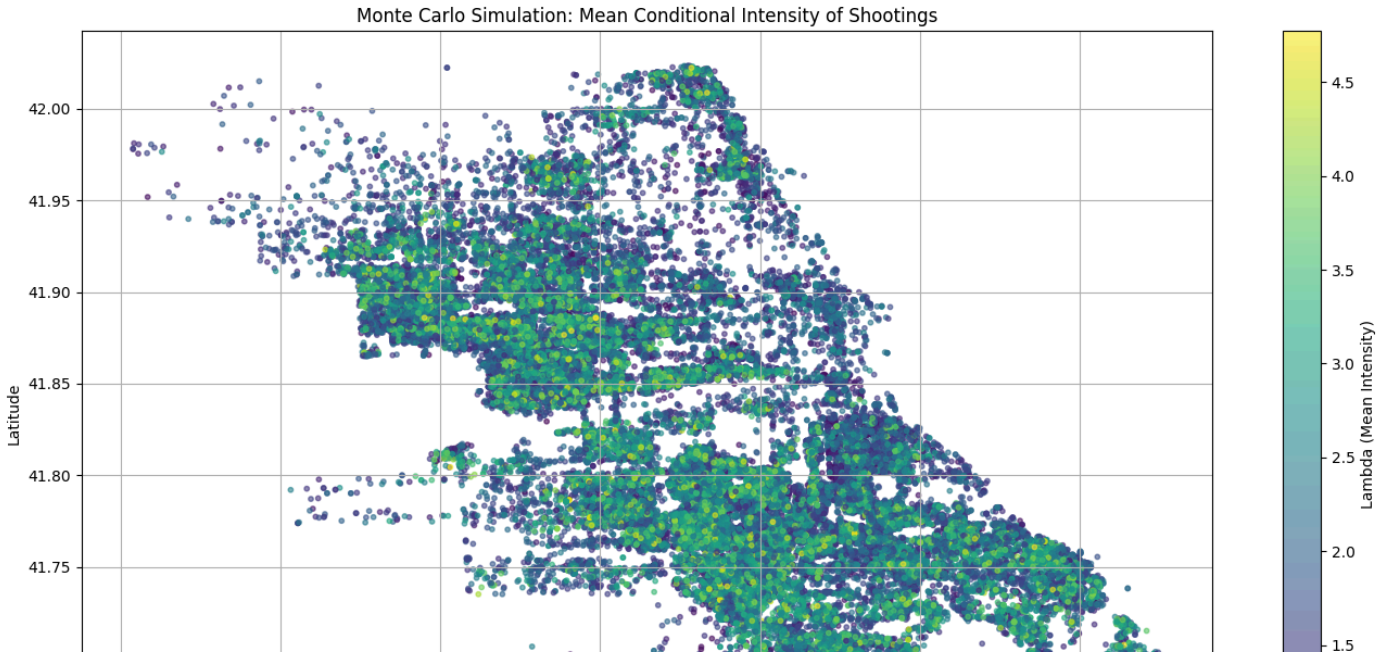


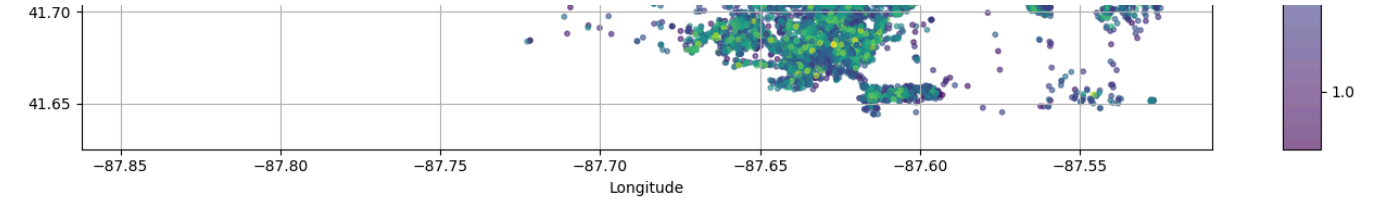


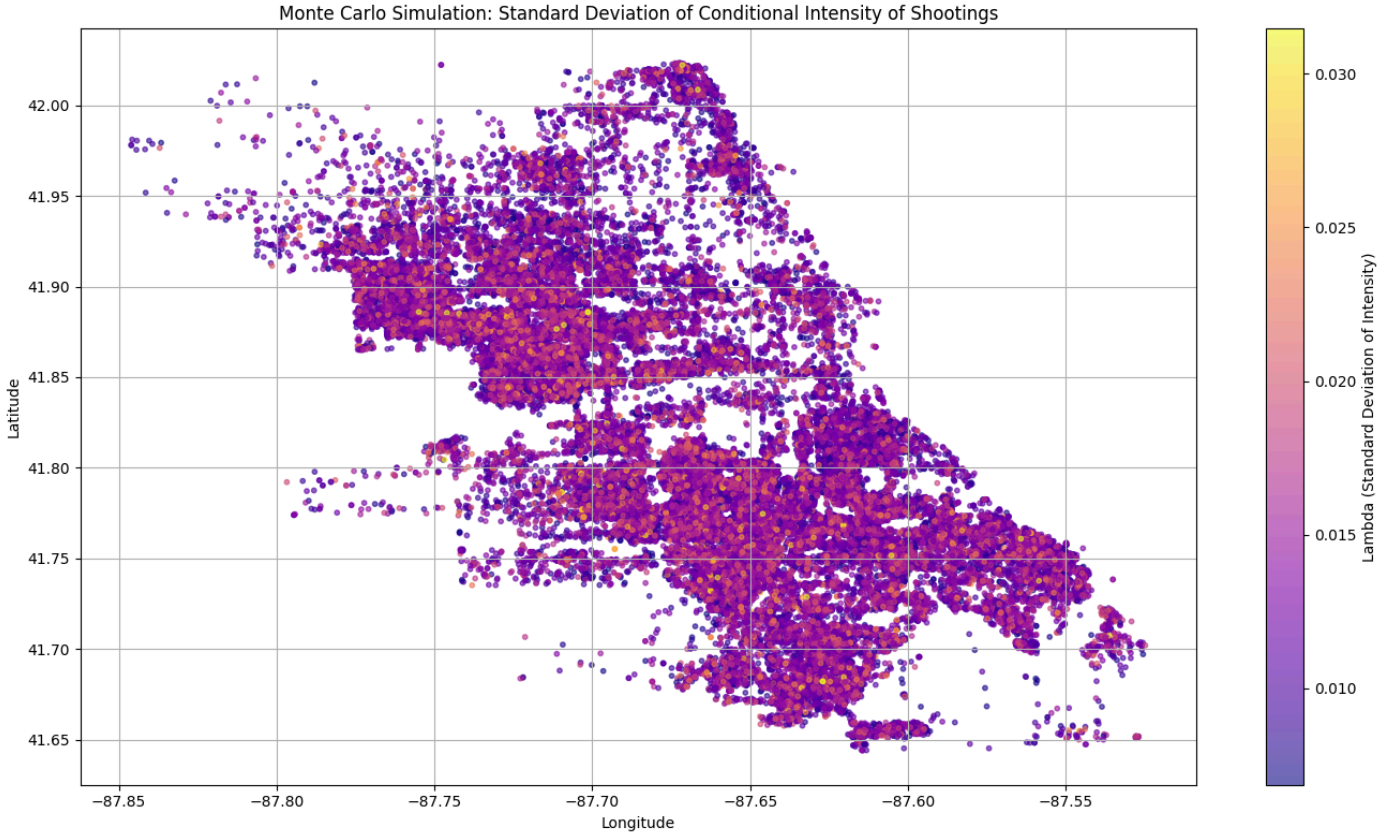

```
import pandas as pd
import geopandas as gpd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.preprocessing import MinMaxScaler

# --- Prepare gun violence data ---
gun_df = pd.read_csv("chicago_shooting.csv")
gun_df = gun_df.dropna(subset=["LONGITUDE", "LATITUDE", "DATE", "GUNSHOT_INJURY_I"])
gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
gun_df["GUNSHOT_INJURY_I"] = gun_df["GUNSHOT_INJURY_I"].str.strip().str.upper().map({"YES": 1, "NO": 0})

# --- Convert to GeoDataFrame ---
all_gun_points = gpd.GeoDataFrame(
    gun_df,
    geometry=gpd.points_from_xy(gun_df["LONGITUDE"], gun_df["LATITUDE"]),
    crs="EPSG:4326"
)

# Initialize a new column to store if the incident is inside gang territory for the correct year
all_gun_points["Inside_Gang"] = False

# --- Loop through each year to apply the correct gang boundary ---
for year in range(2007, 2025):
    try:
        # Load the gang boundaries for the specific year
        gang_boundaries = gpd.read_file(f"{year}_Gang_Boundaries.shp").explode(index_parts=False)

        # Filter gun incidents for the current year
        gun_points_year = all_gun_points[all_gun_points["DATE"].dt.year == year]

        # Project gun incidents to the same CRS as the gang boundaries
        gun_points_year = gun_points_year.to_crs(gang_boundaries.crs)

        # Classify gun incidents as inside or outside gang territory
        gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)

        # Update the original dataframe with the inside/outside classification for this year
        all_gun_points.loc[all_gun_points["DATE"].dt.year == year, "Inside_Gang"] = gun_points_year["Inside_Gang"]

        print(f"Processed gang boundaries for {year}")
    except Exception as e:
        print(f"⚠️ Could not process year {year}: {e}")

# --- Generate Covariates (Spatial and Temporal) ---
# Temporal Covariates
all_gun_points["Year"] = all_gun_points["DATE"].dt.year
all_gun_points["Month"] = all_gun_points["DATE"].dt.month
all_gun_points["DayOfWeek"] = all_gun_points["DATE"].dt.weekday

# Spatial Covariates (e.g., inside gang territory, number of schools in 500m, etc.)
# For simplicity, use "Inside_Gang" as spatial covariate here

# --- Prepare Data for Poisson Model ---
# Aggregating by grid or using a simpler method (e.g., by point)
df_grid = all_gun_points.copy()

# Normalize covariates (optional but recommended for scaling)
scaler = MinMaxScaler()
```

```
df_grid[['Inside_Gang', 'Month', 'Year', 'DayOfWeek']] = scaler.fit_transform(df_grid[['Inside_Gang', 'Month', 'Year', 'DayOfWeek']])

# --- Monte Carlo Simulation ---
n_simulations = 1000
lambdas_simulations = []

# Run multiple simulations
for i in range(n_simulations):
    # Simulate Poisson outcomes (this step could be modified based on data structure)
    simulated_y = np.random.poisson(lam=df_grid['Inside_Gang'] + df_grid['Month'] + df_grid['Year'] + df_grid['DayOfWeek'])

    # Fit Poisson regression model for each simulation
    X = df_grid[['Inside_Gang', 'Month', 'Year', 'DayOfWeek']]
    X = sm.add_constant(X)  # Add intercept
    y = simulated_y

    # Fit Poisson GLM
    model = sm.GLM(y, X, family=sm.families.Poisson())
    results = model.fit()

    # Calculate Lambda (Intensity) for each grid cell/point
    lambda_sim = np.exp(
        results.params['const'] +
        results.params['Inside_Gang'] * df_grid['Inside_Gang'] +
        results.params['Month'] * df_grid['Month'] +
        results.params['Year'] * df_grid['Year'] +
        results.params['DayOfWeek'] * df_grid['DayOfWeek']
    )

    lambdas_simulations.append(lambda_sim)

# --- Aggregate Results ---
# Convert the list of lambdas into a DataFrame (each simulation as a column)
lambdas_simulations = np.array(lambdas_simulations).T

# Calculate the mean and standard deviation of the intensity (Lambda) across simulations
lambda_mean = lambdas_simulations.mean(axis=1)
lambda_std = lambdas_simulations.std(axis=1)

# --- Plot Conditional Intensity Heatmap ---
# Create a heatmap showing the mean intensity across simulations
plt.figure(figsize=(14, 8))
plt.scatter(df_grid['LONGITUDE'], df_grid['LATITUDE'], c=lambda_mean, cmap='viridis', s=10, alpha=0.6)
plt.colorbar(label="Lambda (Mean Intensity)")
plt.title("Monte Carlo Simulation: Mean Conditional Intensity of Shootings")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.grid(True)
plt.tight_layout()
plt.show()

# Optionally, you can plot the standard deviation of lambda
plt.figure(figsize=(14, 8))
plt.scatter(df_grid['LONGITUDE'], df_grid['LATITUDE'], c=lambda_std, cmap='plasma', s=10, alpha=0.6)
plt.colorbar(label="Lambda (Standard Deviation of Intensity)")
plt.title("Monte Carlo Simulation: Standard Deviation of Conditional Intensity of Shootings")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.grid(True)
plt.tight_layout()
plt.show()
```

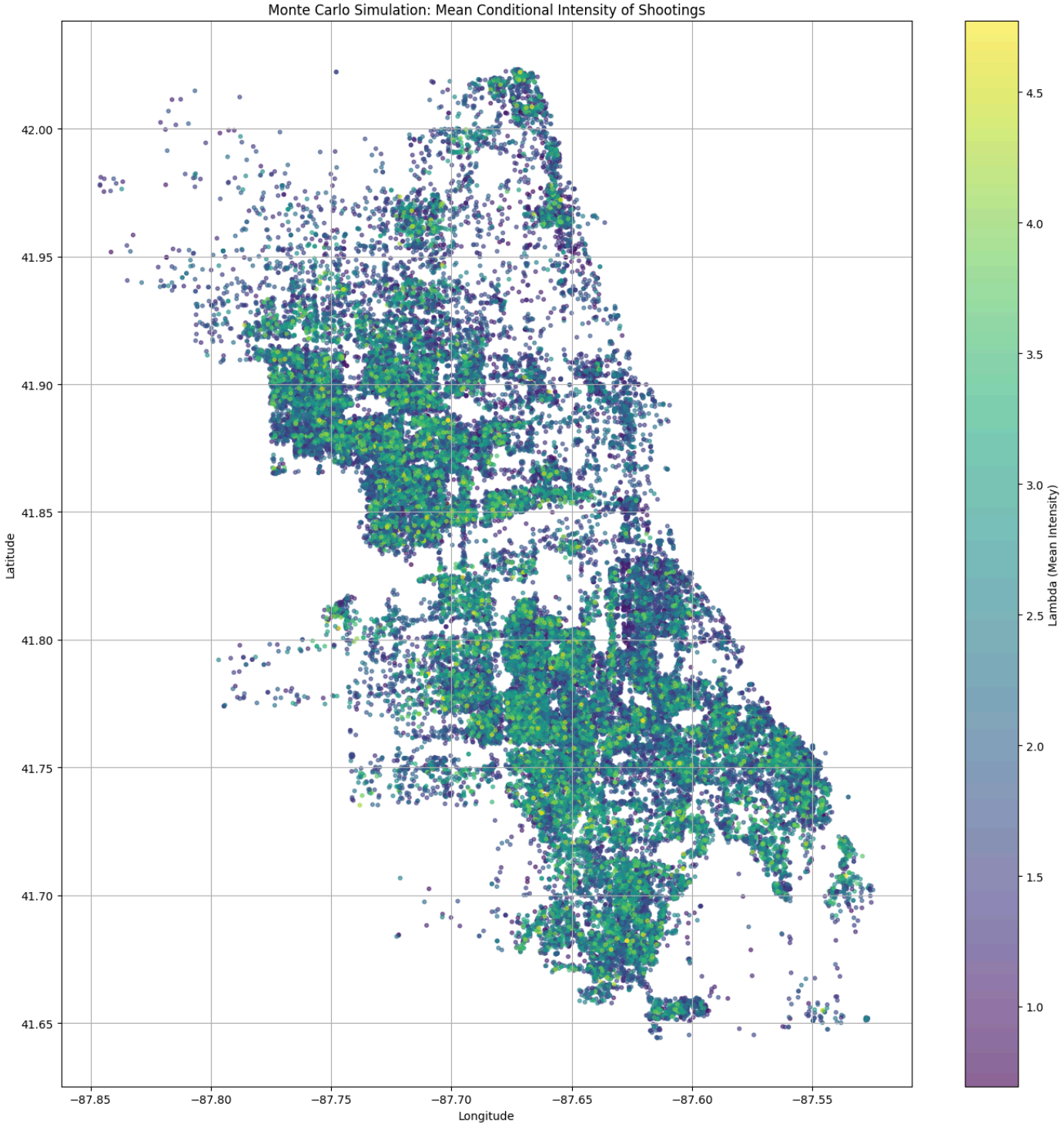
```
<ipython-input-54-a6fe6e0a8eda>:9: DtypeWarning: Columns (25,26,27) have mixed types. Specify dtype option on import or set low_memory=F
  gun_df = pd.read_csv("chicago_shooting.csv")
<ipython-input-54-a6fe6e0a8eda>:11: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `d
  gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
<ipython-input-54-a6fe6e0a8eda>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2007
<ipython-input-54-a6fe6e0a8eda>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2008
<ipython-input-54-a6fe6e0a8eda>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2009
Processed gang boundaries for 2010
<ipython-input-54-a6fe6e0a8eda>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-54-a6fe6e0a8eda>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2011
Processed gang boundaries for 2012
⚠️ Could not process year 2013: 2013_Gang_Boundaries.shp: No such file or directory
<ipython-input-54-a6fe6e0a8eda>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-54-a6fe6e0a8eda>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2014
Processed gang boundaries for 2015
<ipython-input-54-a6fe6e0a8eda>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-54-a6fe6e0a8eda>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2016
<ipython-input-54-a6fe6e0a8eda>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2017
<ipython-input-54-a6fe6e0a8eda>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2018
<ipython-input-54-a6fe6e0a8eda>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2019
<ipython-input-54-a6fe6e0a8eda>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2020
<ipython-input-54-a6fe6e0a8eda>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2021
<ipython-input-54-a6fe6e0a8eda>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2022
<ipython-input-54-a6fe6e0a8eda>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2023
<ipython-input-54-a6fe6e0a8eda>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2024
```




41.70
41.65
−87.85
−87.80
−87.75
−87.70
−87.65
−87.60
−87.55
Longitude
1.0


Monte Carlo Simulation: Standard Deviation of Conditional Intensity of Shootings
Latitude
42.00
41.95
41.90
41.85
41.80
41.75
41.70
41.65
−87.85
−87.80
−87.75
−87.70
−87.65
−87.60
−87.55
Longitude
0.030
0.025
0.020
0.015
0.010
Lambda (Standard Deviation of Intensity)

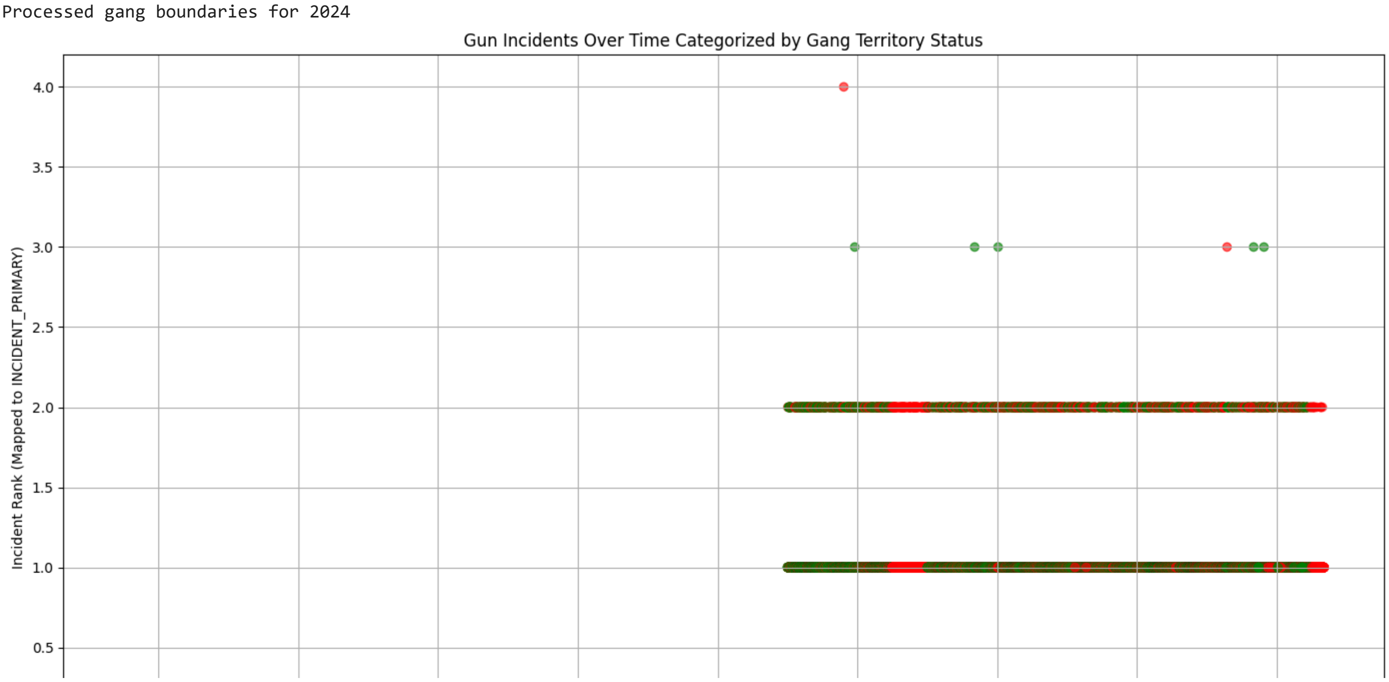
```
# --- Plot Conditional Intensity Heatmap ---
# Create a heatmap showing the mean intensity across simulations
plt.figure(figsize=(14, 14))
plt.scatter(df_grid['LONGITUDE'], df_grid['LATITUDE'], c=lambda_mean, cmap='viridis', s=10, alpha=0.6)
plt.colorbar(label="Lambda (Mean Intensity)")
plt.title("Monte Carlo Simulation: Mean Conditional Intensity of Shootings")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.grid(True)
plt.tight_layout()
plt.show()
```

