ComputationalEssayFinalDraft

December 20, 2024

Robberies in Vegas **Calvin Hahn**

Introduction This project is to look at spatial autocorrelation with point data of robberies in Las Vegas Nevada. For my essay I found point data on reported robberies in Las Vegas. This data has been updated this year and there are 3,726 recorded instances. This will give me a lot to work with when exploring the data. The casinos in Las Vegas create a unique city. In addition it is a newer city. This will allow us to possibly find some interesting correlations. Spatial distribution and clustering will be prevalent in Las Vegas. It will be interesting to see if it clusters by income of an area. Will there be clusters in the rich casino districts or will there be clusters in low income Las Vegas neighborhoods. In addition the point data includes the suburbs.Are the robberies Las Vegas Nevada clustered around a specific place or thing. Are the robberies in random places or are they not random. The points will be explored through a variety of ways. There will be visual plots that provide general overviews and plots that provide more in depth analysis. The goal is to identify patterns within the points.

[1]:
```
import pandas as pd
import geopandas as gpd
```

Importing the librarys pandas and geopandas. Pandas is good for csv files and geopandas has great tools for geospatial analaysis.

[2]:
```
lv_robb_df = pd.read_csv('Robberies/LV_Robb.csv')
```

Read in the CSV file as a variable

[3]:
```
import seaborn
import geopandas as gpd
import pandas as pd
from geosnap import io as gio
import matplotlib.pyplot as plt
```

More imports to assit with Geo Spatial analysis

[4]:
```
print(lv_robb_df.columns)
```

```
Index(['OBJECTID', 'Event Number', 'Reported On Date', 'Location', 'CSZ',
       'Area Command', 'Beat', 'Offense Group', 'Crime Against',
       'Offense Category', 'Offense', 'NIBRS Offense Code', 'Violent Crime',
       'ShootingVictims', 'Shooting Victim Count', 'Weapons', 'Longitude',
```

```
       'Latitude', 'Days From Report Ending', 'x', 'y'],
      dtype='object')
```

Tells the collums that are in the LV_Robb csv

[5]:
```
lv_robb_df = pd.read_csv('Robberies/LV_Robb.csv')
print(lv_robb_df.head())
print(lv_robb_df.columns)
```

```
   OBJECTID     Event Number     Reported On Date                 Location  \
0    190758  LLV220100001394  1/1/2022 3:11:32 PM       100 Block Fremont St
1    151444  LLV220100001498  1/1/2022 3:48:25 PM     4500 Block PARADISE RD
2    101348  LLV220100008942  1/3/2022 5:50:49 AM    1700 Block E OAKEY BLVD
3    298940  LLV220100009295  1/3/2022 7:09:44 AM  1700 Block N Decatur Blvd
4    298477  LLV220100009802  1/3/2022 9:35:21 AM     4100 Block BOULDER HWY

                 CSZ Area Command Beat Offense Group Crime Against  \
0  LAS VEGAS, NV  89101         DTAC   A1             A      Property
1  LAS VEGAS, NV  89119         CCAC   M3             A      Property
2  LAS VEGAS, NV  89104         DTAC   C4             A      Property
3  LAS VEGAS, NV  89108          BAC   U1             A      Property
4  LAS VEGAS, NV  89121         SEAC   H1             A      Property

  Offense Category … NIBRS Offense Code  Violent Crime  ShootingVictims  \
0          ROBBERY …                120           True                N
1          ROBBERY …                120           True                N
2          ROBBERY …                120           True                N
3          ROBBERY …                120           True                N
4          ROBBERY …                120           True                N

  Shooting Victim Count                                     Weapons  \
0                     0  Knife/Cutting Instrument (Icepick, Ax, Etc.)
1                     0                                      Handgun
2                     0             Blunt Object (Club, Hammer, etc.)
3                     0                                        Rifle
4                     0                                      Unknown

    Longitude   Latitude  Days From Report Ending           x          y
0 -115.144923  36.170741                     1021 -115.144923  36.170741
1 -115.152660  36.107343                     1021 -115.152660  36.107343
2 -115.126537  36.151452                     1020 -115.126537  36.151452
3 -115.206116  36.191786                     1020 -115.206116  36.191786
4 -115.083752  36.130725                     1020 -115.083752  36.130725

[5 rows x 21 columns]
Index(['OBJECTID', 'Event Number', 'Reported On Date', 'Location', 'CSZ',
       'Area Command', 'Beat', 'Offense Group', 'Crime Against',
       'Offense Category', 'Offense', 'NIBRS Offense Code', 'Violent Crime',
```

```
 'ShootingVictims', 'Shooting Victim Count', 'Weapons', 'Longitude',
 'Latitude', 'Days From Report Ending', 'x', 'y'],
dtype='object')
```