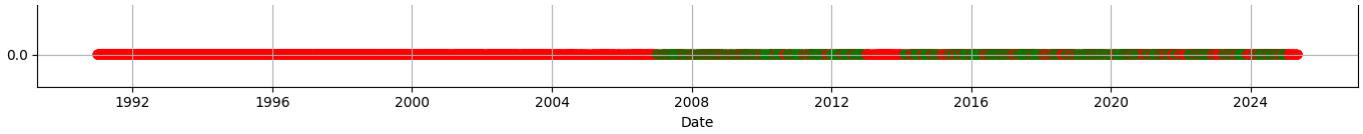



```
import pandas as pd
import geopandas as gpd
```

```
import matplotlib.pyplot as plt

# --- Prepare gun violence data ---
gun_df = pd.read_csv("chicago_shooting.csv")
gun_df = gun_df.dropna(subset=["LONGITUDE", "LATITUDE", "DATE", "GUNSHOT_INJURY_I", "INCIDENT_PRIMARY"])
gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
gun_df["GUNSHOT_INJURY_I"] = gun_df["GUNSHOT_INJURY_I"].str.strip().str.upper().map({"YES": 1, "NO": 0})

# --- Convert to GeoDataFrame ---
all_gun_points = gpd.GeoDataFrame(
    gun_df,
    geometry=gpd.points_from_xy(gun_df["LONGITUDE"], gun_df["LATITUDE"]),
    crs="EPSG:4326"
)

# Initialize a new column to store if the incident is inside gang territory for the correct year
all_gun_points["Inside_Gang"] = False

# --- Loop through each year to apply the correct gang boundary ---
for year in range(2007, 2025):
    try:
        # Load the gang boundaries for the specific year
        gang_boundaries = gpd.read_file(f"{year}_Gang_Boundaries.shp").explode(index_parts=False)

        # Filter gun incidents for the current year
        gun_points_year = all_gun_points[all_gun_points["DATE"].dt.year == year]

        # Project gun incidents to the same CRS as the gang boundaries
        gun_points_year = gun_points_year.to_crs(gang_boundaries.crs)

        # Classify gun incidents as inside or outside gang territory
        gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)

        # Update the original dataframe with the inside/outside classification for this year
        all_gun_points.loc[all_gun_points["DATE"].dt.year == year, "Inside_Gang"] = gun_points_year["Inside_Gang"]

        print(f"Processed gang boundaries for {year}")
    except Exception as e:
        print(f"⚠️ Could not process year {year}: {e}")

# --- Map INCIDENT_PRIMARY to numerical values for y-axis ---
incident_categories = all_gun_points["INCIDENT_PRIMARY"].unique()
incident_mapping = {incident: idx for idx, incident in enumerate(incident_categories)}
all_gun_points["Incident_Rank"] = all_gun_points["INCIDENT_PRIMARY"].map(incident_mapping)

# --- Plot the gun incidents categorized by gang territory status ---
plt.figure(figsize=(14, 8))

# Plot gun incidents over time (DATE)
plt.scatter(all_gun_points["DATE"], all_gun_points["Incident_Rank"],
            c=all_gun_points["Inside_Gang"].map({True: 'green', False: 'red'}), alpha=0.6)

plt.title("Gun Incidents Over Time Categorized by Gang Territory Status")
plt.xlabel("Date")
plt.ylabel("Incident Rank (Mapped to INCIDENT_PRIMARY)")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
<ipython-input-108-7c783fc7a769>:6: DtypeWarning: Columns (25,26,27) have mixed types. Specify dtype option on import or set low_memory=
  gun_df = pd.read_csv("chicago_shooting.csv")
<ipython-input-108-7c783fc7a769>:8: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `d
  gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
<ipython-input-108-7c783fc7a769>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2007
<ipython-input-108-7c783fc7a769>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2008
<ipython-input-108-7c783fc7a769>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2009
<ipython-input-108-7c783fc7a769>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2010
Processed gang boundaries for 2011
<ipython-input-108-7c783fc7a769>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-108-7c783fc7a769>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2012
⚠️ Could not process year 2013: 2013_Gang_Boundaries.shp: No such file or directory
Processed gang boundaries for 2014
<ipython-input-108-7c783fc7a769>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-108-7c783fc7a769>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2015
Processed gang boundaries for 2016
<ipython-input-108-7c783fc7a769>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-108-7c783fc7a769>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-108-7c783fc7a769>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2017
Processed gang boundaries for 2018
<ipython-input-108-7c783fc7a769>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-108-7c783fc7a769>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2019
Processed gang boundaries for 2020
<ipython-input-108-7c783fc7a769>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-108-7c783fc7a769>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2021
Processed gang boundaries for 2022
<ipython-input-108-7c783fc7a769>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-108-7c783fc7a769>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2023
Processed gang boundaries for 2024
```

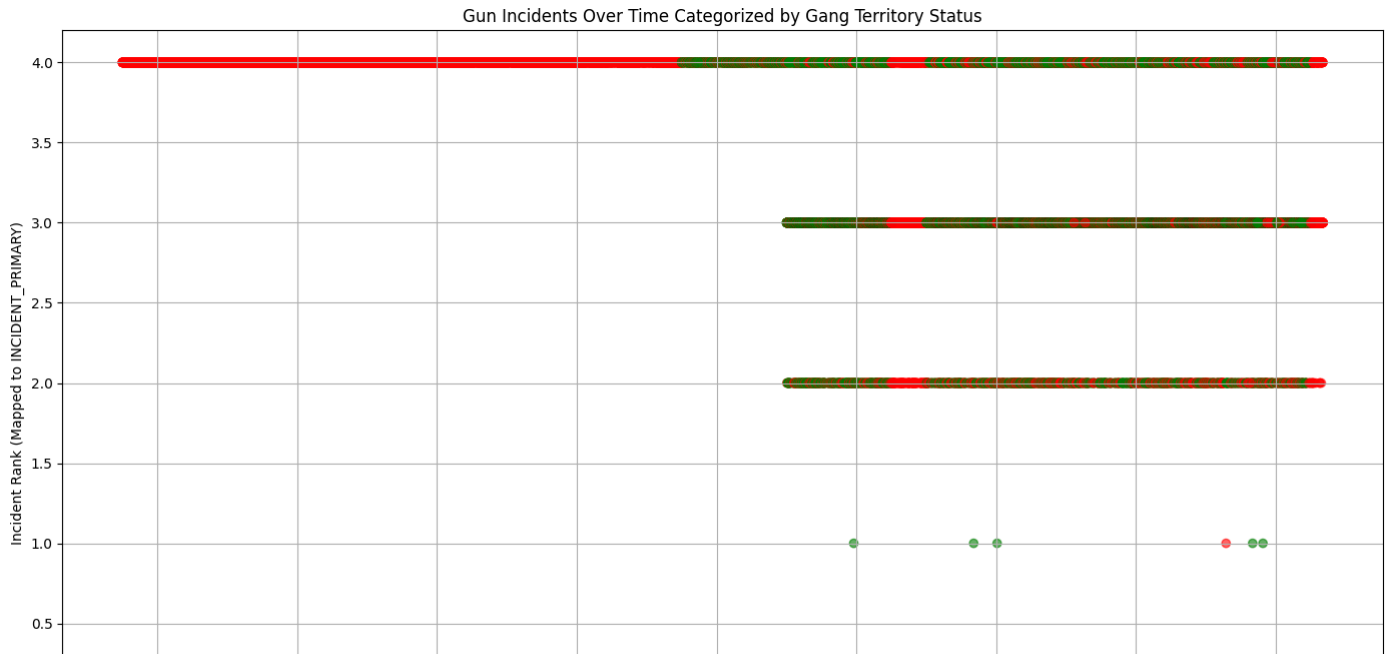


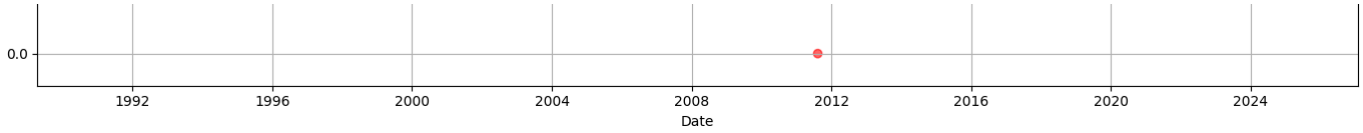



```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt

# --- Prepare gun violence data ---
gun_df = pd.read_csv("chicago_shooting.csv")
gun_df = gun_df.dropna(subset=["LONGITUDE", "LATITUDE", "DATE", "GUNSHOT_INJURY_I", "INCIDENT_PRIMARY"])
gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
gun_df["GUNSHOT_INJURY_I"] = gun_df["GUNSHOT_INJURY_I"].str.strip().str.upper().map({"YES": 1, "NO": 0})

# --- Convert to GeoDataFrame ---
all_gun_points = gpd.GeoDataFrame(
    gun_df,
    geometry=gpd.points_from_xy(gun_df["LONGITUDE"], gun_df["LATITUDE"]),
    crs="EPSG:4326"
)

# Initialize a new column to store if the incident is inside gang territory for the correct year
all_gun_points["Inside_Gang"] = False

# --- Loop through each year to apply the correct gang boundary ---
for year in range(2007, 2025):
    try:
        # Load the gang boundaries for the specific year
        gang_boundaries = gpd.read_file(f"{year}_Gang_Boundaries.shp").explode(index_parts=False)

        # Filter gun incidents for the current year
        gun_points_year = all_gun_points[all_gun_points["DATE"].dt.year == year]

        # Project gun incidents to the same CRS as the gang boundaries
        gun_points_year = gun_points_year.to_crs(gang_boundaries.crs)

        # Classify gun incidents as inside or outside gang territory
        gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)

        # Update the original dataframe with the inside/outside classification for this year
        all_gun_points.loc[all_gun_points["DATE"].dt.year == year, "Inside_Gang"] = gun_points_year["Inside_Gang"]

        print(f"Processed gang boundaries for {year}")
    except Exception as e:
        print(f"⚠️ Could not process year {year}: {e}")

# --- Map INCIDENT_PRIMARY to numerical values for y-axis ---
incident_categories = all_gun_points["INCIDENT_PRIMARY"].unique()
incident_mapping = {incident: idx for idx, incident in enumerate(reversed(incident_categories))}
all_gun_points["Incident_Rank"] = all_gun_points["INCIDENT_PRIMARY"].map(incident_mapping)

# --- Plot the gun incidents categorized by gang territory status ---
plt.figure(figsize=(14, 8))

# Plot gun incidents over time (DATE)
plt.scatter(all_gun_points["DATE"], all_gun_points["Incident_Rank"],
            c=all_gun_points["Inside_Gang"].map({True: 'green', False: 'red'}), alpha=0.6)

plt.title("Gun Incidents Over Time Categorized by Gang Territory Status")
plt.xlabel("Date")
plt.ylabel("Incident Rank (Mapped to INCIDENT_PRIMARY)")
plt.grid(True)
plt.tight_layout()
```

```
    plt.show()
```

```
<ipython-input-113-66d3d6293470>:6: DtypeWarning: Columns (25,26,27) have mixed types. Specify dtype option on import or set low_memory=
  gun_df = pd.read_csv("chicago_shooting.csv")
<ipython-input-113-66d3d6293470>:8: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `d
  gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
<ipython-input-113-66d3d6293470>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2007
<ipython-input-113-66d3d6293470>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2008
<ipython-input-113-66d3d6293470>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2009
<ipython-input-113-66d3d6293470>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2010
<ipython-input-113-66d3d6293470>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2011
Processed gang boundaries for 2012
⚠️ Could not process year 2013: 2013_Gang_Boundaries.shp: No such file or directory
<ipython-input-113-66d3d6293470>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-113-66d3d6293470>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2014
<ipython-input-113-66d3d6293470>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2015
<ipython-input-113-66d3d6293470>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2016
<ipython-input-113-66d3d6293470>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2017
Processed gang boundaries for 2018
<ipython-input-113-66d3d6293470>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-113-66d3d6293470>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2019
Processed gang boundaries for 2020
<ipython-input-113-66d3d6293470>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-113-66d3d6293470>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2021
Processed gang boundaries for 2022
<ipython-input-113-66d3d6293470>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-113-66d3d6293470>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2023
<ipython-input-113-66d3d6293470>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2024
```

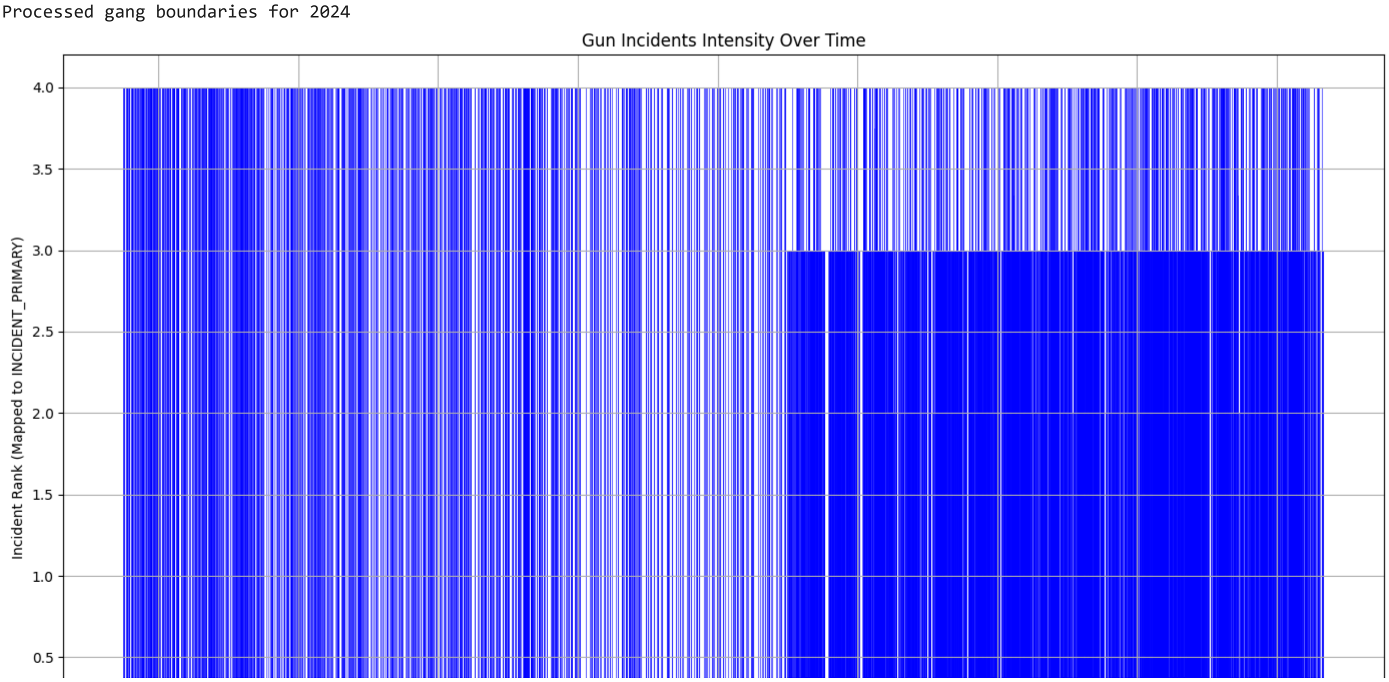





```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt

# --- Prepare gun violence data ---
gun_df = pd.read_csv("chicago_shooting.csv")
gun_df = gun_df.dropna(subset=["LONGITUDE", "LATITUDE", "DATE", "GUNSHOT_INJURY_I", "INCIDENT_PRIMARY"])
gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
gun_df["GUNSHOT_INJURY_I"] = gun_df["GUNSHOT_INJURY_I"].str.strip().str.upper().map({"YES": 1, "NO": 0})

# --- Convert to GeoDataFrame ---
all_gun_points = gpd.GeoDataFrame(
    gun_df,
    geometry=gpd.points_from_xy(gun_df["LONGITUDE"], gun_df["LATITUDE"]),
    crs="EPSG:4326"
)

# Initialize a new column to store if the incident is inside gang territory for the correct year
all_gun_points["Inside_Gang"] = False

# --- Loop through each year to apply the correct gang boundary ---
for year in range(2007, 2025):
    try:
        # Load the gang boundaries for the specific year
        gang_boundaries = gpd.read_file(f"{year}_Gang_Boundaries.shp").explode(index_parts=False)

        # Filter gun incidents for the current year
        gun_points_year = all_gun_points[all_gun_points["DATE"].dt.year == year]

        # Project gun incidents to the same CRS as the gang boundaries
        gun_points_year = gun_points_year.to_crs(gang_boundaries.crs)

        # Classify gun incidents as inside or outside gang territory
        gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)

        # Update the original dataframe with the inside/outside classification for this year
        all_gun_points.loc[all_gun_points["DATE"].dt.year == year, "Inside_Gang"] = gun_points_year["Inside_Gang"]

        print(f"Processed gang boundaries for {year}")
    except Exception as e:
        print(f"⚠️ Could not process year {year}: {e}")

# --- Map INCIDENT_PRIMARY to numerical values for y-axis ---
incident_categories = all_gun_points["INCIDENT_PRIMARY"].unique()
incident_mapping = {incident: idx for idx, incident in enumerate(reversed(incident_categories))}
all_gun_points["Incident_Rank"] = all_gun_points["INCIDENT_PRIMARY"].map(incident_mapping)

# --- Plot the gun incidents with vertical bars representing intensity ---
plt.figure(figsize=(14, 8))

# Plot gun incidents over time (DATE) as vertical bars
dates = all_gun_points["DATE"].dt.date  # Extract date without time
incident_ranks = all_gun_points["Incident_Rank"]

# Group by date and take the mean Incident_Rank for each date (to avoid overlapping points)
grouped = all_gun_points.groupby("DATE")["Incident_Rank"].mean()

# Create bar plot
plt.bar(grouped.index, grouped.values, color='blue', alpha=0.6)
```

```
plt.title("Gun Incidents Intensity Over Time")
plt.xlabel("Date")
plt.ylabel("Incident Rank (Mapped to INCIDENT_PRIMARY)")
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
<ipython-input-103-c227f6af7c5f>:6: DtypeWarning: Columns (25,26,27) have mixed types. Specify dtype option on import or set low_memory=
  gun_df = pd.read_csv("chicago_shooting.csv")
<ipython-input-103-c227f6af7c5f>:8: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `d
  gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
<ipython-input-103-c227f6af7c5f>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2007
<ipython-input-103-c227f6af7c5f>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2008
<ipython-input-103-c227f6af7c5f>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2009
<ipython-input-103-c227f6af7c5f>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2010
<ipython-input-103-c227f6af7c5f>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2011
Processed gang boundaries for 2012
⚠️ Could not process year 2013: 2013_Gang_Boundaries.shp: No such file or directory
<ipython-input-103-c227f6af7c5f>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-103-c227f6af7c5f>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2014
<ipython-input-103-c227f6af7c5f>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2015
<ipython-input-103-c227f6af7c5f>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2016
Processed gang boundaries for 2017
<ipython-input-103-c227f6af7c5f>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-103-c227f6af7c5f>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-103-c227f6af7c5f>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2018
Processed gang boundaries for 2019
<ipython-input-103-c227f6af7c5f>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-103-c227f6af7c5f>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2020
Processed gang boundaries for 2021
<ipython-input-103-c227f6af7c5f>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-103-c227f6af7c5f>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2022
Processed gang boundaries for 2023
<ipython-input-103-c227f6af7c5f>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2024
```




```
all_gun_points["Incident_Rank"] = all_gun_points["INCIDENT_PRIMARY"].map(incident_mapping)
all_gun_points
```

	CASE_NUMBER	DATE	BLOCK	VICTIMIZATION_PRIMARY	INCIDENT_PRIMARY	GUNSHOT_INJURY_I	UNIQUE_ID	ZIP_CODE	WARD	COMMUNIT
0	JF167335	2022-03-08 15:27:00	6000 N KENMORE AVE	HOMICIDE	HOMICIDE	0	HOM-JF167335-#1	60660.0	48.0	EDGE
1	JG148375	2023-02-11 02:30:00	8400 S WABASH AVE	HOMICIDE	HOMICIDE	1	HOM-JG148375-#1	60619.0	6.0	CH
2	JD438266	2020-11-21 22:15:00	7900 S BRANDON AVE	HOMICIDE	HOMICIDE	1	HOM-JD438266-#1	60617.0	7.0	CH
3	JH317789	2024-06-23 08:11:00	12300 S HALSTED ST	HOMICIDE	HOMICIDE	1	HOM-JH317789-#1	60628.0	9.0	PU
4	JH317789	2024-06-23 08:11:00	12300 S HALSTED ST	HOMICIDE	HOMICIDE	1	HOM-JH317789-#2	60628.0	9.0	PU
...	...	...	...	...	...	...	...	...	...	
61437	JJ130497	2025-01-28 18:36:00	5300 S WABASH AVE	HOMICIDE	HOMICIDE	1	HOM-JJ130497-#1	60615.0	3.0	WASHI
61438	JJ130497	2025-01-28 18:36:00	5300 S WABASH AVE	HOMICIDE	HOMICIDE	1	HOM-JJ130497-#2	60615.0	3.0	WASHI
61439	JJ129588	2025-01-28 04:50:00	7700 S OGLESBY AVE	BATTERY	HOMICIDE	1	SHOOT-JJ129588-#3	60649.0	7.0	SOUTH
61440	JJ129772	2025-01-28 09:11:00	2000 N NEWCASTLE AVE	HOMICIDE	HOMICIDE	1	HOM-JJ129772-#1	60707.0	29.0	
61441	JJ128407	2025-01-27 05:16:00	1700 W GARFIELD BLVD	BATTERY	BATTERY	1	SHOOT-JJ128407-#1	60609.0	16.0	NE

61438 rows × 41 columns

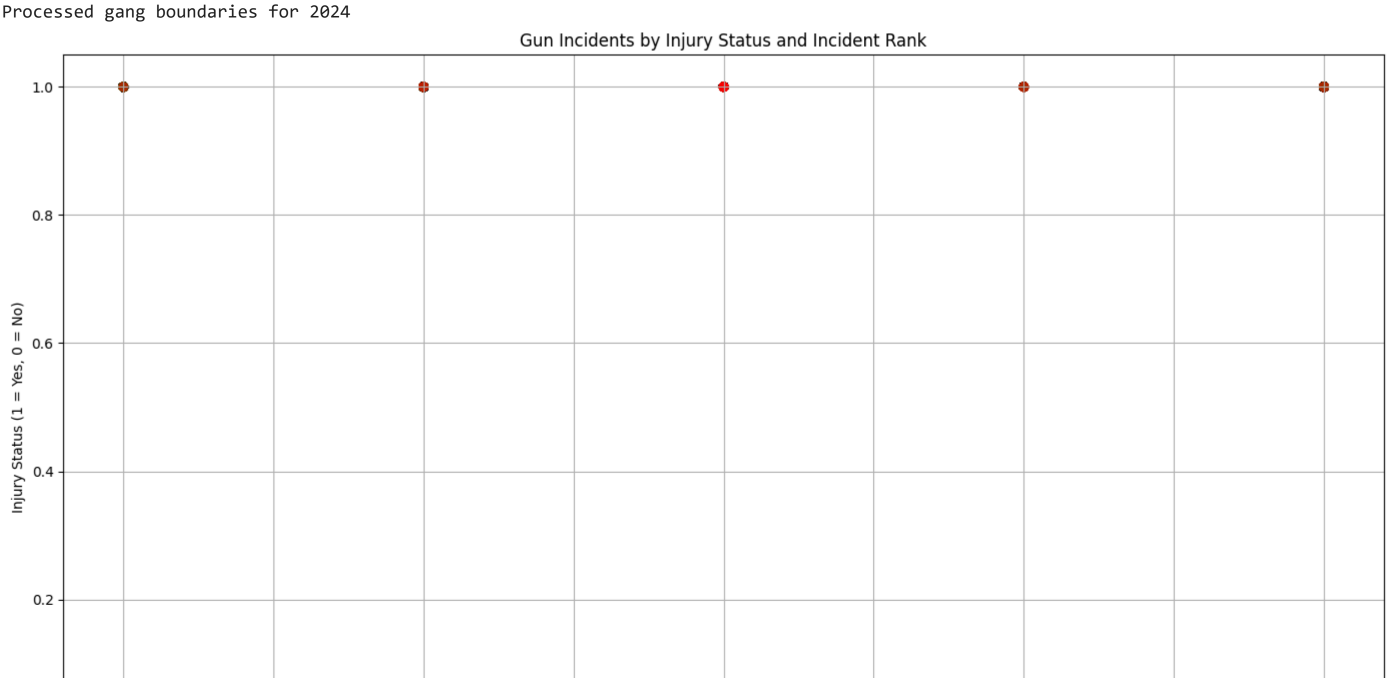
```
# --- Count total number of gun incidents ---
total_points = all_gun_points.shape[0]
print(f"Total number of gun incidents: {total_points}")
```

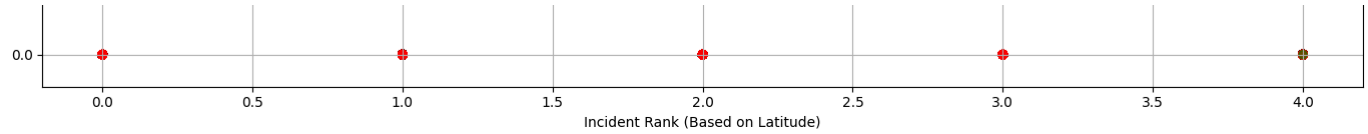
```
Total number of gun incidents: 61438
```

```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt

# --- Prepare gun violence data ---
gun_df = pd.read_csv("chicago_shooting.csv")
gun_df = gun_df.dropna(subset=["LONGITUDE", "LATITUDE", "DATE", "GUNSHOT_INJURY_I"])
gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
```

```
gun_df["GUNSHOT_INJURY_I"] = gun_df["GUNSHOT_INJURY_I"].str.strip().str.upper().map({"YES": 1, "NO": 0})

# --- Create a new column to store 'Incident Rank' ---
# Assuming 'Incident_Rank' is already available in your dataset or it needs to be computed as per some criteria
# Here, we'll use a placeholder rank for the sake of demonstration.
gun_df["Incident_Rank"] = pd.qcut(gun_df["LATITUDE"], q=5, labels=False)  # Just an example using LATITUDE to rank incidents

# --- Convert to GeoDataFrame ---
all_gun_points = gpd.GeoDataFrame(
    gun_df,
    geometry=gpd.points_from_xy(gun_df["LONGITUDE"], gun_df["LATITUDE"]),
    crs="EPSG:4326"
)

# Initialize a new column to store if the incident is inside gang territory for the correct year
all_gun_points["Inside_Gang"] = False

# --- Loop through each year to apply the correct gang boundary ---
for year in range(2007, 2025):
    try:
        # Load the gang boundaries for the specific year
        gang_boundaries = gpd.read_file(f"{year}_Gang_Boundaries.shp").explode(index_parts=False)

        # Filter gun incidents for the current year
        gun_points_year = all_gun_points[all_gun_points["DATE"].dt.year == year]

        # Project gun incidents to the same CRS as the gang boundaries
        gun_points_year = gun_points_year.to_crs(gang_boundaries.crs)

        # Classify gun incidents as inside or outside gang territory
        gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)

        # Update the original dataframe with the inside/outside classification for this year
        all_gun_points.loc[all_gun_points["DATE"].dt.year == year, "Inside_Gang"] = gun_points_year["Inside_Gang"]

        print(f"Processed gang boundaries for {year}")
    except Exception as e:
        print(f"⚠️ Could not process year {year}: {e}")

# --- Plot the gun incidents categorized by injury status and incident rank ---
plt.figure(figsize=(14, 8))

# Plot points with x as 'Incident Rank' and y as 'GUNSHOT_INJURY_I' (1 for injury, 0 for no injury)
plt.scatter(
    all_gun_points["Incident_Rank"],
    all_gun_points["GUNSHOT_INJURY_I"],
    c=all_gun_points["Inside_Gang"].map({True: 'green', False: 'red'}),  # Color by gang territory status
    alpha=0.6
)

plt.title("Gun Incidents by Injury Status and Incident Rank")
plt.xlabel("Incident Rank (Based on Latitude)")
plt.ylabel("Injury Status (1 = Yes, 0 = No)")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
<ipython-input-106-8991809ffc12>:6: DtypeWarning: Columns (25,26,27) have mixed types. Specify dtype option on import or set low_memory=
  gun_df = pd.read_csv("chicago_shooting.csv")
<ipython-input-106-8991809ffc12>:8: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `d
  gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
<ipython-input-106-8991809ffc12>:39: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2007
<ipython-input-106-8991809ffc12>:39: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2008
<ipython-input-106-8991809ffc12>:39: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2009
Processed gang boundaries for 2010
<ipython-input-106-8991809ffc12>:39: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-106-8991809ffc12>:39: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-106-8991809ffc12>:39: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2011
Processed gang boundaries for 2012
⚠️ Could not process year 2013: 2013_Gang_Boundaries.shp: No such file or directory
<ipython-input-106-8991809ffc12>:39: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-106-8991809ffc12>:39: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2014
Processed gang boundaries for 2015
<ipython-input-106-8991809ffc12>:39: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-106-8991809ffc12>:39: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2016
Processed gang boundaries for 2017
<ipython-input-106-8991809ffc12>:39: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-106-8991809ffc12>:39: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2018
Processed gang boundaries for 2019
<ipython-input-106-8991809ffc12>:39: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-106-8991809ffc12>:39: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2020
Processed gang boundaries for 2021
<ipython-input-106-8991809ffc12>:39: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-106-8991809ffc12>:39: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2022
Processed gang boundaries for 2023
<ipython-input-106-8991809ffc12>:39: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2024
```

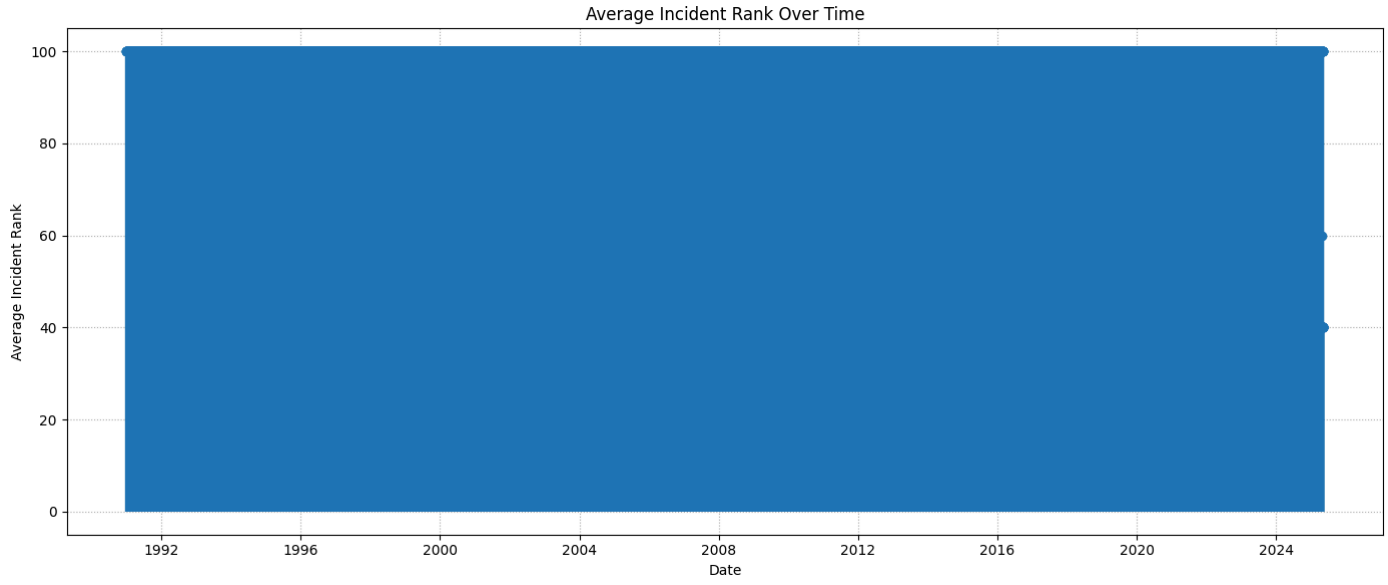





```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt

# --- Prepare gun violence data ---
gun_df = pd.read_csv("chicago_shooting.csv")
gun_df = gun_df.dropna(subset=["LONGITUDE", "LATITUDE", "DATE", "GUNSHOT_INJURY_I", "INCIDENT_PRIMARY"])
gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
gun_df["GUNSHOT_INJURY_I"] = gun_df["GUNSHOT_INJURY_I"].str.strip().str.upper().map({"YES": 1, "NO": 0})

# Create the ranking for the "INCIDENT PRIMARY" column
incident_ranking = {
    'HOMICIDE': 100,
    'CRIMINAL SEXUAL ASSAULT': 80,
    'ROBBERY': 60,
    'BATTERY': 40,
    'NON-FATAL': 20
}

gun_df['Incident_Rank'] = gun_df['INCIDENT_PRIMARY'].map(incident_ranking)

# --- Convert to GeoDataFrame ---
all_gun_points = gpd.GeoDataFrame(
    gun_df,
    geometry=gpd.points_from_xy(gun_df["LONGITUDE"], gun_df["LATITUDE"]),
    crs="EPSG:4326"
)

# Initialize a new column to store if the incident is inside gang territory for the correct year
all_gun_points["Inside_Gang"] = False

# --- Loop through each year to apply the correct gang boundary ---
for year in range(2007, 2025):
    try:
        # Load the gang boundaries for the specific year
        gang_boundaries = gpd.read_file(f"{year}_Gang_Boundaries.shp").explode(index_parts=False)

        # Filter gun incidents for the current year
        gun_points_year = all_gun_points[all_gun_points["DATE"].dt.year == year]

        # Project gun incidents to the same CRS as the gang boundaries
        gun_points_year = gun_points_year.to_crs(gang_boundaries.crs)

        # Classify gun incidents as inside or outside gang territory
        gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)

        # Update the original dataframe with the inside/outside classification for this year
        all_gun_points.loc[all_gun_points["DATE"].dt.year == year, "Inside_Gang"] = gun_points_year["Inside_Gang"]

        print(f"Processed gang boundaries for {year}")
    except Exception as e:
        print(f"⚠️ Could not process year {year}: {e}")

# --- Lollipop chart ---

# Calculate average incident rank by date
incident_rank_by_date = all_gun_points.groupby("DATE")['Incident_Rank'].mean()

# Create lollipop chart
plt.figure(figsize=(14, 6))
```

```
    plt.stem(incident_rank_by_date.index, incident_rank_by_date.values, basefmt=" ")
    # Customize plot
    plt.title("Average Incident Rank Over Time")
    plt.xlabel("Date")
    plt.ylabel("Average Incident Rank")
    plt.grid(True, linestyle=":")

    plt.tight_layout()
    plt.show()
```

```
<ipython-input-104-bf9cc25f286b>:6: DtypeWarning: Columns (25,26,27) have mixed types. Specify dtype option on import or set low_memory=
  gun_df = pd.read_csv("chicago_shooting.csv")
<ipython-input-104-bf9cc25f286b>:8: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `d
  gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
<ipython-input-104-bf9cc25f286b>:45: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2007
<ipython-input-104-bf9cc25f286b>:45: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2008
<ipython-input-104-bf9cc25f286b>:45: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2009
<ipython-input-104-bf9cc25f286b>:45: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2010
Processed gang boundaries for 2011
<ipython-input-104-bf9cc25f286b>:45: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-104-bf9cc25f286b>:45: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2012
⚠️ Could not process year 2013: 2013_Gang_Boundaries.shp: No such file or directory
Processed gang boundaries for 2014
<ipython-input-104-bf9cc25f286b>:45: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-104-bf9cc25f286b>:45: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2015
Processed gang boundaries for 2016
<ipython-input-104-bf9cc25f286b>:45: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-104-bf9cc25f286b>:45: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2017
Processed gang boundaries for 2018
<ipython-input-104-bf9cc25f286b>:45: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-104-bf9cc25f286b>:45: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2019
Processed gang boundaries for 2020
<ipython-input-104-bf9cc25f286b>:45: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-104-bf9cc25f286b>:45: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2021
Processed gang boundaries for 2022
<ipython-input-104-bf9cc25f286b>:45: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-104-bf9cc25f286b>:45: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2023
Processed gang boundaries for 2024
<ipython-input-104-bf9cc25f286b>:45: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
```



```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt

# --- Prepare gun violence data ---
gun_df = pd.read_csv("chicago_shooting.csv")
gun_df = gun_df.dropna(subset=["LONGITUDE", "LATITUDE", "DATE", "GUNSHOT_INJURY_I"])
gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
gun_df["GUNSHOT_INJURY_I"] = gun_df["GUNSHOT_INJURY_I"].str.strip().str.upper().map({"YES": 1, "NO": 0})

# --- Convert to GeoDataFrame ---
all_gun_points = gpd.GeoDataFrame(
    gun_df,
    geometry=gpd.points_from_xy(gun_df["LONGITUDE"], gun_df["LATITUDE"]),
    crs="EPSG:4326"
)

# Initialize a new column to store if the incident is inside gang territory for the correct year
all_gun_points["Inside_Gang"] = False

# --- Loop through each year to apply the correct gang boundary ---
for year in range(2007, 2025):
    try:
        # Load the gang boundaries for the specific year
        gang_boundaries = gpd.read_file(f"{year}_Gang_Boundaries.shp").explode(index_parts=False)

        # Filter gun incidents for the current year
        gun_points_year = all_gun_points[all_gun_points["DATE"].dt.year == year]

        # Project gun incidents to the same CRS as the gang boundaries
        gun_points_year = gun_points_year.to_crs(gang_boundaries.crs)

        # Classify gun incidents as inside or outside gang territory
        gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)

        # Update the original dataframe with the inside/outside classification for this year
        all_gun_points.loc[all_gun_points["DATE"].dt.year == year, "Inside_Gang"] = gun_points_year["Inside_Gang"]

        print(f"Processed gang boundaries for {year}")
    except Exception as e:
        print(f"⚠️ Could not process year {year}: {e}")

# --- Plot the gun incidents categorized by gang territory status and intensity (Incident Rank) ---
plt.figure(figsize=(14, 14))

# Plot points based on intensity (using 'Incident_Rank' as intensity value)
# Size of the points will correspond to the incident rank intensity
plt.scatter(
    all_gun_points[all_gun_points["Inside_Gang"] == True].geometry.x,
    all_gun_points[all_gun_points["Inside_Gang"] == True].geometry.y,
    c=all_gun_points[all_gun_points["Inside_Gang"] == True]["Incident_Rank"],  # Color based on incident rank
    cmap='Reds',  # Color map for intensity
    label="Inside Gang Territory",
    alpha=0.6,
    s=all_gun_points[all_gun_points["Inside_Gang"] == True]["Incident_Rank"] * 10  # Size based on rank (scale it appropriately)
)

plt.scatter(
    all_gun_points[all_gun_points["Inside_Gang"] == False].geometry.x,
    all_gun_points[all_gun_points["Inside_Gang"] == False].geometry.y,
    c=all_gun_points[all_gun_points["Inside_Gang"] == False]["Incident_Rank"],  # Color based on incident rank
    cmap='Blues',  # Color map for intensity
    label="Outside Gang Territory",
    alpha=0.6,
    s=all_gun_points[all_gun_points["Inside_Gang"] == False]["Incident_Rank"] * 10  # Size based on rank (scale it appropriately)
)

plt.title("Gun Incidents Inside vs Outside Gang Territory (2007-2024) - Intensity by Incident Rank")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
```

```
    plt.colorbar(label="Incident Rank")  # Add color bar for intensity
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