This is us loading the data and inspecting it

[6]:
```python
lv_robb_gdf = gpd.GeoDataFrame(lv_robb_df, geometry=gpd.
 ↪points_from_xy(lv_robb_df['Longitude'], lv_robb_df['Latitude']))
print(lv_robb_gdf.head())
```

```
   OBJECTID      Event Number      Reported On Date                  Location  \
0    190758  LLV220100001394  1/1/2022 3:11:32 PM       100 Block Fremont St
1    151444  LLV220100001498  1/1/2022 3:48:25 PM     4500 Block PARADISE RD
2    101348  LLV220100008942  1/3/2022 5:50:49 AM    1700 Block E OAKEY BLVD
3    298940  LLV220100009295  1/3/2022 7:09:44 AM  1700 Block N Decatur Blvd
4    298477  LLV220100009802  1/3/2022 9:35:21 AM     4100 Block BOULDER HWY

                 CSZ Area Command Beat Offense Group Crime Against  \
0  LAS VEGAS, NV  89101         DTAC   A1             A      Property
1  LAS VEGAS, NV  89119         CCAC   M3             A      Property
2  LAS VEGAS, NV  89104         DTAC   C4             A      Property
3  LAS VEGAS, NV  89108          BAC   U1             A      Property
4  LAS VEGAS, NV  89121         SEAC   H1             A      Property

  Offense Category  … Violent Crime  ShootingVictims  Shooting Victim Count  \
0          ROBBERY  …          True                N                      0
1          ROBBERY  …          True                N                      0
2          ROBBERY  …          True                N                      0
3          ROBBERY  …          True                N                      0
4          ROBBERY  …          True                N                      0

                                        Weapons   Longitude   Latitude  \
0  Knife/Cutting Instrument (Icepick, Ax, Etc.) -115.144923  36.170741
1                                       Handgun -115.152660  36.107343
2             Blunt Object (Club, Hammer, etc.) -115.126537  36.151452
3                                         Rifle -115.206116  36.191786
4                                       Unknown -115.083752  36.130725

   Days From Report Ending           x          y                    geometry
0                     1021 -115.144923  36.170741  POINT (-115.14492 36.17074)
1                     1021 -115.152660  36.107343  POINT (-115.15266 36.10734)
2                     1020 -115.126537  36.151452  POINT (-115.12654 36.15145)
3                     1020 -115.206116  36.191786  POINT (-115.20612 36.19179)
4                     1020 -115.083752  36.130725  POINT (-115.08375 36.13072)

[5 rows x 22 columns]
```

This is us converting the data into a geodataframe

Map with CRS of the Points

[7]:
```
lv_robb_gdf = lv_robb_gdf.set_crs("EPSG:4326")
```

Sets the CRS with a geographic coordinate system

[8]:
```
lv_robb_gdf.explore()
```

[8]: <folium.folium.Map at 0x750f708aeb30>

Map of Points with CRS voomed out

[9]:
```
lv_robb_gdf = lv_robb_gdf.set_crs("EPSG:4326")
print(lv_robb_gdf.crs)
lv_robb_gdf.explore()
```

```
EPSG:4326
```

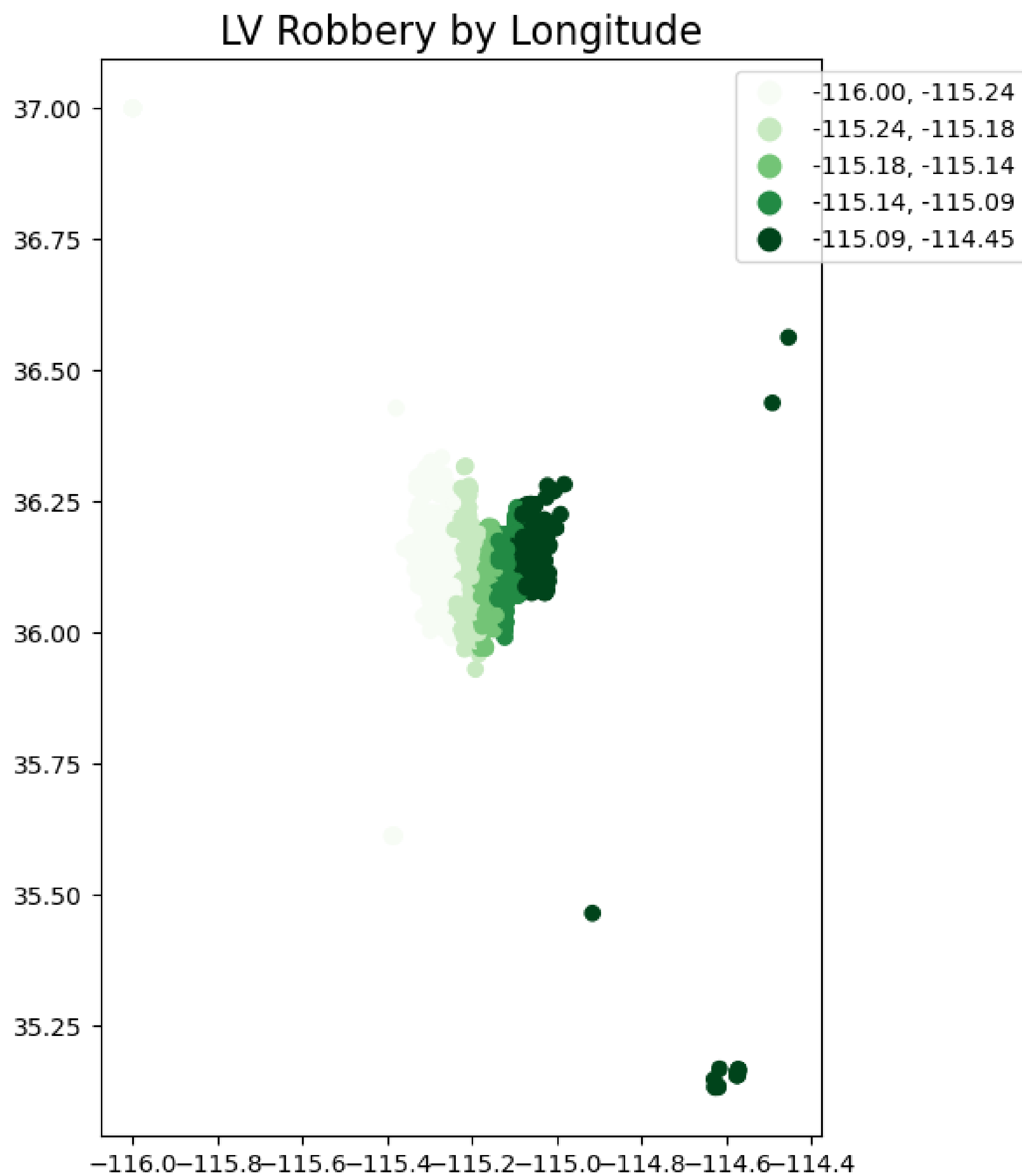
[9]: <folium.folium.Map at 0x750f708ad1b0>