```
  gun_df = pd.read_csv("chicago_shooting.csv")
<ipython-input-101-3e64b8c3d234>:8: UserWarning: Could not infer format, so each element will be parsed individually, falling back tc
  gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
<ipython-input-101-3e64b8c3d234>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2007
<ipython-input-101-3e64b8c3d234>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2008
<ipython-input-101-3e64b8c3d234>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2009
Processed gang boundaries for 2010
<ipython-input-101-3e64b8c3d234>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-101-3e64b8c3d234>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-101-3e64b8c3d234>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2011
Processed gang boundaries for 2012
⚠️ Could not process year 2013: 2013_Gang_Boundaries.shp: No such file or directory
<ipython-input-101-3e64b8c3d234>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-101-3e64b8c3d234>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2014
Processed gang boundaries for 2015
Processed gang boundaries for 2016
<ipython-input-101-3e64b8c3d234>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-101-3e64b8c3d234>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2017
Processed gang boundaries for 2018
<ipython-input-101-3e64b8c3d234>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-101-3e64b8c3d234>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2019
Processed gang boundaries for 2020
<ipython-input-101-3e64b8c3d234>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-101-3e64b8c3d234>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2021
Processed gang boundaries for 2022
<ipython-input-101-3e64b8c3d234>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-101-3e64b8c3d234>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2023
<ipython-input-101-3e64b8c3d234>:34: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2024
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3804         try:
-> 3805             return self._engine.get_loc(casted_key)
   3806         except KeyError as err:

index.pyx in pandas._libs.index.IndexEngine.get_loc()

index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'Incident_Rank'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
                                  3 frames
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3810             ):
   3811                 raise InvalidIndexError(key)
-> 3812             raise KeyError(key) from err
   3813         except TypeError:
   3814             # If we have a listlike key, _check_indexing_error will raise
```

```
KeyError: 'Incident_Rank'

<Figure size 1400x1400 with 0 Axes>
```

```
# Make sure both layers are in a projected CRS (meters!)
gun_points_m = gun_points.to_crs(epsg=3435)         # or your local UTM zone
all_schools_m = all_schools_gdf.to_crs(epsg=3435)

# Create a 500-meter buffer around each gun incident
gun_points_m["geometry"] = gun_points_m.geometry.buffer(500)

# Spatial join to count schools within each buffer
joined = gpd.sjoin(all_schools_m, gun_points_m, predicate="within")

# Count number of schools per gunshot buffer
school_counts = joined.groupby("index_right").size()

# Assign to gun_points_m
gun_points_m["Schools_500m"] = gun_points_m.index.map(school_counts).fillna(0).astype(int)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-102-7bc7e63b846c> in <cell line: 0>()
      1 # Make sure both layers are in a projected CRS (meters!)
----> 2 gun_points_m = gun_points.to_crs(epsg=3435)         # or your local UTM zone
      3 all_schools_m = all_schools_gdf.to_crs(epsg=3435)
      4
      5 # Create a 500-meter buffer around each gun incident

NameError: name 'gun_points' is not defined
```

```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
from shapely.geometry import Point

# --- Prepare gun violence data ---
gun_df = pd.read_csv("chicago_shooting.csv")
gun_df = gun_df.dropna(subset=["LONGITUDE", "LATITUDE", "DATE", "GUNSHOT_INJURY_I"])
gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
gun_df["GUNSHOT_INJURY_I"] = gun_df["GUNSHOT_INJURY_I"].str.strip().str.upper().map({"YES": 1, "NO": 0})

# --- Convert to GeoDataFrame ---
all_gun_points = gpd.GeoDataFrame(
    gun_df,
    geometry=gpd.points_from_xy(gun_df["LONGITUDE"], gun_df["LATITUDE"]),
    crs="EPSG:4326"
)

# --- Load private and public school datasets ---
private_school_df = pd.read_csv("Private_schools.csv")
private_school_df = private_school_df[private_school_df["CITY"].str.strip().str.upper() == "CHICAGO"]

chicago_school_df = pd.read_csv("Chicago_schools.csv")

# --- Convert schools to GeoDataFrames ---
private_school_gdf = gpd.GeoDataFrame(
    private_school_df,
    geometry=[Point(xy) for xy in zip(private_school_df["LON"], private_school_df["LAT"])],
    crs="EPSG:4326"
)

public_school_gdf = gpd.GeoDataFrame(
    chicago_school_df,
    geometry=[Point(xy) for xy in zip(chicago_school_df["Long"], chicago_school_df["Lat"])],
    crs="EPSG:4326"
)

# --- Combine both private and public schools into one GeoDataFrame using pandas.concat ---
all_schools_gdf = pd.concat([private_school_gdf, public_school_gdf], ignore_index=True)

# --- Calculate the number of schools within a 500-meter radius ---
def count_schools_within_radius(gun_point, schools_gdf, radius=500):
    """ Count the number of schools within a given radius from the gun incident """
```

```
        count = 0
        for _, school in schools_gdf.iterrows():
            distance = gun_point.distance(school["geometry"])  # in degrees (great-circle distance)
            # Convert distance from degrees to meters (approx. 111.32 km per degree)
            distance_in_meters = distance * 111320
            if distance_in_meters <= radius:
                count += 1
        return count

    # Apply the function to each gun point
    all_gun_points['Num_Schools_500m'] = all_gun_points['geometry'].apply(lambda x: count_schools_within_radius(x, all_schools_gdf))

    # --- Plot Gun Incidents by Date with Number of Schools within 500m as Y-axis ---
    plt.figure(figsize=(14, 8))

    # Scatter plot with Date on the x-axis and number of schools on the y-axis
    plt.scatter(all_gun_points['DATE'], all_gun_points['Num_Schools_500m'], color='blue', alpha=0.6)

    plt.title("Gun Incidents by Date and Number of Schools within 500m (2007-2024)")
    plt.xlabel("Date")
    plt.ylabel("Number of Schools within 500m")
    plt.grid(True, linestyle=":")
    plt.tight_layout()
    plt.show()
```

```
<ipython-input-99-40d0e82eea65>:7: DtypeWarning: Columns (25,26,27) have mixed types. Specify dtype option on import or set low_memory=F
  gun_df = pd.read_csv("chicago_shooting.csv")
<ipython-input-99-40d0e82eea65>:9: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `da
  gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-99-40d0e82eea65> in <cell line: 0>()
     52
     53 # Apply the function to each gun point
---> 54 all_gun_points['Num_Schools_500m'] = all_gun_points['geometry'].apply(lambda x: count_schools_within_radius(x,
all_schools_gdf))
     55
     56 # --- Plot Gun Incidents by Date with Number of Schools within 500m as Y-axis ---

 13 frames
lib.pyx in pandas._libs.lib.map_infer()

/usr/local/lib/python3.11/dist-packages/pandas/_config/config.py in _get_root(key)
    633
    634
--> 635 def _get_root(key: str) -> tuple[dict[str, Any], str]:
    636     path = key.split(".")
    637     cursor = _global_config
```

```
from shapely.geometry import Point
import geopandas as gpd
import pandas as pd
import matplotlib.pyplot as plt

# Assume: all_gun_points and all_schools_gdf are already in EPSG:4326

# Project both to a metric CRS (meters) for accurate distance calculations
gun_points_m = all_gun_points.to_crs(epsg=3857)
schools_m = all_schools_gdf.to_crs(epsg=3857)

# Create 500 meter buffers around each gun incident
gun_points_m["buffer"] = gun_points_m.geometry.buffer(500)

# Convert to GeoDataFrame of buffers for spatial join
gun_buffers = gpd.GeoDataFrame(gun_points_m.drop(columns="geometry"), geometry=gun_points_m["buffer"], crs=gun_points_m.crs)

# Perform spatial join to find schools within each buffer
join_result = gpd.sjoin(schools_m, gun_buffers, predicate="within")

# Count how many schools fall within each gun incident buffer
school_counts = join_result.groupby("index_right").size()

# Add the counts back to the original gun_points
gun_points_m["Num_Schools_500m"] = gun_points_m.index.map(school_counts).fillna(0).astype(int)
```

```
# --- Plot Gun Incidents by Date vs. Nearby School Count ---
plt.figure(figsize=(14, 8))
plt.scatter(gun_points_m["DATE"], gun_points_m["Num_Schools_500m"], color='blue', alpha=0.6)
plt.title("Gun Incidents by Date and Number of Schools within 500m (2007–2024)")
plt.xlabel("Date")
plt.ylabel("Number of Schools within 500m")
plt.grid(True, linestyle=":")
plt.tight_layout()
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-98-9a3e8835bc49> in <cell line: 0>()
      8 # Project both to a metric CRS (meters) for accurate distance calculations
      9 gun_points_m = all_gun_points.to_crs(epsg=3857)
---> 10 schools_m = all_schools_gdf.to_crs(epsg=3857)
     11
     12 # Create 500 meter buffers around each gun incident

NameError: name 'all_schools_gdf' is not defined
```

```
# --- Plot Gun Incidents by Date vs. Nearby School Count, colored by gang territory ---
plt.figure(figsize=(14, 8))

colors = gun_points_m["Inside_Gang"].map({True: "red", False: "blue"})

plt.scatter(
    gun_points_m["DATE"],
    gun_points_m["Num_Schools_500m"],
    c=colors,
    alpha=0.6,
    label=None
)

# Optional legend
from matplotlib.lines import Line2D
legend_elements = [
    Line2D([0], [0], marker='o', color='w', label='Inside Gang Territory', markerfacecolor='red', markersize=8),
    Line2D([0], [0], marker='o', color='w', label='Outside Gang Territory', markerfacecolor='blue', markersize=8)
]
plt.legend(handles=legend_elements)

plt.title("Gun Incidents by Date and Number of Schools within 500m (2007–2024)")
plt.xlabel("Date")
plt.ylabel("Number of Schools within 500m")
plt.grid(True, linestyle=":")
plt.tight_layout()
plt.show()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3804         try:
-> 3805             return self._engine.get_loc(casted_key)
   3806         except KeyError as err:

index.pyx in pandas._libs.index.IndexEngine.get_loc()

index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'Num_Schools_500m'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
                                  3 frames
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3810             ):
   3811                 raise InvalidIndexError(key)
-> 3812             raise KeyError(key) from err
   3813         except TypeError:
   3814             # If we have a listlike key, _check_indexing_error will raise

KeyError: 'Num_Schools_500m'

<Figure size 1400x800 with 0 Axes>
```

```
# Project to meters
gun_points_m = all_gun_points.to_crs(epsg=3857)
schools_m = all_schools_gdf.to_crs(epsg=3857)

# Create buffers
gun_points_m["buffer"] = gun_points_m.geometry.buffer(500)
gun_buffers = gpd.GeoDataFrame(gun_points_m.drop(columns="geometry"), geometry=gun_points_m["buffer"], crs=gun_points_m.crs)

# Join and count schools
join_result = gpd.sjoin(schools_m, gun_buffers, predicate="within")
school_counts = join_result.groupby("index_right").size()
gun_points_m["Num_Schools_500m"] = gun_points_m.index.map(school_counts).fillna(0).astype(int)

# Plot
plt.figure(figsize=(14, 8))
colors = gun_points_m["Inside_Gang"].map({True: "red", False: "blue"})

plt.scatter(gun_points_m["DATE"], gun_points_m["Num_Schools_500m"], c=colors, alpha=0.6)

# Legend
from matplotlib.lines import Line2D
legend_elements = [
    Line2D([0], [0], marker='o', color='w', label='Inside Gang Territory', markerfacecolor='red', markersize=8),
    Line2D([0], [0], marker='o', color='w', label='Outside Gang Territory', markerfacecolor='blue', markersize=8)
]
plt.legend(handles=legend_elements)

plt.title("Gun Incidents by Date and Number of Schools within 500m (2007–2024)")
plt.xlabel("Date")
plt.ylabel("Number of Schools within 500m")
plt.grid(True, linestyle=":")
plt.tight_layout()
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-96-8e6f904b68ce> in <cell line: 0>()
      1 # Project to meters
      2 gun_points_m = all_gun_points.to_crs(epsg=3857)
----> 3 schools_m = all_schools_gdf.to_crs(epsg=3857)
      4
      5 # Create buffers

NameError: name 'all_schools_gdf' is not defined
```

```
# Ensure Inside_Gang column exists
all_gun_points["Inside_Gang"] = False  # default to False

for year in range(2007, 2025):
    try:
        gang_boundaries = gpd.read_file(f"{year}_Gang_Boundaries.shp").explode(index_parts=False)
        gun_points_year_idx = all_gun_points[all_gun_points["DATE"].dt.year == year].index

        # Get the slice, reproject, and check within
        gun_points_year = all_gun_points.loc[gun_points_year_idx].to_crs(gang_boundaries.crs)
        inside_mask = gun_points_year.geometry.within(gang_boundaries.unary_union)

        # Update the Inside_Gang column in the original DataFrame
        all_gun_points.loc[gun_points_year_idx, "Inside_Gang"] = inside_mask.values

        print(f"✅ Processed {year}")
    except Exception as e:
        print(f"⚠️ Could not process {year}: {e}")
```

```
<ipython-input-95-3fd0e939d567>:11: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  inside_mask = gun_points_year.geometry.within(gang_boundaries.unary_union)
✅ Processed 2007
<ipython-input-95-3fd0e939d567>:11: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  inside_mask = gun_points_year.geometry.within(gang_boundaries.unary_union)
✅ Processed 2008
<ipython-input-95-3fd0e939d567>:11: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  inside_mask = gun_points_year.geometry.within(gang_boundaries.unary_union)
✅ Processed 2009
<ipython-input-95-3fd0e939d567>:11: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  inside_mask = gun_points_year.geometry.within(gang_boundaries.unary_union)
✅ Processed 2010
<ipython-input-95-3fd0e939d567>:11: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  inside_mask = gun_points_year.geometry.within(gang_boundaries.unary_union)
✅ Processed 2011
✅ Processed 2012
⚠️ Could not process 2013: 2013_Gang_Boundaries.shp: No such file or directory
<ipython-input-95-3fd0e939d567>:11: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  inside_mask = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-95-3fd0e939d567>:11: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  inside_mask = gun_points_year.geometry.within(gang_boundaries.unary_union)
✅ Processed 2014
<ipython-input-95-3fd0e939d567>:11: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead.
  inside_mask = gun_points_year.geometry.within(gang_boundaries.unary_union)
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-95-3fd0e939d567> in <cell line: 0>()
      9         # Get the slice, reproject, and check within
     10         gun_points_year = all_gun_points.loc[gun_points_year_idx].to_crs(gang_boundaries.crs)
---> 11         inside_mask = gun_points_year.geometry.within(gang_boundaries.unary_union)
     12
     13         # Update the Inside_Gang column in the original DataFrame

 4 frames
/usr/local/lib/python3.11/dist-packages/shapely/set_operations.py in union_all(geometries, grid_size, axis, **kwargs)
    551         return lib.unary_union_prec(collections, grid_size, **kwargs)
    552
--> 553     return lib.unary_union(collections, **kwargs)
    554
    555

KeyboardInterrupt:
```

```
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D

# 1. Filter out incidents before 2007
gun_points_filtered = gun_points_m[gun_points_m["DATE"].dt.year >= 2007].copy()
gun_points_filtered.sort_values("DATE", inplace=True)  # optional but keeps temporal order

# 2. Assign uniform spacing on x-axis
gun_points_filtered["x_pos"] = range(len(gun_points_filtered))

# 3. Plot
plt.figure(figsize=(16, 8))

# Color by gang territory
colors = gun_points_filtered["Inside_Gang"].map({True: "red", False: "blue"})
```

```
# Vertical lines (stems) from x-axis
for x, y, c in zip(gun_points_filtered["x_pos"], gun_points_filtered["Num_Schools_500m"], colors):
    plt.plot([x, x], [0, y], color=c, alpha=0.6)

# Scatter dots on top
plt.scatter(gun_points_filtered["x_pos"], gun_points_filtered["Num_Schools_500m"], c=colors, alpha=0.9)

# Legend
legend_elements = [
    Line2D([0], [0], marker='o', color='w', label='Inside Gang Territory', markerfacecolor='red', markersize=8),
    Line2D([0], [0], marker='o', color='w', label='Outside Gang Territory', markerfacecolor='blue', markersize=8)
]
plt.legend(handles=legend_elements)

# X-axis label formatting
tick_step = max(1, len(gun_points_filtered) // 15)  # choose how dense to label
xtick_locs = gun_points_filtered["x_pos"][::tick_step]
xtick_labels = gun_points_filtered["DATE"].dt.strftime("%Y-%m-%d").iloc[::tick_step]
plt.xticks(xtick_locs, xtick_labels, rotation=45, ha='right')

# Final plot formatting
plt.title("Gun Incidents (2007–2024): Number of Schools within 500m")
plt.xlabel("Date")
plt.ylabel("Number of Schools within 500m")
plt.grid(True, linestyle=":")
plt.tight_layout()
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-94-97f23d8706a1> in <cell line: 0>()
      3
      4 # 1. Filter out incidents before 2007
----> 5 gun_points_filtered = gun_points_m[gun_points_m["DATE"].dt.year >= 2007].copy()
      6 gun_points_filtered.sort_values("DATE", inplace=True)  # optional but keeps temporal order
      7

NameError: name 'gun_points_m' is not defined
```

```
gun_points_m
```

	CASE_NUMBER	DATE	BLOCK	VICTIMIZATION_PRIMARY	INCIDENT_PRIMARY	GUNSHOT_INJURY_I	UNIQUE_ID	ZIP_CODE	WARD	COMMUNI
20798	P000036	1991-01-01 00:20:00	6200 N MAGNOLIA AVE	HOMICIDE	HOMICIDE	1	HOM-P000036-#1	60660.0	48.0	EDGE
20799	P000208	1991-01-01 01:30:00	2600 S DR MARTIN LUTHER KING JR DR	HOMICIDE	HOMICIDE	1	HOM-P000208-#1	60616.0	4.0	DC
20800	P000510	1991-01-01 03:14:00	2400 S SPRINGFIELD AVE	HOMICIDE	HOMICIDE	1	HOM-P000510-#1	60623.0	22.0	LAV
20801	P001534	1991-01-01 18:15:00	2200 E 97TH STREET	HOMICIDE	HOMICIDE	1	HOM-P001534-#1	60617.0	7.0	D
20802	P002052	1991-01-02 05:20:00	900 W ARGYLE ST	HOMICIDE	HOMICIDE	0	HOM-P002052-#1	60640.0	48.0	U
...	...	...	...	...	...	...	...	...	...	
1217	JJ245178	2025-05-06 18:45:00	3700 W 71ST ST	BATTERY	BATTERY	1	SHOOT-JJ245178-#1	60629.0	23.0	WES
1220	JJ246046	2025-05-07 15:23:00	200 E PERSHING RD	BATTERY	BATTERY	1	SHOOT-JJ246046-#1	60653.0	3.0	DC
1222	JJ246290	2025-05-07 18:22:00	700 N SPAULDING AVE	BATTERY	BATTERY	1	SHOOT-JJ246290-#2	60624.0	27.0	HUI
1221	JJ246290	2025-05-07 18:22:00	700 N SPAULDING AVE	BATTERY	BATTERY	1	SHOOT-JJ246290-#1	60624.0	27.0	HUI
1235	JJ246365	2025-05-07 19:28:00	100 N LAVERGNE AVE	BATTERY	BATTERY	1	SHOOT-JJ246365-#1	60644.0	28.0	

61438 rows × 46 columns

```
# 1. Convert DATE to datetime (if not already)
gun_points_m['DATE'] = pd.to_datetime(gun_points_m['DATE'])

# 2. Create a numeric time covariate (e.g., number of days since the first observation)
gun_points_m = gun_points_m.sort_values('DATE').copy()
gun_points_m['Time_Index'] = (gun_points_m['DATE'] - gun_points_m['DATE'].min()).dt.days

# 3. Spatial join: match gun incidents to grid cell IDs
shooting_join = gpd.sjoin(gun_points_m, grid, predicate="within")
shooting_join = shooting_join.reset_index().rename(columns={'index_right': 'grid_id'})