Sets the CRS and displays an interactive map

Las Vegas Robberies by Longitude

[10]:
```
lv_robb_df = pd.read_csv('Robberies/LV_Robb.csv')



lv_robb_gdf = gpd.GeoDataFrame(lv_robb_df, geometry=gpd.
 ↪points_from_xy(lv_robb_df['Longitude'], lv_robb_df['Latitude']))
lv_robb_gdf = lv_robb_gdf.set_crs("EPSG:4326")



print(lv_robb_gdf.crs)



fig, ax = plt.subplots(1, 1, figsize=(12, 8))
lv_robb_gdf.plot(column='Longitude', scheme='quantiles', legend=True,
                 legend_kwds={'bbox_to_anchor': (1.3, 1)},
                 cmap='Greens', ax=ax)

plt.title('LV Robbery by Longitude', fontsize=16)
plt.show()
```

```
EPSG:4326
```



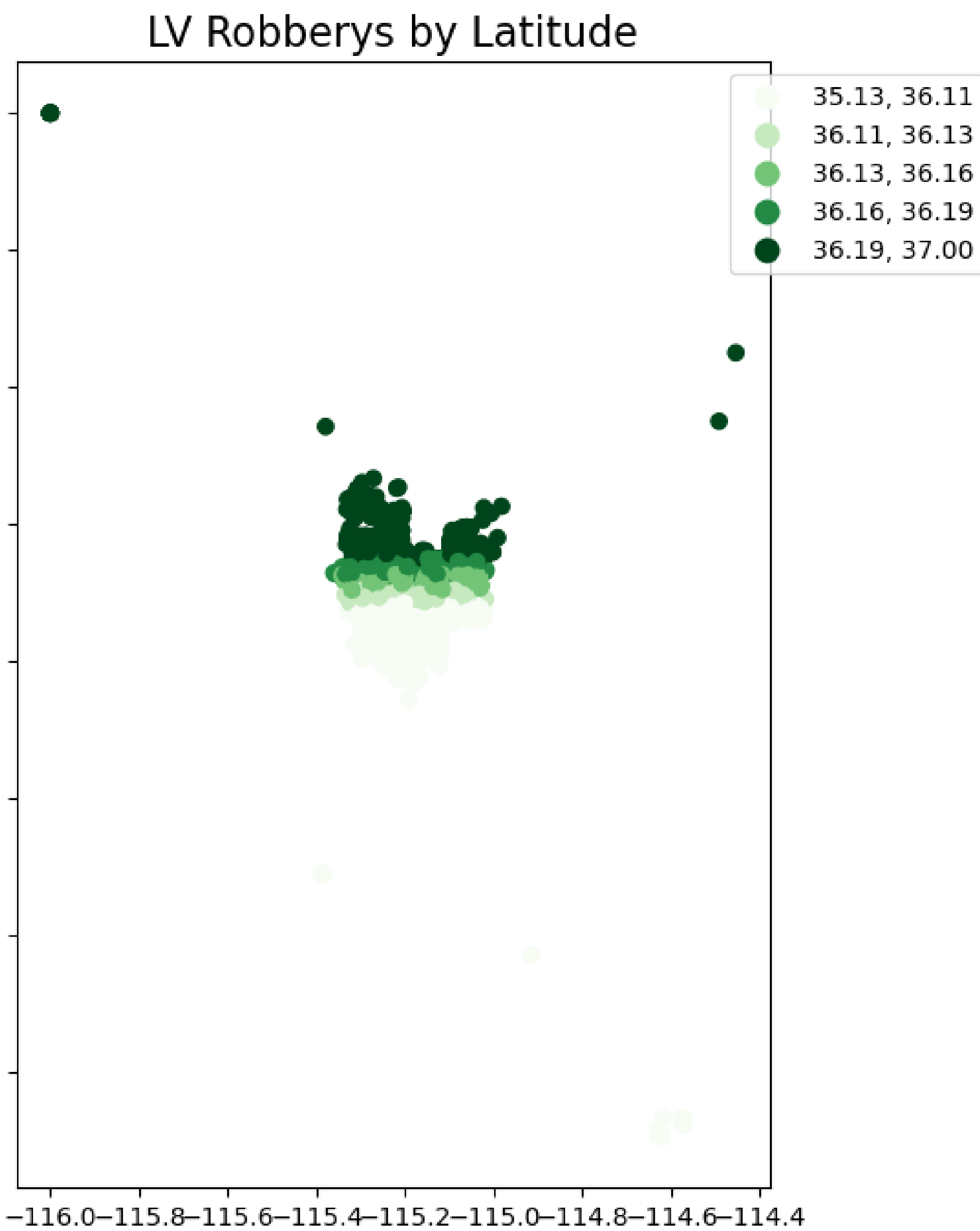
This is that data being shown in terms of the different longitudes

Las Vegas Robberies by Latitude

[11]:
```
lv_robb_df = pd.read_csv('Robberies/LV_Robb.csv')

lv_robb_gdf = gpd.GeoDataFrame(lv_robb_df, geometry=gpd.
 ↪points_from_xy(lv_robb_df['Longitude'], lv_robb_df['Latitude']))
lv_robb_gdf = lv_robb_gdf.set_crs("EPSG:4326")
```

```python
print(lv_robb_gdf.crs)

fig, ax = plt.subplots(1, 1, figsize=(12, 8))
lv_robb_gdf.plot(column='Latitude', scheme='quantiles', legend=True,
                 legend_kwds={'bbox_to_anchor': (1.3, 1)},
                 cmap='Greens', ax=ax)

plt.title('LV Robberys by Latitude', fontsize=16)
plt.show()
```

```
EPSG:4326
```

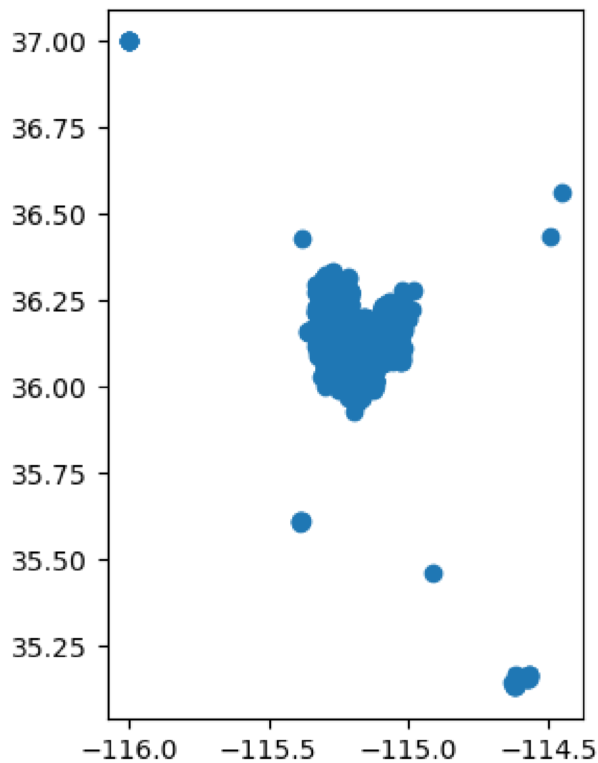


Shows the distribution of robbery points by latitude. There is a clustering of points in the top of the higer latitude.

Simple Plot

[12]:
```
lv_robb_gdf.plot()
```

[12]: <Axes: >



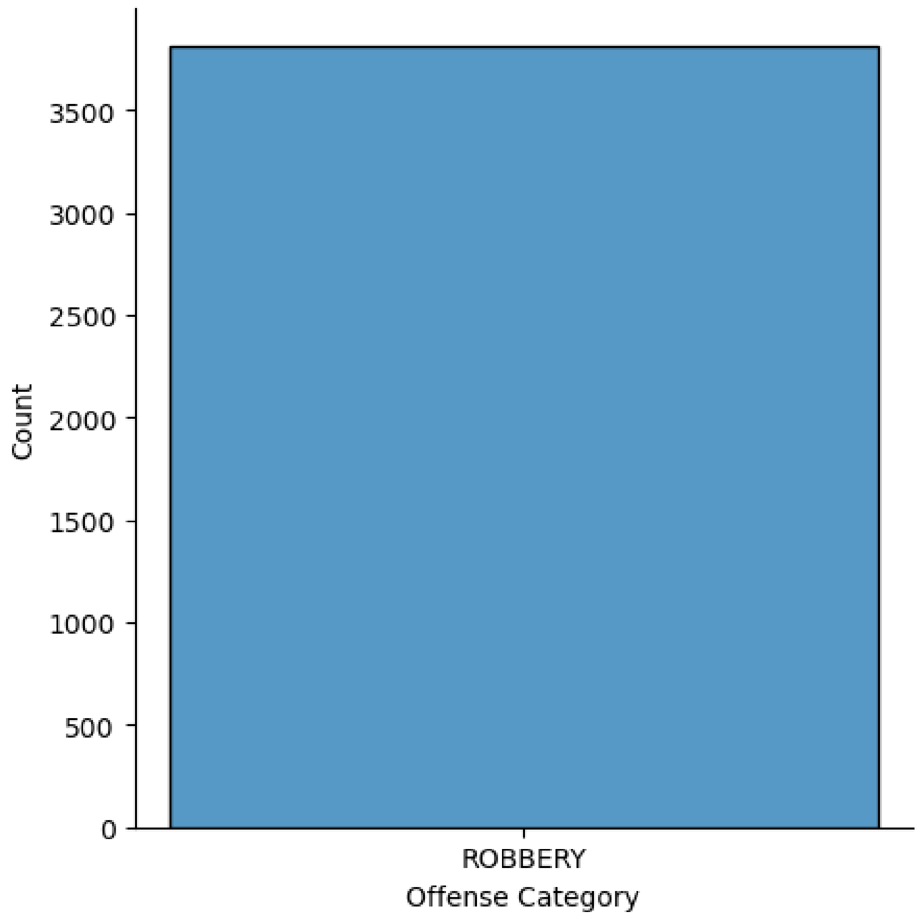
We see the data is centered in what is the urban Las Vegas area

[13]:
```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt


lv_robb_df = pd.read_csv('Robberies/LV_Robb.csv')

lv_robb_gdf = gpd.GeoDataFrame(lv_robb_df, geometry=gpd.
 ↪points_from_xy(lv_robb_df['Longitude'], lv_robb_df['Latitude']))
lv_robb_gdf = lv_robb_gdf.set_crs("EPSG:4326")
```

Seaborn Plot

[14]:
```
seaborn.displot(lv_robb_gdf, x='Offense Category')
```

[14]: <seaborn.axisgrid.FacetGrid at 0x750f675b79d0>



All the grid plotted was offense category of the robbery there is over 3500 instances