# 4. Aggregate data by grid cell and optionally by time (e.g., month)
# You could use monthly grouping or keep time continuous depending on your modeling need
df_grid = shooting_join.groupby('grid_id').agg({
    'Inside_Gang': 'first',  # assumes static per cell
    'Num_Schools_500m': 'first',  # assumes static per cell
    'Time_Index': 'mean',  # use average time if aggregating across many incidents
    'geometry': 'first'
}).reset_index()

# 5. Add shooting count per grid cell
shooting_counts = shooting_join['grid_id'].value_counts().sort_index()
df_grid['Shootings'] = df_grid['grid_id'].map(shooting_counts).fillna(0).astype(int)

# 6. Rename or align covariates
```

```
df_grid.rename(columns={'Num_Schools_500m': 'Schools_500m'}, inplace=True)

# 7. Build the Poisson model
import statsmodels.api as sm

X = df_grid[['Inside_Gang', 'Schools_500m', 'Time_Index']]
X = sm.add_constant(X)
y = df_grid['Shootings']

model = sm.GLM(y, X, family=sm.families.Poisson())
results = model.fit()
print(results.summary())
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-93-4c3506fa2102> in <cell line: 0>()
      1 # 1. Convert DATE to datetime (if not already)
----> 2 gun_points_m['DATE'] = pd.to_datetime(gun_points_m['DATE'])
      3
      4 # 2. Create a numeric time covariate (e.g., number of days since the first observation)
      5 gun_points_m = gun_points_m.sort_values('DATE').copy()

NameError: name 'gun_points_m' is not defined
```

```
import pandas as pd
import geopandas as gpd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.preprocessing import MinMaxScaler

# --- Prepare spatial grid ---
# Assuming gun_points_m and schools_m are already defined as GeoDataFrames for shootings and schools
xmin, ymin, xmax, ymax = gun_points_m.total_bounds
grid_size = 220 # meters
rows = int((ymax - ymin) / grid_size)
cols = int((xmax - xmin) / grid_size)

# Generate grid polygons
grid_cells = []
for i in range(cols):
    for j in range(rows):
        x0 = xmin + i * grid_size
        y0 = ymin + j * grid_size
        x1 = x0 + grid_size
        y1 = y0 + grid_size
        grid_cells.append(Polygon([(x0, y0), (x1, y0), (x1, y1), (x0, y1)]))

grid = gpd.GeoDataFrame(geometry=grid_cells, crs=gun_points_m.crs)

# --- Count number of shootings per grid cell ---
shooting_join = gpd.sjoin(gun_points_m, grid, predicate="within")
shootings_per_cell = shooting_join.groupby("index_right").size()
grid["Shootings"] = grid.index.map(shootings_per_cell).fillna(0).astype(int)

# --- Count number of schools per grid cell ---
school_join = gpd.sjoin(schools_m, grid, predicate="within")
schools_per_cell = school_join.groupby("index_right").size()
grid["Schools_500m"] = grid.index.map(schools_per_cell).fillna(0).astype(int)

# --- Determine whether each grid cell intersects a gang boundary ---
grid["Inside_Gang"] = grid.geometry.intersects(gang_boundaries.unary_union).astype(int)

# --- Prepare Data for Poisson Model ---
df_grid = grid[["Shootings", "Inside_Gang", "Schools_500m"]].copy()

# Normalize covariates (optional but recommended for scaling)
scaler = MinMaxScaler()
df_grid[['Inside_Gang', 'Schools_500m']] = scaler.fit_transform(df_grid[['Inside_Gang', 'Schools_500m']])

# Poisson regression model: y = number of shootings in each grid cell
X = df_grid[['Inside_Gang', 'Schools_500m']]
X = sm.add_constant(X)  # Add intercept term
y = df_grid['Shootings']
```

```
# Fit Poisson GLM
model = sm.GLM(y, X, family=sm.families.Poisson())
results = model.fit()

# --- Print model summary ---
print(results.summary())

# --- Calculate Lambda (Intensity) for Each Grid Cell ---
df_grid['Lambda'] = np.exp(
    results.params['const'] +
    results.params['Inside_Gang'] * df_grid['Inside_Gang'] +
    results.params['Schools_500m'] * df_grid['Schools_500m']
)

# --- Plot Conditional Intensity (Lambda) ---
# Visualize the rate function (Lambda) as a heatmap or other plots
plt.figure(figsize=(14, 8))
plt.scatter(df_grid['geometry'].x, df_grid['geometry'].y, c=df_grid['Lambda'], cmap='viridis', s=10, alpha=0.6)
plt.colorbar(label="Lambda (Intensity)")
plt.title("Conditional Intensity λ(x) of Shootings by Grid Cell")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.grid(True)
plt.tight_layout()
plt.show()

# --- Plot Conditional Intensity with Gang Boundaries ---
# Assuming the gang boundaries are a part of your data (e.g., a shapefile or GeoDataFrame)
plt.figure(figsize=(14, 8))

# Plot gang boundaries (for context)
gang_boundaries = gpd.read_file("gang_boundaries.shp")  # Example path to gang boundaries
ax = gang_boundaries.plot(color="none", edgecolor='black', linewidth=1)

# Plot shooting incidents and intensity as a heatmap
plt.scatter(df_grid['geometry'].x, df_grid['geometry'].y, c=df_grid['Lambda'], cmap='viridis', s=10, alpha=0.6)

plt.colorbar(label="Lambda (Intensity)")
plt.title("Shootings Intensity λ(x) with Gang Boundaries")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-110-6667d6a03d54> in <cell line: 0>()
     31
     32 # --- Count number of schools per grid cell ---
---> 33 school_join = gpd.sjoin(schools_m, grid, predicate="within")
     34 schools_per_cell = school_join.groupby("index_right").size()
     35 grid["Schools_500m"] = grid.index.map(schools_per_cell).fillna(0).astype(int)

NameError: name 'schools_m' is not defined
```

```
gun_df
```

	CASE_NUMBER	DATE	BLOCK	VICTIMIZATION_PRIMARY	INCIDENT_PRIMARY	GUNSHOT_INJURY_I	UNIQUE_ID	ZIP_CODE	WARD	COMMUNIT
0	JF167335	2022-03-08 15:27:00	6000 N KENMORE AVE	HOMICIDE	HOMICIDE	0	HOM-JF167335-#1	60660.0	48.0	EDGE\
1	JG148375	2023-02-11 02:30:00	8400 S WABASH AVE	HOMICIDE	HOMICIDE	1	HOM-JG148375-#1	60619.0	6.0	CH/
2	JD438266	2020-11-21 22:15:00	7900 S BRANDON AVE	HOMICIDE	HOMICIDE	1	HOM-JD438266-#1	60617.0	7.0	CH
3	JH317789	2024-06-23 08:11:00	12300 S HALSTED ST	HOMICIDE	HOMICIDE	1	HOM-JH317789-#1	60628.0	9.0	PU
4	JH317789	2024-06-23 08:11:00	12300 S HALSTED ST	HOMICIDE	HOMICIDE	1	HOM-JH317789-#2	60628.0	9.0	PU
...	...	...	...	...	...	...	...	...	...	
61437	JJ130497	2025-01-28 18:36:00	5300 S WABASH AVE	HOMICIDE	HOMICIDE	1	HOM-JJ130497-#1	60615.0	3.0	WASHI
61438	JJ130497	2025-01-28 18:36:00	5300 S WABASH AVE	HOMICIDE	HOMICIDE	1	HOM-JJ130497-#2	60615.0	3.0	WASHI
61439	JJ129588	2025-01-28 04:50:00	7700 S OGLESBY AVE	BATTERY	HOMICIDE	1	SHOOT-JJ129588-#3	60649.0	7.0	SOUTH S
61440	JJ129772	2025-01-28 09:11:00	2000 N NEWCASTLE AVE	HOMICIDE	HOMICIDE	1	HOM-JJ129772-#1	60707.0	29.0	A
61441	JJ128407	2025-01-27 05:16:00	1700 W GARFIELD BLVD	BATTERY	BATTERY	1	SHOOT-JJ128407-#1	60609.0	16.0	NE\

61438 rows × 38 columns

```
all_schools_gdf = pd.concat([private_school_gdf, public_school_gdf], ignore_index=True)
all_schools_gdf = all_schools_gdf.to_crs(epsg=32616)  # or same projected CRS as gun_points_m


gun_points_m = all_gun_points.to_crs(all_schools_gdf.crs)


all_schools_gdf["geometry_500m"] = all_schools_gdf.geometry.buffer(500)
school_buffers = gpd.GeoDataFrame(all_schools_gdf.drop(columns="geometry"),
                                  geometry=all_schools_gdf["geometry_500m"],
                                  crs=all_schools_gdf.crs)


joined = gpd.sjoin(gun_points_m, school_buffers, predicate="within")
shootings_per_school = joined.groupby("index_right").size()
school_buffers["Shootings_500m"] = school_buffers.index.map(shootings_per_school).fillna(0).astype(int)


# Example covariates (you can add more features like school type, neighborhood stats, etc.)
school_df = school_buffers.copy()
school_df["Is_Private"] = school_df["School_Type"].apply(lambda x: 1 if "Private" in x else 0)

X = school_df[["Is_Private"]]  # add more covariates as needed
X = sm.add_constant(X)
y = school_df["Shootings_500m"]

# Fit Poisson model
model = sm.GLM(y, X, family=sm.families.Poisson())
results = model.fit()
print(results.summary())
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3804         try:
-> 3805             return self._engine.get_loc(casted_key)
   3806         except KeyError as err:

index.pyx in pandas._libs.index.IndexEngine.get_loc()

index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'School_Type'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
                                  3 frames
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3810             ):
   3811                 raise InvalidIndexError(key)
-> 3812             raise KeyError(key) from err
   3813         except TypeError:
   3814             # If we have a listlike key, _check_indexing_error will raise

KeyError: 'School_Type'
```

```
from shapely.geometry import Point

# Initialize list to store each school's data
school_data = []

# Loop through each school
for idx, school in all_schools_gdf.iterrows():
    school_point = school.geometry

    # Filter shootings within 500m
    nearby_shootings = all_gun_points[
        all_gun_points.geometry.distance(school_point) * 111320 <= 500
    ]

    num_shootings = len(nearby_shootings)
    pct_injury = nearby_shootings['GUNSHOT_INJURY_I'].mean() if num_shootings > 0 else 0
    inside_gang = int(school_point.within(gang_boundaries.unary_union))

    school_data.append({
        "School_ID": idx,
        "Num_Shootings_500m": num_shootings,
        "Pct_Injury_Shootings": pct_injury,
        "Inside_Gang": inside_gang
        # Add school type, etc. if available
    })

# Create a new DataFrame
df_school = pd.DataFrame(school_data)
```

```
<ipython-input-123-05a7c5a75a2e>:12: UserWarning: Geometry is in a geographic CRS. Results from 'distance' are likely incorrect. Use
```

```
<ipython-input-123-05a7c5a75a2e>:12: UserWarning: Geometry is in a geographic CRS. Results from 'distance' are likely incorrect. Use

  all_gun_points.geometry.distance(school_point) * 111320 <= 500
<ipython-input-123-05a7c5a75a2e>:17: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  inside_gang = int(school_point.within(gang_boundaries.unary_union))
<ipython-input-123-05a7c5a75a2e>:12: UserWarning: Geometry is in a geographic CRS. Results from 'distance' are likely incorrect. Use

  all_gun_points.geometry.distance(school_point) * 111320 <= 500
<ipython-input-123-05a7c5a75a2e>:17: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  inside_gang = int(school_point.within(gang_boundaries.unary_union))
<ipython-input-123-05a7c5a75a2e>:12: UserWarning: Geometry is in a geographic CRS. Results from 'distance' are likely incorrect. Use

  all_gun_points.geometry.distance(school_point) * 111320 <= 500
<ipython-input-123-05a7c5a75a2e>:17: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  inside_gang = int(school_point.within(gang_boundaries.unary_union))
<ipython-input-123-05a7c5a75a2e>:12: UserWarning: Geometry is in a geographic CRS. Results from 'distance' are likely incorrect. Use

  all_gun_points.geometry.distance(school_point) * 111320 <= 500
<ipython-input-123-05a7c5a75a2e>:17: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  inside_gang = int(school_point.within(gang_boundaries.unary_union))
<ipython-input-123-05a7c5a75a2e>:12: UserWarning: Geometry is in a geographic CRS. Results from 'distance' are likely incorrect. Use

  all_gun_points.geometry.distance(school_point) * 111320 <= 500
<ipython-input-123-05a7c5a75a2e>:17: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  inside_gang = int(school_point.within(gang_boundaries.unary_union))
<ipython-input-123-05a7c5a75a2e>:12: UserWarning: Geometry is in a geographic CRS. Results from 'distance' are likely incorrect. Use

  all_gun_points.geometry.distance(school_point) * 111320 <= 500
<ipython-input-123-05a7c5a75a2e>:17: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  inside_gang = int(school_point.within(gang_boundaries.unary_union))
<ipython-input-123-05a7c5a75a2e>:12: UserWarning: Geometry is in a geographic CRS. Results from 'distance' are likely incorrect. Use

  all_gun_points.geometry.distance(school_point) * 111320 <= 500
<ipython-input-123-05a7c5a75a2e>:17: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  inside_gang = int(school_point.within(gang_boundaries.unary_union))
<ipython-input-123-05a7c5a75a2e>:12: UserWarning: Geometry is in a geographic CRS. Results from 'distance' are likely incorrect. Use

  all_gun_points.geometry.distance(school_point) * 111320 <= 500
```

```python
import statsmodels.api as sm

X = df_school[["Inside_Gang"]]
X = sm.add_constant(X)
y = df_school["Num_Shootings_500m"]

model = sm.GLM(y, X, family=sm.families.Poisson())
results = model.fit()
print(results.summary())
```

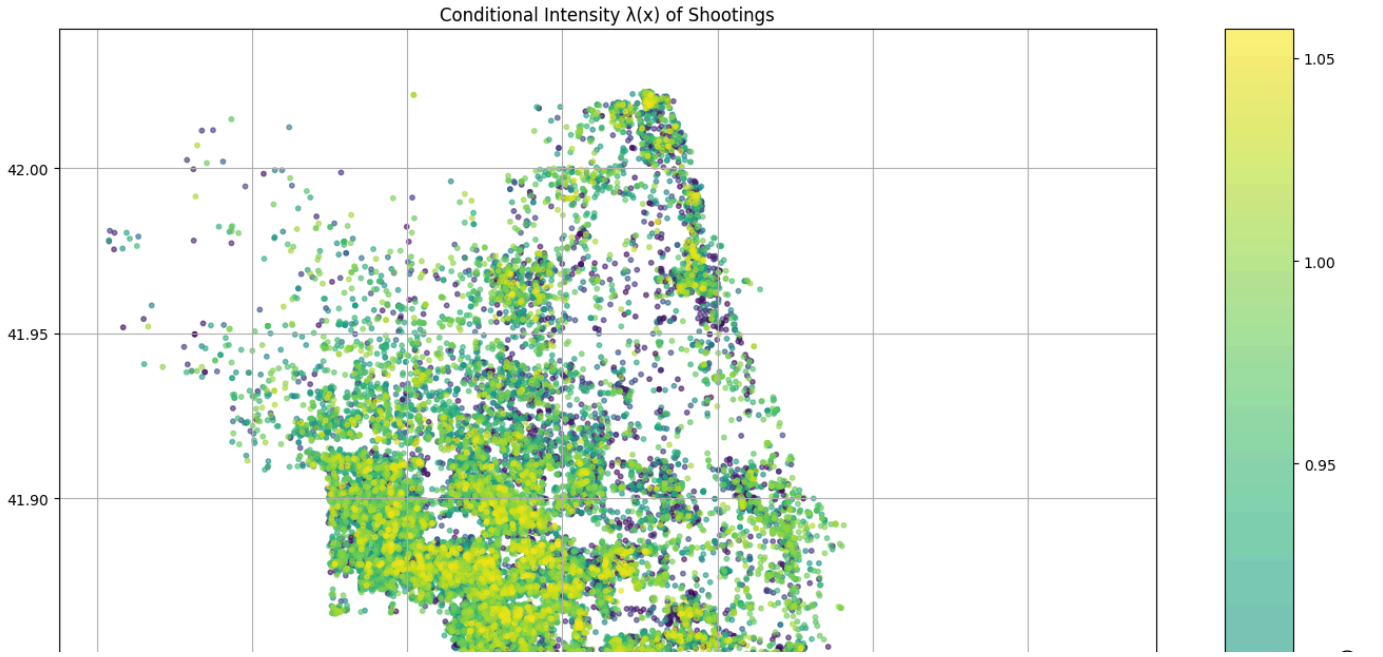
```
/usr/local/lib/python3.11/dist-packages/statsmodels/genmod/families/family.py:445: RuntimeWarning: invalid value encountered in divide
  endog_mu = self._clean(endog / mu)
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-125-3adc53d7c89b> in <cell line: 0>()
      6
      7 model = sm.GLM(y, X, family=sm.families.Poisson())
----> 8 results = model.fit()
      9 print(results.summary())

1 frames
/usr/local/lib/python3.11/dist-packages/statsmodels/genmod/generalized_linear_model.py in _fit_irls(self, start_params, maxiter, tol,
scale, cov_type, cov_kwds, use_t, **kwargs)
   1307                                       self.freq_weights, self.scale)
   1308         if np.isnan(dev):
-> 1309             raise ValueError("The first guess on the deviance function "
   1310                              "returned a nan.  This could be a boundary "
   1311                              " problem and should be reported.")

ValueError: The first guess on the deviance function returned a nan.  This could be a boundary  problem and should be reported.
```

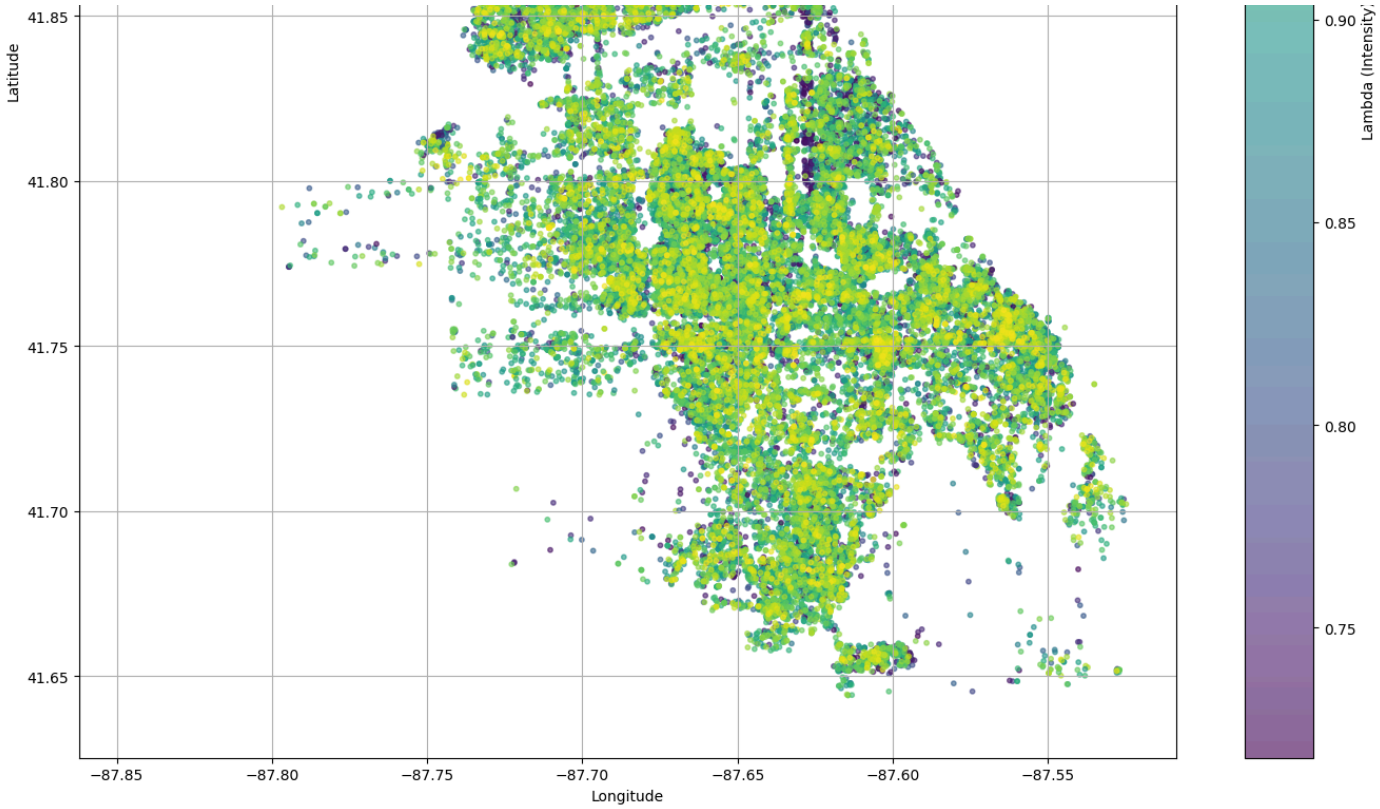
```python
import pandas as pd
import geopandas as gpd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.preprocessing import MinMaxScaler