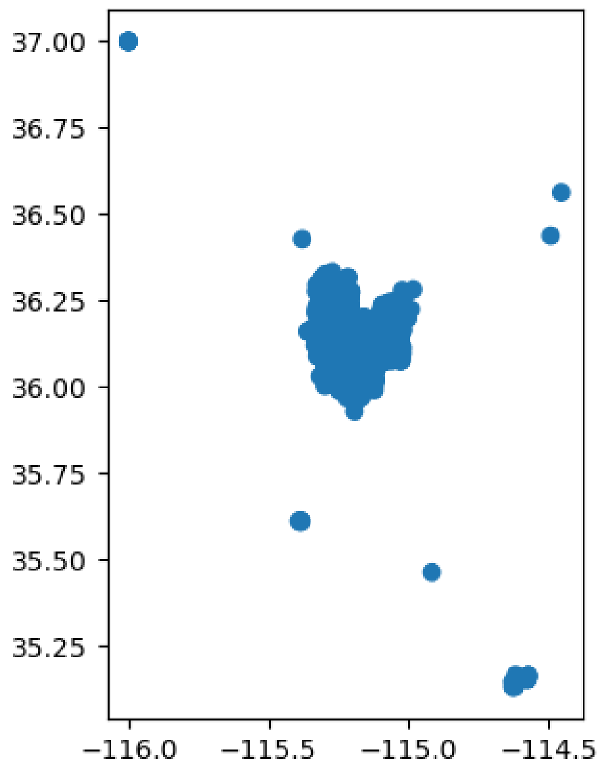
[15]:
```
print(lv_robb_gdf['Offense Category'].describe())
```

```
count        3808
unique          1
top       ROBBERY
freq         3808
Name: Offense Category, dtype: object
```

Offense Category Plot

[16]:
```
lv_robb_gdf.plot(column='Offense Category')
```

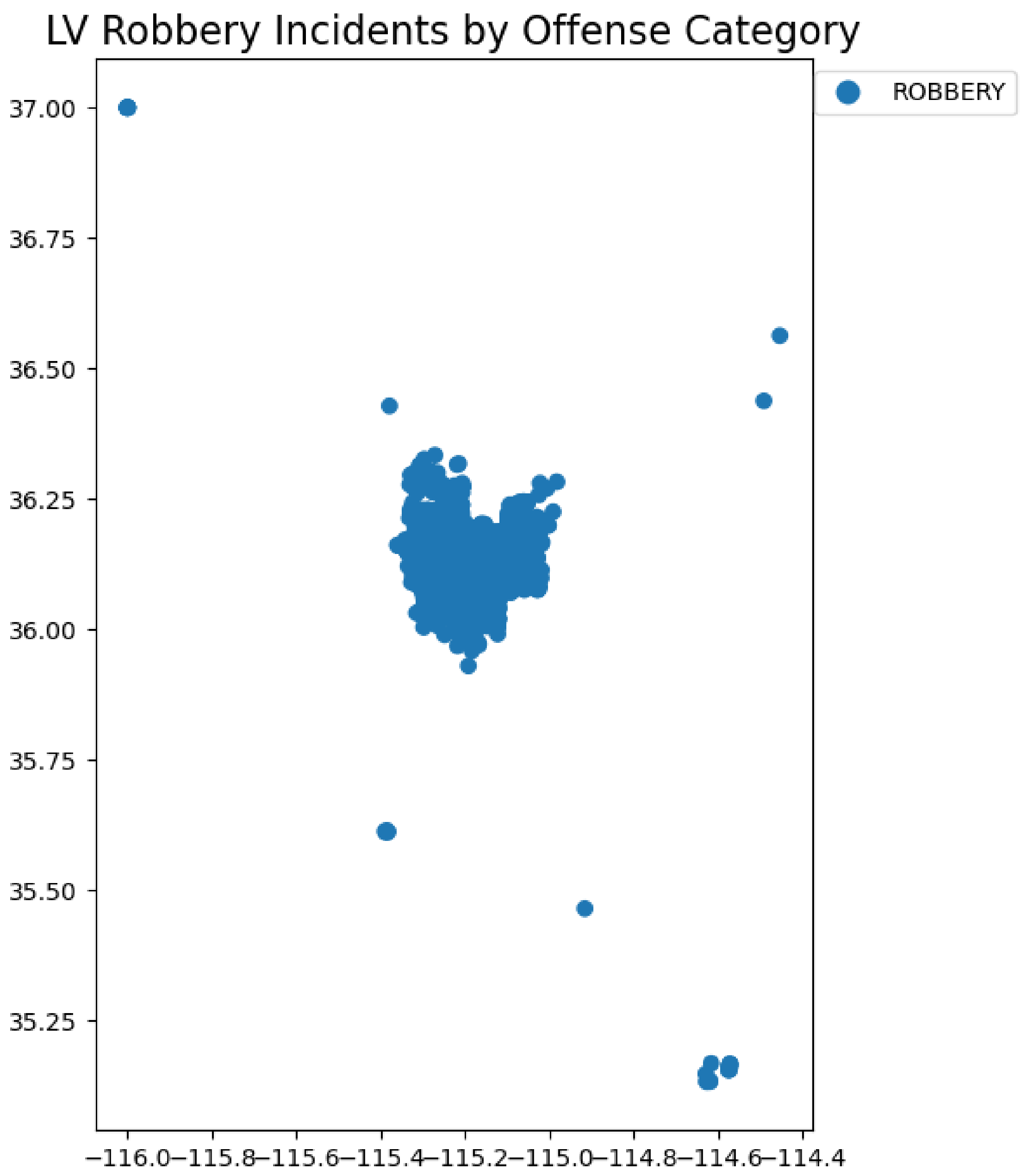
[16]: `<Axes: >`



Basic plot, once again we see the robberies clustered in the urban Las Vegas area

Chrolopleath Map

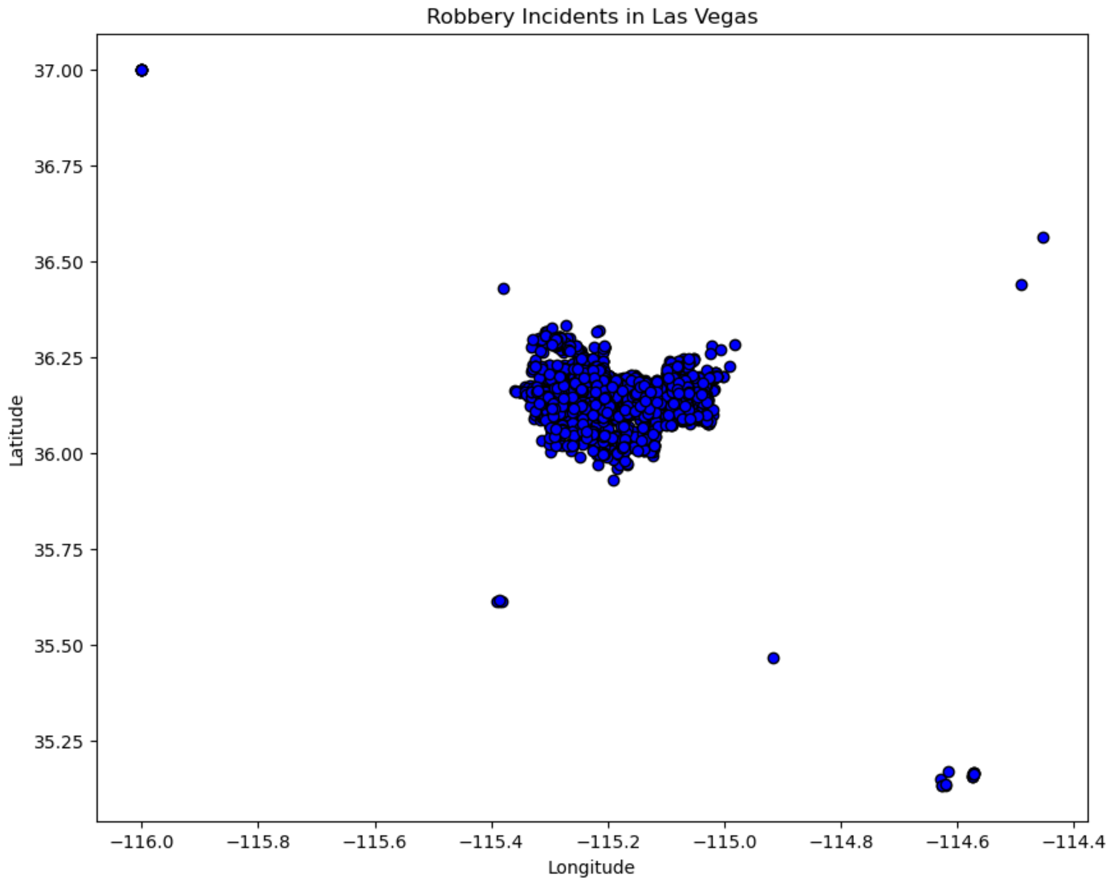
[17]:
```
fig, ax = plt.subplots(1, 1, figsize=(12, 8))
lv_robb_gdf.plot(column='Offense Category', categorical=True, legend=True,
                 legend_kwds={'bbox_to_anchor': (1.3, 1)},
                 cmap='tab20', ax=ax)

plt.title('LV Robbery Incidents by Offense Category', fontsize=16)
plt.show()
```



Every point in the data set is a robbery. Reinforces how there is a lot of Robbery in the Urban Las Vegas area.

[18]:
```
import numpy as np
from pointpats import PointPattern
import matplotlib.pyplot as plt



points = lv_robb_gdf[['Longitude', 'Latitude']].values.tolist()
```

```
pp = PointPattern(points)


plt.figure(figsize=(10, 8))
plt.scatter(*zip(*points), c='blue', marker='o', edgecolor='k')
plt.title('Robbery Incidents in Las Vegas')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
```



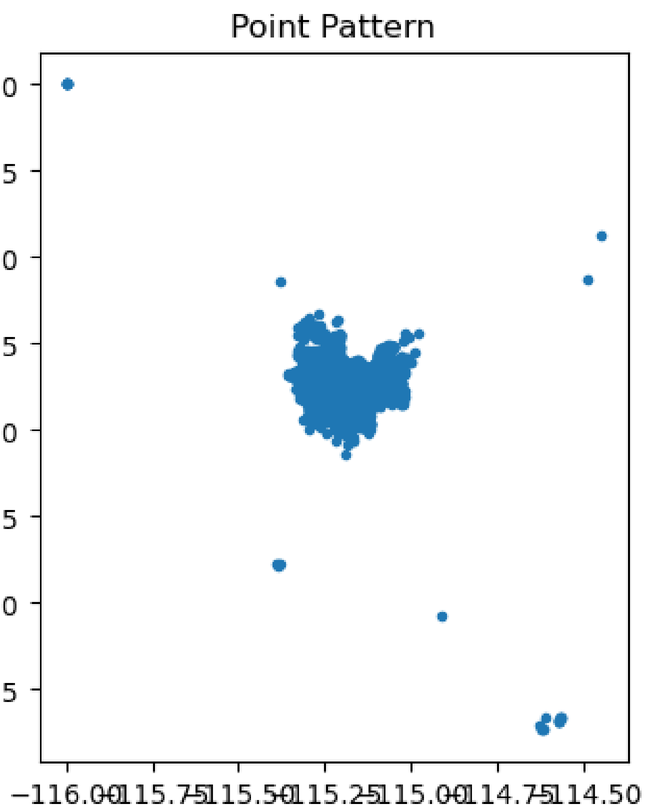
This is a scatter plot. Interesting small gap on the Northwest of the cluster