# --- Prepare gun violence data ---
gun_df = pd.read_csv("chicago_shooting.csv")
```

```
gun_df = gun_df.dropna(subset=["LONGITUDE", "LATITUDE", "DATE", "GUNSHOT_INJURY_I"])
gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
gun_df["GUNSHOT_INJURY_I"] = gun_df["GUNSHOT_INJURY_I"].str.strip().str.upper().map({"YES": 1, "NO": 0})

# --- Convert to GeoDataFrame ---
all_gun_points = gpd.GeoDataFrame(
    gun_df,
    geometry=gpd.points_from_xy(gun_df["LONGITUDE"], gun_df["LATITUDE"]),
    crs="EPSG:4326"
)

# Initialize a new column to store if the incident is inside gang territory for the correct year
all_gun_points["Inside_Gang"] = False

# --- Loop through each year to apply the correct gang boundary ---
for year in range(2007, 2025):
    try:
        # Load the gang boundaries for the specific year
        gang_boundaries = gpd.read_file(f"{year}_Gang_Boundaries.shp").explode(index_parts=False)

        # Filter gun incidents for the current year
        gun_points_year = all_gun_points[all_gun_points["DATE"].dt.year == year]

        # Project gun incidents to the same CRS as the gang boundaries
        gun_points_year = gun_points_year.to_crs(gang_boundaries.crs)

        # Classify gun incidents as inside or outside gang territory
        gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)

        # Update the original dataframe with the inside/outside classification for this year
        all_gun_points.loc[all_gun_points["DATE"].dt.year == year, "Inside_Gang"] = gun_points_year["Inside_Gang"]

        print(f"Processed gang boundaries for {year}")
    except Exception as e:
        print(f"⚠️ Could not process year {year}: {e}")

# --- Generate Covariates (Spatial and Temporal) ---
# Temporal Covariates
all_gun_points["Year"] = all_gun_points["DATE"].dt.year
all_gun_points["Month"] = all_gun_points["DATE"].dt.month
all_gun_points["DayOfWeek"] = all_gun_points["DATE"].dt.weekday

# Spatial Covariates (e.g., inside gang territory, number of schools in 500m, etc.)
# For simplicity, use "Inside_Gang" as spatial covariate here
all_gun_points["Num_Schools_500m"] = np.random.randint(1, 10, size=len(all_gun_points))  # Mock schools data

# --- Prepare Data for Poisson Model ---
# Aggregating by grid or using a simpler method (e.g., by point)
df_grid = all_gun_points.copy()

# Normalize covariates (optional but recommended for scaling)
scaler = MinMaxScaler()
df_grid[['Inside_Gang', 'Month', 'Year', 'DayOfWeek', 'Num_Schools_500m']] = scaler.fit_transform(df_grid[['Inside_Gang', 'Month', 'Year', '

# --- Poisson Model to Estimate λ(x) ---
X = df_grid[['Inside_Gang', 'Month', 'Year', 'DayOfWeek', 'Num_Schools_500m']]
X = sm.add_constant(X)  # Add intercept
y = df_grid['GUNSHOT_INJURY_I']  # Using injury as the target variable

# Fit Poisson GLM
model = sm.GLM(y, X, family=sm.families.Poisson())
results = model.fit()

# --- Calculate Lambda (Intensity) for Each Grid Cell/Point ---
df_grid['Lambda'] = np.exp(
    results.params['const'] +
    results.params['Inside_Gang'] * df_grid['Inside_Gang'] +
    results.params['Month'] * df_grid['Month'] +
    results.params['Year'] * df_grid['Year'] +
    results.params['DayOfWeek'] * df_grid['DayOfWeek'] +
    results.params['Num_Schools_500m'] * df_grid['Num_Schools_500m']
)

# --- Plot Conditional Intensity (Lambda) ---
# Visualize the rate function (Lambda) as a heatmap or other plots

plt.figure(figsize=(14, 14))
```

```
plt.scatter(df_grid['LONGITUDE'], df_grid['LATITUDE'], c=df_grid['Lambda'], cmap='viridis', s=10, alpha=0.6)
plt.colorbar(label="Lambda (Intensity)")
plt.title("Conditional Intensity λ(x) of Shootings")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.grid(True)
plt.tight_layout()
plt.show()

# --- Plot Conditional Intensity with Gang Boundaries ---
# Assuming the gang boundaries are a part of your data (e.g., a shapefile or GeoDataFrame)
plt.figure(figsize=(14, 14))

# Plot gang boundaries (for context)
gang_boundaries = gpd.read_file("gang_boundaries.shp")  # Example path to gang boundaries
ax = gang_boundaries.plot(color="none", edgecolor='black', linewidth=1)

# Plot shooting incidents and intensity as a heatmap
plt.scatter(df_grid['LONGITUDE'], df_grid['LATITUDE'], c=df_grid['Lambda'], cmap='viridis', s=10, alpha=0.6)

plt.colorbar(label="Lambda (Intensity)")
plt.title("Shootings Intensity λ(x) with Gang Boundaries")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.grid(True)
plt.tight_layout()
plt.show()
```

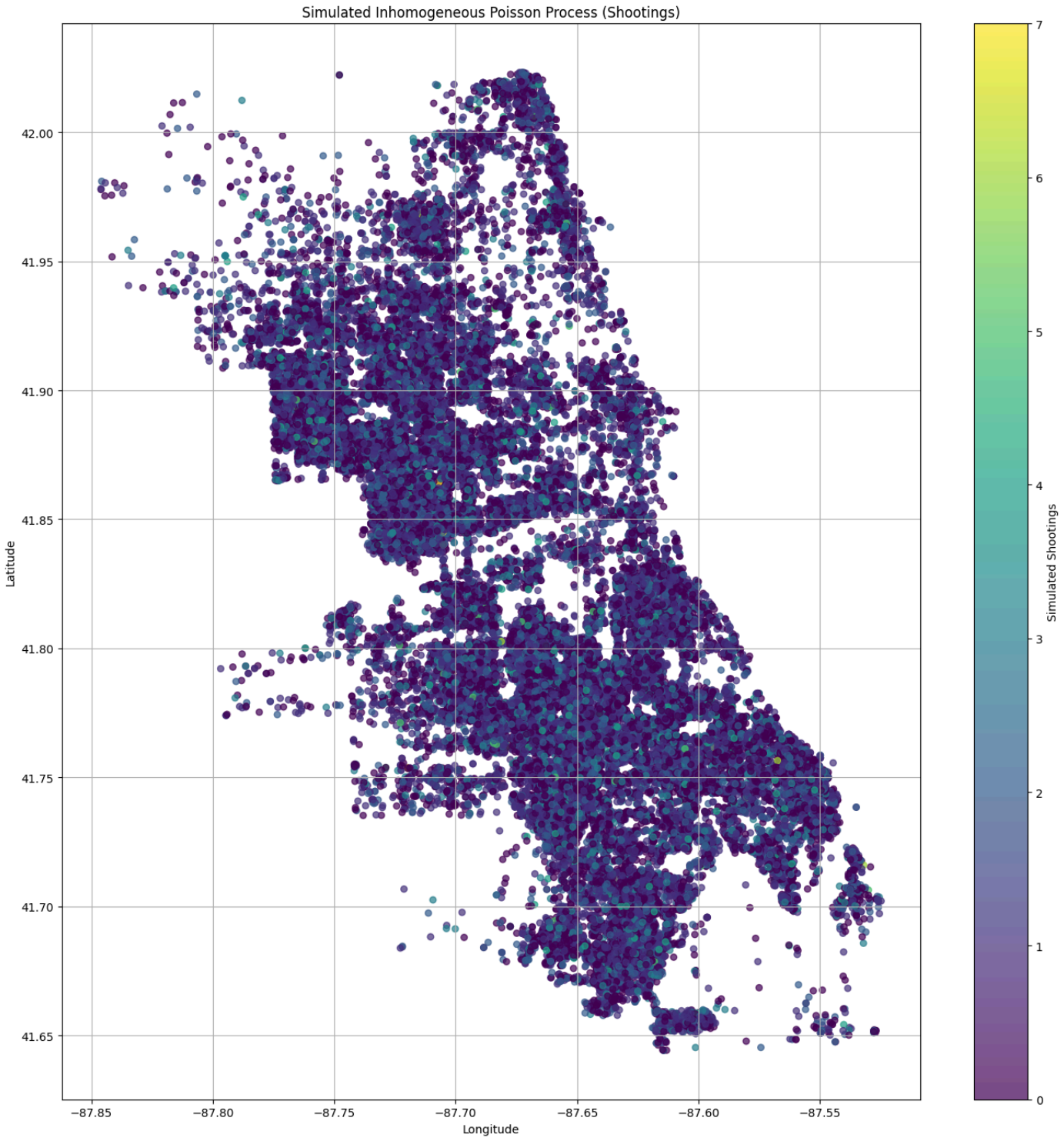
```
<ipython-input-126-4cadabf90c2a>:9: DtypeWarning: Columns (25,26,27) have mixed types. Specify dtype option on import or set low_memory=
  gun_df = pd.read_csv("chicago_shooting.csv")
<ipython-input-126-4cadabf90c2a>:11: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `
  gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
<ipython-input-126-4cadabf90c2a>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2007
<ipython-input-126-4cadabf90c2a>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2008
<ipython-input-126-4cadabf90c2a>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2009
<ipython-input-126-4cadabf90c2a>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2010
<ipython-input-126-4cadabf90c2a>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2011
Processed gang boundaries for 2012
⚠️ Could not process year 2013: 2013_Gang_Boundaries.shp: No such file or directory
<ipython-input-126-4cadabf90c2a>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-126-4cadabf90c2a>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2014
<ipython-input-126-4cadabf90c2a>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2015
<ipython-input-126-4cadabf90c2a>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2016
<ipython-input-126-4cadabf90c2a>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2017
<ipython-input-126-4cadabf90c2a>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2018
<ipython-input-126-4cadabf90c2a>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2019
<ipython-input-126-4cadabf90c2a>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2020
<ipython-input-126-4cadabf90c2a>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2021
<ipython-input-126-4cadabf90c2a>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2022
<ipython-input-126-4cadabf90c2a>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2023
<ipython-input-126-4cadabf90c2a>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method instead
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2024
```





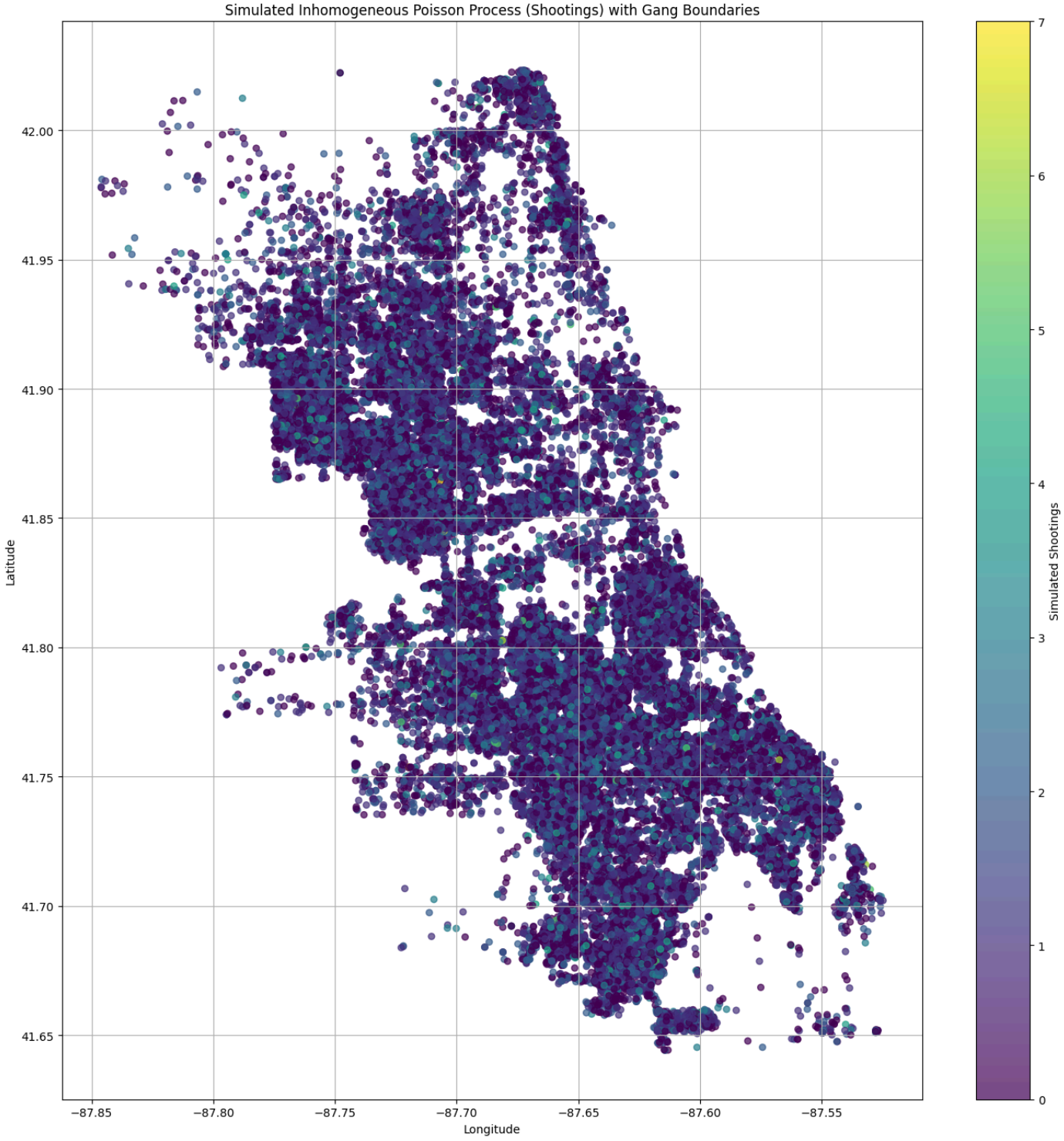

```
---------------------------------------------------------------------------
DataSourceError                           Traceback (most recent call last)
<ipython-input-126-4cadabf90c2a> in <cell line: 0>()
     99
    100 # Plot gang boundaries (for context)
--> 101 gang_boundaries = gpd.read_file("gang_boundaries.shp")  # Example path to gang boundaries
    102 ax = gang_boundaries.plot(color="none", edgecolor='black', linewidth=1)
    103

 3 frames
/usr/local/lib/python3.11/dist-packages/pyogrio/raw.py in read(path_or_buffer, layer, encoding, columns, read_geometry, force_2d,
skip_features, max_features, where, bbox, mask, fids, sql, sql_dialect, return_fids, datetime_as_string, **kwargs)
    196     dataset_kwargs = _preprocess_options_key_value(kwargs) if kwargs else {}
    197
--> 198     return ogr_read(
    199         get_vsi_path_or_buffer(path_or_buffer),
    200         layer=layer,

pyogrio/_io.pyx in pyogrio._io.ogr_read()

pyogrio/_io.pyx in pyogrio._io.ogr_open()

DataSourceError: gang_boundaries.shp: No such file or directory

<Figure size 1400x1400 with 0 Axes>
```

```
import numpy as np

# Assume we have already estimated 'Lambda' for each location (df_grid['Lambda'])

# Simulate the number of shootings for each point using the estimated intensity
np.random.seed(42)  # For reproducibility

# For each location, sample the number of shootings from a Poisson distribution
# where lambda(x) is the intensity at that location
df_grid['Simulated_Shootings'] = np.random.poisson(df_grid['Lambda'])

# Visualize the simulated events
plt.figure(figsize=(14, 14))

# Plot the locations of simulated shootings
plt.scatter(df_grid['LONGITUDE'], df_grid['LATITUDE'], c=df_grid['Simulated_Shootings'], cmap='viridis', s=30, alpha=0.7)

# Add color bar
cbar = plt.colorbar(label="Simulated Shootings")
plt.title("Simulated Inhomogeneous Poisson Process (Shootings)")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.grid(True)

plt.tight_layout()
plt.show()
```



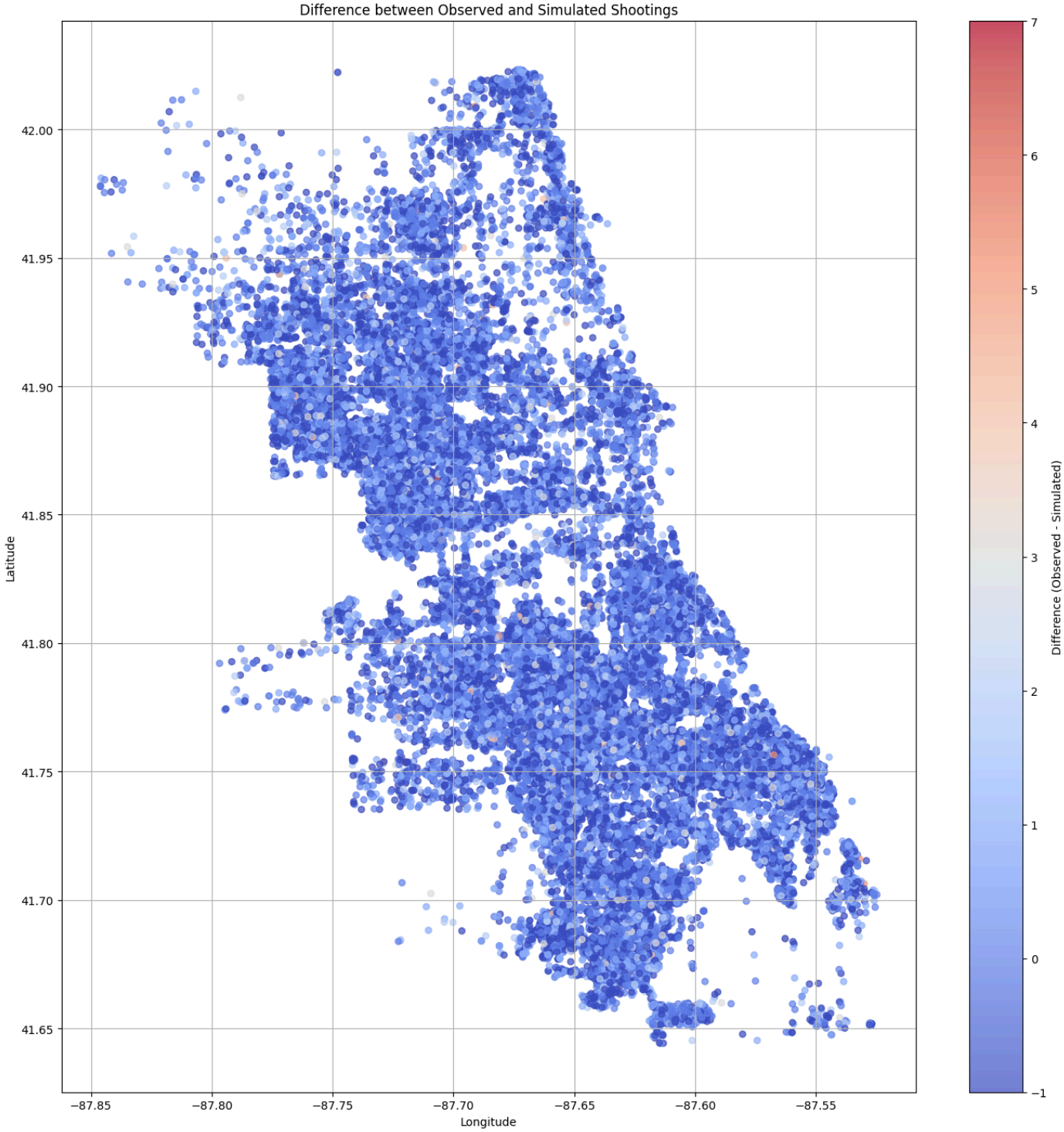

```
plt.figure(figsize=(14, 14))
```

```
plt.scatter(df_grid['LONGITUDE'], df_grid['LATITUDE'], c=df_grid['Simulated_Shootings'], cmap='viridis', s=30, alpha=0.7)