[19]:
```
type(pp.points)
```

[19]: pandas.core.frame.DataFrame

This lets us know that it is a pandas data frame and not a list of coordinates

[20]:
```
pp.plot()
```



Here we use built in features to make a point pattern visual. Some outlier points are more visable on the outskirts of the plot indicating rual crime

[21]:
```
from pointpats.centrography import (hull, mbr, mean_center,
                                    weighted_mean_center, manhattan_median,
                                    std_distance,euclidean_median,ellipse)
```

Pulls functions from pointpats.centrography to do geo spatial analysis with

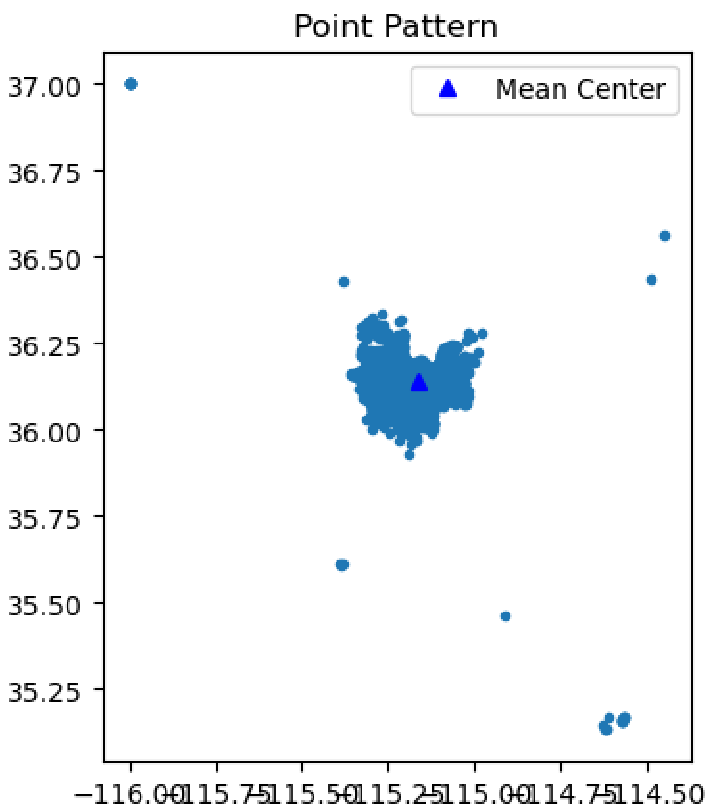
[22]:
```
mc = mean_center(pp.points)
mc
```

[22]:
```
array([-115.16582754,   36.14303413])
```

This is the geographic center of robbery incidents

[23]:
```
pp.plot()
plt.plot(mc[0], mc[1], 'b^', label='Mean Center')
plt.legend(numpoints=1)
```

[23]:
```
<matplotlib.legend.Legend at 0x750f640094b0>
```



The blue triangle represents the mean center which is the middle location of the points. Robbery points are the blue dots. The mean center appears a little north of central Las Vegas.

[24]:
```
import numpy as np
from pointpats import PointPattern, weighted_mean_center
import matplotlib.pyplot as plt
import geopandas as gpd

points = lv_robb_gdf[['Longitude', 'Latitude']].values.tolist()

weights = np.arange(12)
print("Weights:", weights)

if len(weights) != len(points):
    print("Weights array and points list must have the same length.")
else:
    pp = PointPattern(points)

    wmc = weighted_mean_center(pp.points, weights)
    print("Weighted Mean Center:", wmc)
```

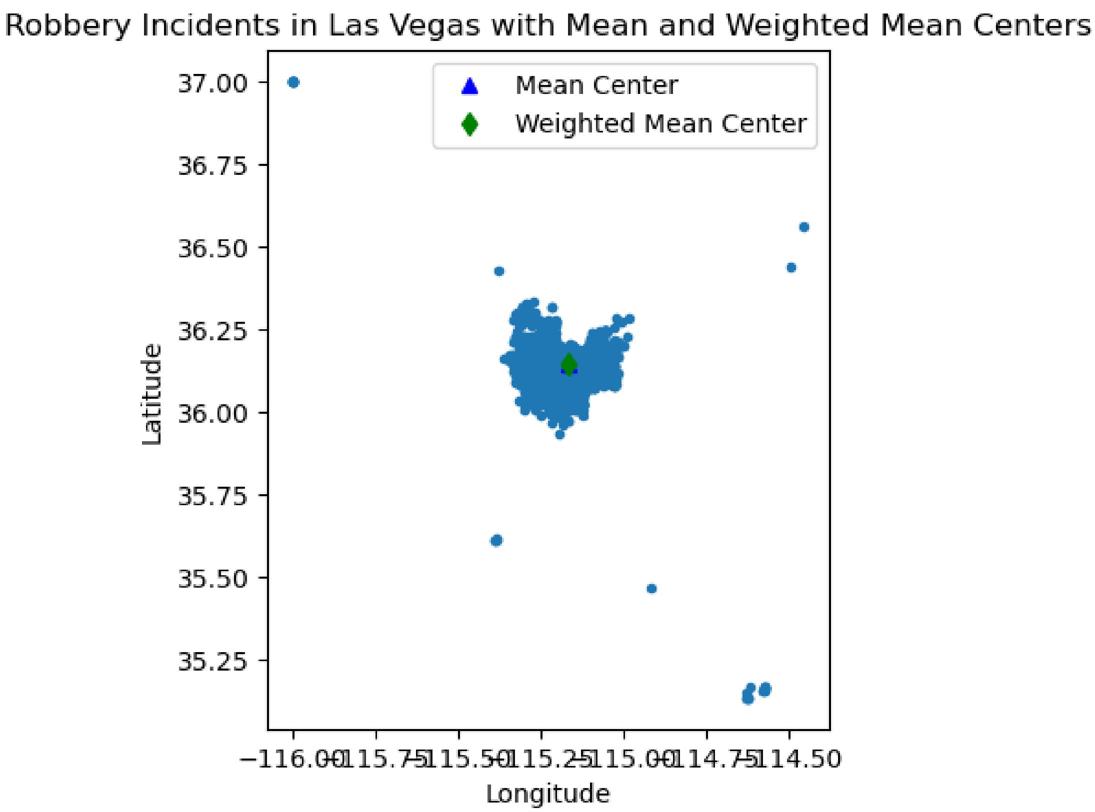
```
    plt.figure(figsize=(10, 8))
    plt.scatter(*zip(*points), c='blue', marker='o', edgecolor='k',␣
 ↪label='Robbery Incidents')
    plt.scatter(wmc[0], wmc[1], c='red', marker='x', s=100, label='Weighted␣
 ↪Mean Center')
    plt.title('Robbery Incidents in Las Vegas with Weighted Mean Center')
    plt.xlabel('Longitude')
    plt.ylabel('Latitude')
    plt.legend()
    plt.show()
```

```
Weights: [ 0  1  2  3  4  5  6  7  8  9 10 11]
Weights array and points list must have the same length.
```