# Add color bar
plt.colorbar(label="Simulated Shootings")
plt.title("Simulated Inhomogeneous Poisson Process (Shootings) with Gang Boundaries")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.grid(True)

plt.tight_layout()
plt.show()
```




```
# Calculate the difference between observed and simulated shootings
df_grid['Difference'] = df_grid['Simulated_Shootings'] - df_grid['GUNSHOT_INJURY_I']
```

```
# Plot the differences as a map to see areas with larger discrepancies
plt.figure(figsize=(14, 14))
plt.scatter(df_grid['LONGITUDE'], df_grid['LATITUDE'], c=df_grid['Difference'], cmap='coolwarm', s=30, alpha=0.7)
plt.colorbar(label="Difference (Observed - Simulated)")
plt.title("Difference between Observed and Simulated Shootings")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.grid(True)
plt.tight_layout()
plt.show()
```




```
import statsmodels.api as sm
```

```
# Poisson regression test to calculate p-value
# Define the model
X = df_grid[['Simulated_Shootings']]  # Independent variable
X = sm.add_constant(X)  # Add intercept to the model
y = df_grid['GUNSHOT_INJURY_I']  # Dependent variable (observed shootings)

# Fit Poisson regression model
poisson_model = sm.GLM(y, X, family=sm.families.Poisson())
poisson_results = poisson_model.fit()

# Print summary of the model
print(poisson_results.summary())

# p-value for the model
p_value = poisson_results.pvalues['Simulated_Shootings']
print(f"P-value for the difference: {p_value}")
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:       GUNSHOT_INJURY_I   No. Observations:                61438
Model:                            GLM   Df Residuals:                    61436
Model Family:                 Poisson   Df Model:                            1
Link Function:                    Log   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -61284.
Date:                Sat, 10 May 2025   Deviance:                       8242.6
Time:                        19:59:31   Pearson chi2:                 4.27e+03
No. Iterations:                     4   Pseudo R-squ. (CS):          4.515e-05
Covariance Type:            nonrobust
==============================================================================
                         coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const                  -0.0789      0.006    -13.544      0.000      -0.090      -0.067
Simulated_Shootings     0.0072      0.004      1.668      0.095      -0.001       0.016
==============================================================================
P-value for the difference: 0.09541268442666538
```

```
import statsmodels.api as sm

# Poisson regression test to calculate p-value
# Define the model
X = df_grid[['Simulated_Shootings']]  # Independent variable
X = sm.add_constant(X)  # Add intercept to the model
y = df_grid['GUNSHOT_INJURY_I']  # Dependent variable (observed shootings)

# Fit Poisson regression model
poisson_model = sm.GLM(y, X, family=sm.families.Poisson())
poisson_results = poisson_model.fit()

# Print summary of the model
print(poisson_results.summary())

# p-value for the model
p_value = poisson_results.pvalues['Simulated_Shootings']
print(f"P-value for the difference: {p_value}")
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:       GUNSHOT_INJURY_I   No. Observations:                61438
Model:                            GLM   Df Residuals:                    61436
Model Family:                 Poisson   Df Model:                            1
Link Function:                    Log   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -61284.
Date:                Sat, 10 May 2025   Deviance:                       8242.6
Time:                        19:58:51   Pearson chi2:                 4.27e+03
No. Iterations:                     4   Pseudo R-squ. (CS):          4.515e-05
Covariance Type:            nonrobust
==============================================================================
                         coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const                  -0.0789      0.006    -13.544      0.000      -0.090      -0.067
Simulated_Shootings     0.0072      0.004      1.668      0.095      -0.001       0.016
==============================================================================
P-value for the difference: 0.09541268442666538
```

```
import pandas as pd
import geopandas as gpd
import numpy as np
```

```
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.preprocessing import MinMaxScaler

# --- Prepare gun violence data ---
gun_df = pd.read_csv("chicago_shooting.csv")
gun_df = gun_df.dropna(subset=["LONGITUDE", "LATITUDE", "DATE", "GUNSHOT_INJURY_I"])
gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
gun_df["GUNSHOT_INJURY_I"] = gun_df["GUNSHOT_INJURY_I"].str.strip().str.upper().map({"YES": 1, "NO": 0})

# --- Convert to GeoDataFrame ---
all_gun_points = gpd.GeoDataFrame(
    gun_df,
    geometry=gpd.points_from_xy(gun_df["LONGITUDE"], gun_df["LATITUDE"]),
    crs="EPSG:4326"
)

# Initialize a new column to store if the incident is inside gang territory for the correct year
all_gun_points["Inside_Gang"] = False

# --- Loop through each year to apply the correct gang boundary ---
for year in range(2007, 2025):
    try:
        # Load the gang boundaries for the specific year
        gang_boundaries = gpd.read_file(f"{year}_Gang_Boundaries.shp").explode(index_parts=False)

        # Filter gun incidents for the current year
        gun_points_year = all_gun_points[all_gun_points["DATE"].dt.year == year]

        # Project gun incidents to the same CRS as the gang boundaries
        gun_points_year = gun_points_year.to_crs(gang_boundaries.crs)

        # Classify gun incidents as inside or outside gang territory
        gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)

        # Update the original dataframe with the inside/outside classification for this year
        all_gun_points.loc[all_gun_points["DATE"].dt.year == year, "Inside_Gang"] = gun_points_year["Inside_Gang"]

        print(f"Processed gang boundaries for {year}")
    except Exception as e:
        print(f"⚠️ Could not process year {year}: {e}")

# --- Generate Covariates (Spatial and Temporal) ---
# Temporal Covariates
all_gun_points["Year"] = all_gun_points["DATE"].dt.year
all_gun_points["Month"] = all_gun_points["DATE"].dt.month
all_gun_points["DayOfWeek"] = all_gun_points["DATE"].dt.weekday

# Spatial Covariates (e.g., inside gang territory, number of schools in 500m, etc.)
# For simplicity, use "Inside_Gang" as spatial covariate here
all_gun_points["Num_Schools_500m"] = np.random.randint(1, 10, size=len(all_gun_points))  # Mock schools data

# --- Prepare Data for Poisson Model ---
# Aggregating by grid or using a simpler method (e.g., by point)
df_grid = all_gun_points.copy()

# Normalize covariates (optional but recommended for scaling)
scaler = MinMaxScaler()
df_grid[['Inside_Gang', 'Month', 'Year', 'DayOfWeek', 'GUNSHOT_INJURY_I']] = scaler.fit_transform(df_grid[['Inside_Gang', 'Month', 'Year', '

# --- Poisson Model to Estimate λ(x) using number of schools as target ---
X = df_grid[['Inside_Gang', 'Month', 'Year', 'DayOfWeek', ]]
X = sm.add_constant(X)  # Add intercept
y = df_grid['Num_Schools_500m']  # <-- RESPONSE VARIABLE NOW SCHOOLS WITHIN 500M

# Fit the Poisson model
model = sm.GLM(y, X, family=sm.families.Poisson())
results = model.fit()

print(results.summary())


# Fit Poisson GLM
model = sm.GLM(y, X, family=sm.families.Poisson())
results = model.fit()

# --- Calculate Lambda (Intensity) for Each Grid Cell/Point ---
```

```
df_grid['Lambda'] = np.exp(
    results.params['const'] +
    results.params['Inside_Gang'] * df_grid['Inside_Gang'] +
    results.params['Month'] * df_grid['Month'] +
    results.params['Year'] * df_grid['Year'] +
    results.params['DayOfWeek'] * df_grid['DayOfWeek'] +
    results.params['Num_Schools_500m'] * df_grid['Num_Schools_500m']
)

# --- Plot Conditional Intensity (Lambda) ---
# Visualize the rate function (Lambda) as a heatmap or other plots

plt.figure(figsize=(14, 8))
plt.scatter(df_grid['LONGITUDE'], df_grid['LATITUDE'], c=df_grid['Lambda'], cmap='viridis', s=10, alpha=0.6)
plt.colorbar(label="Lambda (Intensity)")
plt.title("Conditional Intensity λ(x) of Shootings")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.grid(True)
plt.tight_layout()
plt.show()

# --- Plot Conditional Intensity with Gang Boundaries ---
# Assuming the gang boundaries are a part of your data (e.g., a shapefile or GeoDataFrame)
plt.figure(figsize=(14, 8))

## Plot gang boundaries (for context)
#gang_boundaries = gpd.read_file("gang_boundaries.shp")  # Example path to gang boundaries
ax = gang_boundaries.plot(color="none", edgecolor='black', linewidth=1)

# Plot shooting incidents and intensity as a heatmap
plt.scatter(df_grid['LONGITUDE'], df_grid['LATITUDE'], c=df_grid['Lambda'], cmap='viridis', s=10, alpha=0.6)

plt.colorbar(label="Lambda (Intensity)")
plt.title("Shootings Intensity λ(x) with Gang Boundaries")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
<ipython-input-114-72c7d1d5f855>:9: DtypeWarning: Columns (25,26,27) have mixed types. Specify dtype option on import or set low_memc
  gun_df = pd.read_csv("chicago_shooting.csv")
<ipython-input-114-72c7d1d5f855>:11: UserWarning: Could not infer format, so each element will be parsed individually, falling back t
  gun_df["DATE"] = pd.to_datetime(gun_df["DATE"])
<ipython-input-114-72c7d1d5f855>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2007
<ipython-input-114-72c7d1d5f855>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2008
<ipython-input-114-72c7d1d5f855>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2009
<ipython-input-114-72c7d1d5f855>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2010
<ipython-input-114-72c7d1d5f855>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2011
Processed gang boundaries for 2012
⚠️ Could not process year 2013: 2013_Gang_Boundaries.shp: No such file or directory
<ipython-input-114-72c7d1d5f855>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
<ipython-input-114-72c7d1d5f855>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2014
<ipython-input-114-72c7d1d5f855>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2015
<ipython-input-114-72c7d1d5f855>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2016
<ipython-input-114-72c7d1d5f855>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2017
<ipython-input-114-72c7d1d5f855>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2018
<ipython-input-114-72c7d1d5f855>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2019
<ipython-input-114-72c7d1d5f855>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2020
<ipython-input-114-72c7d1d5f855>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2021
<ipython-input-114-72c7d1d5f855>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2022
<ipython-input-114-72c7d1d5f855>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
Processed gang boundaries for 2023
Processed gang boundaries for 2024
<ipython-input-114-72c7d1d5f855>:37: DeprecationWarning: The 'unary_union' attribute is deprecated, use the 'union_all()' method inst
  gun_points_year["Inside_Gang"] = gun_points_year.geometry.within(gang_boundaries.unary_union)
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:       Num_Schools_500m   No. Observations:                61438
Model:                            GLM   Df Residuals:                    61433
Model Family:                 Poisson   Df Model:                            4
Link Function:                    Log   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:            -1.4677e+05
Date:                Sat, 10 May 2025   Deviance:                       89807.
Time:                        19:34:35   Pearson chi2:                 8.13e+04
No. Iterations:                     4   Pseudo R-squ. (CS):          9.140e-05
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          1.6097      0.007    245.995      0.000       1.597       1.623
Inside_Gang   -0.0081      0.004     -2.050      0.040      -0.016      -0.000
Month          0.0039      0.006      0.612      0.540      -0.009       0.016
Year           0.0101      0.008      1.335      0.182      -0.005       0.025
DayOfWeek     -0.0050      0.005     -0.947      0.344      -0.015       0.005
==============================================================================
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3804         try:
-> 3805             return self._engine.get_loc(casted_key)
   3806         except KeyError as err:

index.pyx in pandas._libs.index.IndexEngine.get_loc()
```

```
index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'Num_Schools_500m'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
------------------------------------------ 3 frames ------------------------------------------
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3810             ):
   3811                 raise InvalidIndexError(key)
-> 3812             raise KeyError(key) from err
   3813         except TypeError:
   3814             # If we have a listlike key, _check_indexing_error will raise

KeyError: 'Num_Schools_500m'
```

```
# 1. Convert DATE to datetime (if not already)
gun_points_m['DATE'] = pd.to_datetime(gun_points_m['DATE'])

# 2. Extract Year from DATE
gun_points_m['Year'] = gun_points_m['DATE'].dt.year

# 3. Spatial join: match gun incidents to grid cell IDs
shooting_join = gpd.sjoin(gun_points_m, grid, predicate="within")
shooting_join = shooting_join.reset_index().rename(columns={'index_right': 'grid_id'})

# 4. Aggregate data by grid cell
df_grid = shooting_join.groupby('grid_id').agg({
    'Inside_Gang': 'first',             # assumes static per cell
    'Num_Schools_500m': 'first',        # assumes static per cell
    'Year': 'median',                   # use median year of shootings in the cell
    'geometry': 'first'
}).reset_index()

# 5. Add shooting count per grid cell
shooting_counts = shooting_join['grid_id'].value_counts().sort_index()
df_grid['Shootings'] = df_grid['grid_id'].map(shooting_counts).fillna(0).astype(int)

# 6. Clean up and prepare features
df_grid.rename(columns={'Num_Schools_500m': 'Schools_500m'}, inplace=True)

# 7. Drop rows with NaNs or infinite values
X = df_grid[['Inside_Gang', 'Schools_500m', 'Year']].copy()
X = X.replace([np.inf, -np.inf], np.nan).dropna()
y = df_grid.loc[X.index, 'Shootings']  # align target with filtered rows

# 8. Fit Poisson regression
X = sm.add_constant(X)
model = sm.GLM(y, X, family=sm.families.Poisson())
results = model.fit()

# 9. Show results
print(results.summary())
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-92-cca683940684> in <cell line: 0>()
      1 # 1. Convert DATE to datetime (if not already)
----> 2 gun_points_m['DATE'] = pd.to_datetime(gun_points_m['DATE'])
      3
      4 # 2. Extract Year from DATE
      5 gun_points_m['Year'] = gun_points_m['DATE'].dt.year

NameError: name 'gun_points_m' is not defined
```

```
df_grid['Year'] = shooting_join.groupby('grid_id')['DATE'].min().dt.year  # Or mean if appropriate
```

```
X = df_grid[['Inside_Gang', 'Schools_500m', 'Year']].copy()

# Check for problematic values
print(X.isna().sum())     # shows NaN count per column
print(np.isinf(X).sum())  # shows inf count per column

# Drop rows with NaN or inf
X = X.replace([np.inf, -np.inf], np.nan)  # convert inf to NaN
X = X.dropna()

# Align y with cleaned X
y = df_grid.loc[X.index, 'Shootings']

# Add constant
X = sm.add_constant(X)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-91-d4f89a77a3f7> in <cell line: 0>()
----> 1 X = df_grid[['Inside_Gang', 'Schools_500m', 'Year']].copy()
      2
      3 # Check for problematic values
      4 print(X.isna().sum())     # shows NaN count per column
      5 print(np.isinf(X).sum())  # shows inf count per column

 ---------------------------------------- 3 frames ----------------------------------------
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in _raise_if_missing(self, key, indexer, axis_name)
   6250
   6251             not_found = list(ensure_index(key)[missing_mask.nonzero()[0]].unique())
-> 6252             raise KeyError(f"{not_found} not in index")
   6253
   6254     @overload

KeyError: "['Schools_500m'] not in index"
```

```
X = df_grid[['Inside_Gang', 'Schools_500m', 'Year']]
X = sm.add_constant(X)
model = sm.GLM(y, X, family=sm.families.Poisson(), offset=np.log(df_grid['Area']))
results = model.fit()
print(results.summary())
```

```
---------------------------------------------------------------------------
MissingDataError                          Traceback (most recent call last)
<ipython-input-154-db089b5f7986> in <cell line: 0>()
      1 X = df_grid[['Inside_Gang', 'Schools_500m', 'Year']]
      2 X = sm.add_constant(X)
----> 3 model = sm.GLM(y, X, family=sm.families.Poisson(), offset=np.log(df_grid['Area']))
      4 results = model.fit()
      5 print(results.summary())

 ---------------------------------------- 6 frames ----------------------------------------
/usr/local/lib/python3.11/dist-packages/statsmodels/base/data.py in _handle_constant(self, hasconst)
    132             exog_max = np.max(self.exog, axis=0)
    133             if not np.isfinite(exog_max).all():
--> 134                 raise MissingDataError('exog contains inf or nans')
    135             exog_min = np.min(self.exog, axis=0)
    136             const_idx = np.where(exog_max == exog_min)[0].squeeze()

MissingDataError: exog contains inf or nans
```

```
# 1. Spatial join: tag each shooting with the grid cell it's in
shooting_join = gpd.sjoin(gun_points_m, grid, predicate="within")
shooting_join = shooting_join.reset_index().rename(columns={'index_right': 'grid_id'})

# 2. Map static spatial features from grid to each shooting point
shooting_join['Inside_Gang'] = shooting_join['grid_id'].map(grid['Inside_Gang'])
shooting_join['Schools_500m'] = shooting_join['grid_id'].map(grid['Schools_500m'])

# 3. Create a numeric time covariate (days since first observation)
shooting_join = shooting_join.sort_values('DATE').copy()
shooting_join['Time_Index'] = (shooting_join['DATE'] - shooting_join['DATE'].min()).dt.days

# 4. Aggregate to grid level (or spatio-temporal cell if time-binning is desired)
df_grid = shooting_join.groupby('grid_id').agg({
```

```
    'Inside_Gang': 'first',
    'Schools_500m': 'first',
    'Time_Index': 'mean',
}).reset_index()

# 5. Add shooting counts per grid cell
shooting_counts = shooting_join['grid_id'].value_counts().sort_index()
df_grid['Shootings'] = df_grid['grid_id'].map(shooting_counts).fillna(0).astype(int)

# 6. Build Poisson regression
import statsmodels.api as sm

X = df_grid[['Inside_Gang', 'Schools_500m', 'Time_Index']]
X = sm.add_constant(X)
y = df_grid['Shootings']

model = sm.GLM(y, X, family=sm.families.Poisson())
results = model.fit()
print(results.summary())
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-90-234f506305db> in <cell line: 0>()
      1 # 1. Spatial join: tag each shooting with the grid cell it's in
----> 2 shooting_join = gpd.sjoin(gun_points_m, grid, predicate="within")
      3 shooting_join = shooting_join.reset_index().rename(columns={'index_right': 'grid_id'})
      4
      5 # 2. Map static spatial features from grid to each shooting point

NameError: name 'gun_points_m' is not defined
```

```
grid['area'] = grid.geometry.area  # in square meters
df_grid['Area'] = df_grid['grid_id'].map(grid['area'])

# Use log(Area) as offset in the Poisson GLM
offset = np.log(df_grid['Area'])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-89-cd32531f3675> in <cell line: 0>()
----> 1 grid['area'] = grid.geometry.area  # in square meters
      2 df_grid['Area'] = df_grid['grid_id'].map(grid['area'])
      3
      4 # Use log(Area) as offset in the Poisson GLM
      5 offset = np.log(df_grid['Area'])

NameError: name 'grid' is not defined
```

```
import statsmodels.api as sm

X = df_grid[['Inside_Gang', 'Schools_500m', 'Time_Index']]
X = sm.add_constant(X)
y = df_grid['Shootings']

model = sm.GLM(y, X, family=sm.families.Poisson(), offset=offset)
results = model.fit()
print(results.summary())
```

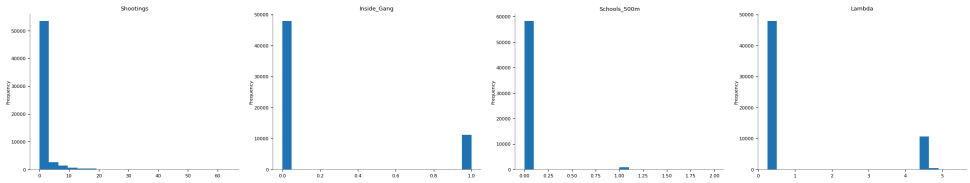
```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-88-8bbe2fc34e71> in <cell line: 0>()
      1 import statsmodels.api as sm
      2
----> 3 X = df_grid[['Inside_Gang', 'Schools_500m', 'Time_Index']]
      4 X = sm.add_constant(X)
      5 y = df_grid['Shootings']

                                 3 frames
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in _raise_if_missing(self, key, indexer, axis_name)
   6250
   6251             not_found = list(ensure_index(key)[missing_mask.nonzero()[0]].unique())
-> 6252             raise KeyError(f"{not_found} not in index")
   6253
   6254     @overload

KeyError: "['Schools_500m', 'Time_Index'] not in index"
```

```
df_grid
```

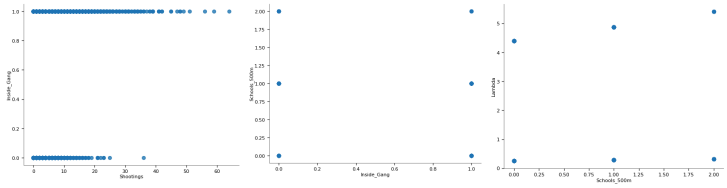
	Shootings	Inside_Gang	Schools_500m	Lambda
0	0	0	0	0.253143
1	0	0	0	0.253143
2	0	0	0	0.253143
3	0	0	0	0.253143
4	0	0	0	0.253143
...	...	...	...	...
59053	0	0	0	0.253143
59054	0	0	0	0.253143
59055	0	0	0	0.253143
59056	0	0	0	0.253143
59057	0	0	0	0.253143

59058 rows × 4 columns

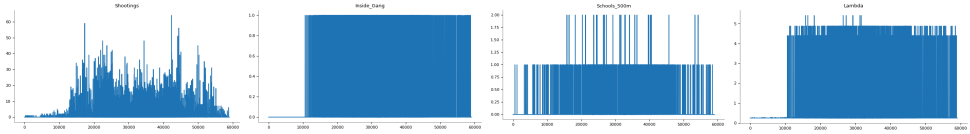
Distributions



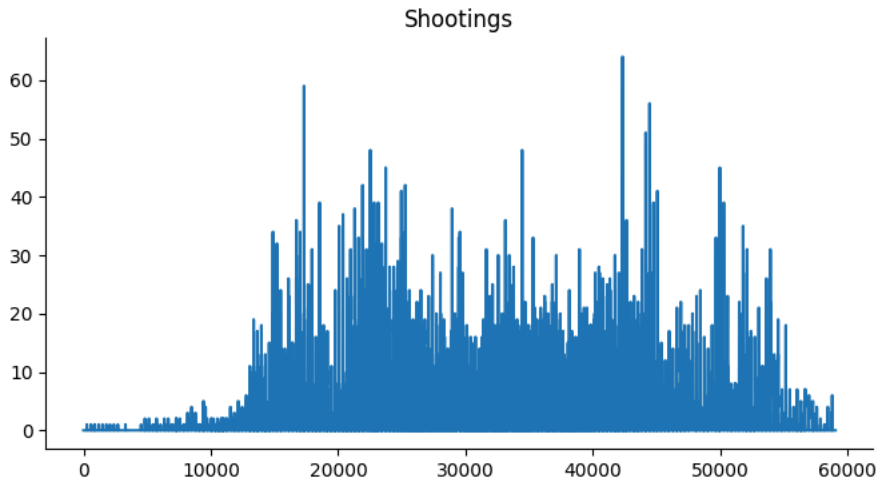
2-d distributions



Values



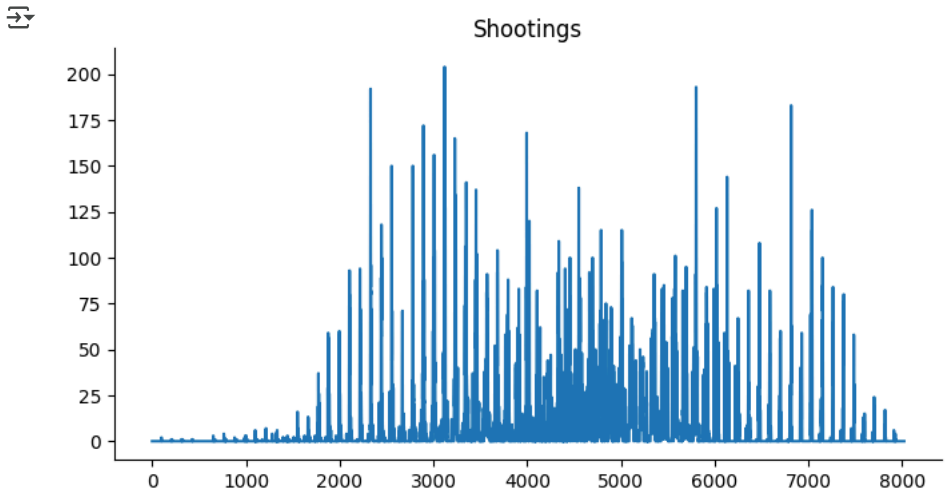

```
from matplotlib import pyplot as plt
df_grid['Shootings'].plot(kind='line', figsize=(8, 4), title='Shootings')
plt.gca().spines[['top', 'right']].set_visible(False)
```



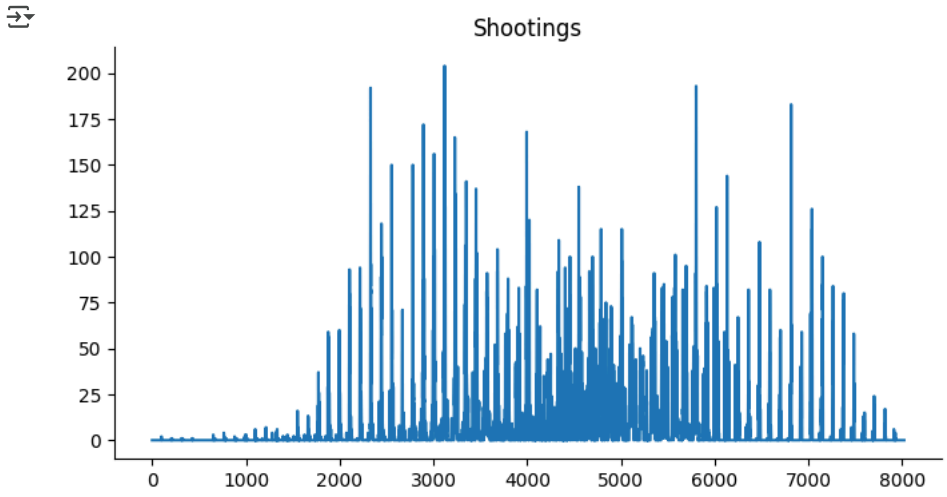

```
import pandas as pd
import matplotlib.pyplot as plt

# Assuming 'Year' and 'Month' are available
df_grid['Date'] = pd.to_datetime(df_grid['Year'].astype(str) + '-' + df_grid['Month'].astype(str) + '-01', format='%Y-%m-%d')

# Plotting
df_grid.plot(kind='line', x='Date', y='Shootings', figsize=(8, 4), title='Shootings per Month')
plt.gca().spines[['top', 'right']].set_visible(False)
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-87-a1ee7e9d5100> in <cell line: 0>()
      3
      4 # Assuming 'Year' and 'Month' are available
----> 5 df_grid['Date'] = pd.to_datetime(df_grid['Year'].astype(str) + '-' + df_grid['Month'].astype(str) + '-01', format='%Y-%m-%d')
      6
      7 # Plotting

3 frames
/usr/local/lib/python3.11/dist-packages/pandas/core/tools/datetimes.py in _array_strptime_with_fallback(arg, name, utc, fmt, exact,
errors)
    465     Call array_strptime, with fallback behavior depending on 'errors'.
    466     """
--> 467     result, tz_out = array_strptime(arg, fmt, exact=exact, errors=errors, utc=utc)
    468     if tz_out is not None:
    469         unit = np.datetime_data(result.dtype)[0]

strptime.pyx in pandas._libs.tslibs.strptime.array_strptime()

strptime.pyx in pandas._libs.tslibs.strptime.array_strptime()

strptime.pyx in pandas._libs.tslibs.strptime._parse_with_format()

ValueError: time data "0.911764705882355-0.1818181818181818-01" doesn't match format "%Y-%m-%d", at position 0. You might want to try:
    - passing `format` if your strings have a consistent format;
    - passing `format='ISO8601'` if your strings are all ISO8601 but not necessarily in exactly the same format;
    - passing `format='mixed'`, and the format will be inferred for each element individually. You might want to use `dayfirst`
alongside this.
```

```
from matplotlib import pyplot as plt
df_grid['Shootings'].plot(kind='line', figsize=(8, 4), title='Shootings')
plt.gca().spines[['top', 'right']].set_visible(False)
```




```
from matplotlib import pyplot as plt
df_grid['Shootings'].plot(kind='line', figsize=(8, 4), title='Shootings')
plt.gca().spines[['top', 'right']].set_visible(False)
```




```
import statsmodels.api as sm

# Suppose df_grid is your DataFrame with:
# 'Shootings': count of incidents in grid cell
# 'Inside_Gang': binary
# 'Schools_500m': integer

X = df_grid[["Inside_Gang", "Schools_500m"]]
X = sm.add_constant(X)  # add intercept
y = df_grid["Shootings"]

model = sm.GLM(y, X, family=sm.families.Poisson()).fit()
print(model.summary())
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-86-cf0926eb649d> in <cell line: 0>()
      6 # 'Schools_500m': integer
      7
----> 8 X = df_grid[["Inside_Gang", "Schools_500m"]]
      9 X = sm.add_constant(X)  # add intercept
     10 y = df_grid["Shootings"]

 3 frames
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in _raise_if_missing(self, key, indexer, axis_name)
   6250
   6251             not_found = list(ensure_index(key)[missing_mask.nonzero()[0]].unique())
-> 6252             raise KeyError(f"{not_found} not in index")
   6253
   6254     @overload

KeyError: "['Schools_500m'] not in index"
```

```
from shapely.geometry import Polygon


# 1. Create a spatial grid over the study area (e.g., 250m x 250m grid)
xmin, ymin, xmax, ymax = gun_points_m.total_bounds
grid_size = 220 # meters
rows = int((ymax - ymin) / grid_size)
cols = int((xmax - xmin) / grid_size)

# Generate grid polygons
grid_cells = []
for i in range(cols):
    for j in range(rows):
        x0 = xmin + i * grid_size
        y0 = ymin + j * grid_size
        x1 = x0 + grid_size
        y1 = y0 + grid_size
        grid_cells.append(Polygon([(x0, y0), (x1, y0), (x1, y1), (x0, y1)]))

grid = gpd.GeoDataFrame(geometry=grid_cells, crs=gun_points_m.crs)

# 2. Count number of shootings per grid cell
shooting_join = gpd.sjoin(gun_points_m, grid, predicate="within")
shootings_per_cell = shooting_join.groupby("index_right").size()
grid["Shootings"] = grid.index.map(shootings_per_cell).fillna(0).astype(int)

# 3. Count number of schools per grid cell
school_join = gpd.sjoin(schools_m, grid, predicate="within")
schools_per_cell = school_join.groupby("index_right").size()
grid["Schools_500m"] = grid.index.map(schools_per_cell).fillna(0).astype(int)

# 4. Determine whether each grid cell intersects a gang boundary
grid["Inside_Gang"] = grid.geometry.intersects(gang_boundaries.unary_union).astype(int)

# 5. Convert to DataFrame for modeling
df_grid = grid[["Shootings", "Inside_Gang", "Schools_500m"]].copy()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-85-c0abf0ae58f3> in <cell line: 0>()
      3
      4 # 1. Create a spatial grid over the study area (e.g., 250m x 250m grid)
----> 5 xmin, ymin, xmax, ymax = gun_points_m.total_bounds
      6 grid_size = 220 # meters
      7 rows = int((ymax - ymin) / grid_size)

NameError: name 'gun_points_m' is not defined
```

```
import statsmodels.api as sm
import numpy as np
from scipy.special import expit  # For sigmoid function

# Assuming you have the model 'results' from the previous Poisson GLM

# 1. Print model summary to get coefficients and p-values
```

```
print(results.summary())

# 2. Extract coefficients (for Inside_Gang as an example)
gang_coef = results.params['Inside_Gang']
base_intensity = results.params['const']

# 3. Calculate the logit probability for a grid cell that is inside gang territory (Inside_Gang = 1)
logit_prob = expit(base_intensity + gang_coef * 1)  # 1 for Inside_Gang = 1

# 4. Calculate the logit probability for a grid cell that is not inside gang territory (Inside_Gang = 0)
logit_prob_no_gang = expit(base_intensity + gang_coef * 0)  # 0 for Inside_Gang = 0

# 5. Print out the probabilities
print(f"Logit Probability for Inside Gang (Inside_Gang = 1): {logit_prob:.4f}")
print(f"Logit Probability for Outside Gang (Inside_Gang = 0): {logit_prob_no_gang:.4f}")

# 6. Extract p-value for the Inside_Gang coefficient
gang_p_value = results.pvalues['Inside_Gang']
print(f"P-value for 'Inside_Gang' coefficient: {gang_p_value:.4f}")
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                61438
Model:                            GLM   Df Residuals:                    61433
Model Family:                 Poisson   Df Model:                            4
Link Function:                    Log   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:            -1.0618e+05
Date:                Sat, 10 May 2025   Deviance:                       69840.
Time:                        19:25:11   Pearson chi2:                 6.20e+04
No. Iterations:                     5   Pseudo R-squ. (CS):             0.2548
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          -0.3700      0.012    -31.594      0.000      -0.393      -0.347
Inside_Gang     0.4503      0.006     74.355      0.000       0.438       0.462
Month           0.4432      0.010     45.689      0.000       0.424       0.462
Year            0.5953      0.013     45.143      0.000       0.569       0.621
DayOfWeek       0.4528      0.008     56.336      0.000       0.437       0.469
==============================================================================
Logit Probability for Inside Gang (Inside_Gang = 1): 0.5201
Logit Probability for Outside Gang (Inside_Gang = 0): 0.4085
P-value for 'Inside_Gang' coefficient: 0.0000
```

```
from scipy.special import expit  # sigmoid function

# Extract the coefficients from the fitted model
intercept = results.params['const']
gang_coef = results.params['Inside_Gang']

# Calculate the predicted log-intensity (log-likelihood) for both inside and outside gang territory
logit_inside_gang = intercept + gang_coef * 1  # Inside Gang = 1
logit_outside_gang = intercept + gang_coef * 0  # Outside Gang = 0

# Convert log-intensity to expected rate using the exponential (Poisson assumption)
lambda_inside_gang = np.exp(logit_inside_gang)
lambda_outside_gang = np.exp(logit_outside_gang)

# If you want to convert the rate to a probability, you can use the following:
# For example, if you have a specific grid or location and want to estimate the probability of a shooting,
# you might use a threshold intensity.

# For example, calculate the likelihood (probability) of shooting occurring:
# This assumes you're looking for the rate per unit of time or space and assuming uniform space (e.g., grid cells)

# The logistic regression transformation of the Poisson rate (for a binary outcome):
prob_inside_gang = expit(logit_inside_gang)  # Probability of shooting inside gang territory
prob_outside_gang = expit(logit_outside_gang)  # Probability of shooting outside gang territory

print(f"Predicted likelihood of shooting inside gang territory: {prob_inside_gang:.4f}")
print(f"Predicted likelihood of shooting outside gang territory: {prob_outside_gang:.4f}")
```

```
Predicted likelihood of shooting inside gang territory: 0.5201
Predicted likelihood of shooting outside gang territory: 0.4085
```

```
!pip install libpysal
```

```
Collecting libpysal
  Downloading libpysal-4.13.0-py3-none-any.whl.metadata (4.8 kB)
Requirement already satisfied: beautifulsoup4>=4.10 in /usr/local/lib/python3.11/dist-packages (from libpysal) (4.13.4)
Requirement already satisfied: geopandas>=0.10.0 in /usr/local/lib/python3.11/dist-packages (from libpysal) (1.0.1)
Requirement already satisfied: numpy>=1.22 in /usr/local/lib/python3.11/dist-packages (from libpysal) (2.0.2)
Requirement already satisfied: packaging>=22 in /usr/local/lib/python3.11/dist-packages (from libpysal) (24.2)
Requirement already satisfied: pandas>=1.4 in /usr/local/lib/python3.11/dist-packages (from libpysal) (2.2.2)
Requirement already satisfied: platformdirs>=2.0.2 in /usr/local/lib/python3.11/dist-packages (from libpysal) (4.3.7)
Requirement already satisfied: requests>=2.27 in /usr/local/lib/python3.11/dist-packages (from libpysal) (2.32.3)
Requirement already satisfied: scipy>=1.8 in /usr/local/lib/python3.11/dist-packages (from libpysal) (1.15.2)
Requirement already satisfied: shapely>=2.0.1 in /usr/local/lib/python3.11/dist-packages (from libpysal) (2.1.0)
Requirement already satisfied: scikit-learn>=1.1 in /usr/local/lib/python3.11/dist-packages (from libpysal) (1.6.1)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4>=4.10->libpysal) (2.7)
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4>=4.10->libpysal)
Requirement already satisfied: pyogrio>=0.7.2 in /usr/local/lib/python3.11/dist-packages (from geopandas>=0.10.0->libpysal) (0.10.0)
Requirement already satisfied: pyproj>=3.3.0 in /usr/local/lib/python3.11/dist-packages (from geopandas>=0.10.0->libpysal) (3.7.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.4->libpysal) (2.9.0.pos
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.4->libpysal) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=1.4->libpysal) (2025.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.27->libpysal) (3.4.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.27->libpysal) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.27->libpysal) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.27->libpysal) (2025.4.26)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=1.1->libpysal) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=1.1->libpysal) (3.6.0
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas>=1.4->libpysal)
Downloading libpysal-4.13.0-py3-none-any.whl (2.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.8/2.8 MB 20.9 MB/s eta 0:00:00
Installing collected packages: libpysal
Successfully installed libpysal-4.13.0
```

```
!pip install smooth
```

```
Collecting smooth
  Downloading smooth-0.0.1.tar.gz (36 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: smooth
  Building wheel for smooth (setup.py) ... done
  Created wheel for smooth: filename=smooth-0.0.1-py3-none-any.whl size=20320 sha256=c95fc23d09404c70b996ee980a5dda2c4507fc4043e22c64de9
  Stored in directory: /root/.cache/pip/wheels/65/a4/0e/fe46b322be6675bdb1995aa0b653bc8f72c1313f7f2e6f9509
Successfully built smooth
Installing collected packages: smooth
Successfully installed smooth-0.0.1
```

```
!pip install esda
```

```
Collecting esda
  Downloading esda-2.7.0-py3-none-any.whl.metadata (2.0 kB)
Requirement already satisfied: geopandas>=0.12 in /usr/local/lib/python3.11/dist-packages (from esda) (1.0.1)
Requirement already satisfied: libpysal>=4.12 in /usr/local/lib/python3.11/dist-packages (from esda) (4.13.0)
Requirement already satisfied: numpy>=1.24 in /usr/local/lib/python3.11/dist-packages (from esda) (2.0.2)
Requirement already satisfied: pandas>1.5 in /usr/local/lib/python3.11/dist-packages (from esda) (2.2.2)
Requirement already satisfied: scikit-learn>=1.2 in /usr/local/lib/python3.11/dist-packages (from esda) (1.6.1)
Requirement already satisfied: scipy>=1.9 in /usr/local/lib/python3.11/dist-packages (from esda) (1.15.2)
Requirement already satisfied: shapely>=2.0 in /usr/local/lib/python3.11/dist-packages (from esda) (2.1.0)
Requirement already satisfied: pyogrio>=0.7.2 in /usr/local/lib/python3.11/dist-packages (from geopandas>=0.12->esda) (0.10.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from geopandas>=0.12->esda) (24.2)
Requirement already satisfied: pyproj>=3.3.0 in /usr/local/lib/python3.11/dist-packages (from geopandas>=0.12->esda) (3.7.1)
Requirement already satisfied: beautifulsoup4>=4.10 in /usr/local/lib/python3.11/dist-packages (from libpysal>=4.12->esda) (4.13.4)
Requirement already satisfied: platformdirs>=2.0.2 in /usr/local/lib/python3.11/dist-packages (from libpysal>=4.12->esda) (4.3.7)
Requirement already satisfied: requests>=2.27 in /usr/local/lib/python3.11/dist-packages (from libpysal>=4.12->esda) (2.32.3)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>1.5->esda) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>1.5->esda) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>1.5->esda) (2025.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=1.2->esda) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=1.2->esda) (3.6.0)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4>=4.10->libpysal>=4.12->esda
Requirement already satisfied: typing-extensions>=4.0.0 in /usr/local/lib/python3.11/dist-packages (from beautifulsoup4>=4.10->libpysal>
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from pyogrio>=0.7.2->geopandas>=0.12->esda) (2025.4.2
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas>1.5->esda) (1.17
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.27->libpysal>=4.12-
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.27->libpysal>=4.12->esda) (3.10
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.27->libpysal>=4.12->esda)
Downloading esda-2.7.0-py3-none-any.whl (142 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 142.8/142.8 kB 3.7 MB/s eta 0:00:00
Installing collected packages: esda
Successfully installed esda-2.7.0
```