Adds to calculating the mean center

[25]:
```
import numpy as np
from pointpats import PointPattern, mean_center, weighted_mean_center
import matplotlib.pyplot as plt
import geopandas as gpd

points = lv_robb_gdf[['Longitude', 'Latitude']].values.tolist()

weights = np.arange(1, len(points) + 1)

pp = PointPattern(points)


mc = mean_center(pp.points)
wmc = weighted_mean_center(pp.points, weights)

plt.figure(figsize=(10, 8))
pp.plot()
plt.plot(mc[0], mc[1], 'b^', label='Mean Center')
plt.plot(wmc[0], wmc[1], 'gd', label='Weighted Mean Center')
plt.legend(numpoints=1)
plt.title('Robbery Incidents in Las Vegas with Mean and Weighted Mean Centers')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
```

```
<Figure size 1000x800 with 0 Axes>
```



The weighted mean center accounts for the importance or weight of each point. This data set is just points of locations where a robbery related crime took place. This does not have any weight on it so it is the same as the mean center.

[26]:
```
pp.n
```

[26]: 3808

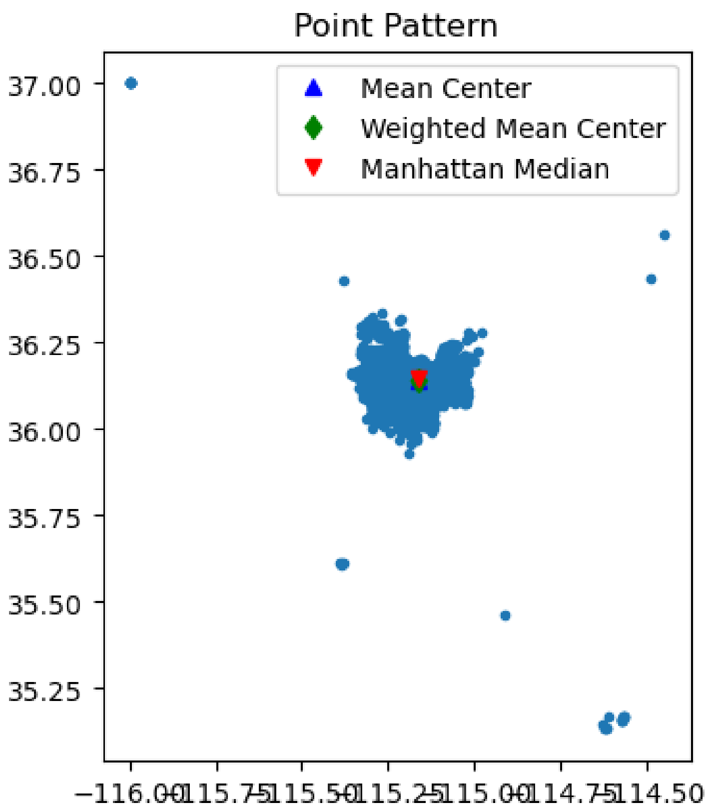
[27]:
```
mm = manhattan_median(pp.points)
mm
```

```
/opt/tljh/user/lib/python3.10/site-packages/pointpats/centrography.py:214:
UserWarning: Manhattan Median is not unique for even point patterns.
  warnings.warn(s)
```

[27]: array([-115.1629425,   36.144684])

[28]:
```
pp.plot()
plt.plot(mc[0], mc[1], 'b^', label='Mean Center')
plt.plot(wmc[0], wmc[1], 'gd', label='Weighted Mean Center')
plt.plot(mm[0], mm[1], 'rv', label='Manhattan Median')
plt.legend(numpoints=1)
```

[28]: <matplotlib.legend.Legend at 0x750f6405a1a0>



The manhatten median is the minimized sum of the distances to all the other points. It is another way of seeing a central point. The fact all thes points are together means points are symmerically distributted.

[29]:
```
def median_center(points, crit=0.0001):
    points = np.asarray(points)
    x0, y0 = points.mean(axis=0)
    dx = np.inf
    dy = np.inf
    iteration = 0
    while np.abs(dx) > crit or np.abs(dy) > crit:
        xd = points[:, 0] - x0
        yd = points[:, 1] - y0
        d = np.sqrt(xd*xd + yd*yd)
        w = 1./d
        w = w / w.sum()
        x1 = w * points[:, 0]
        x1 = x1.sum()
        y1 = w * points[:, 1]
```

```
        y1 = y1.sum()
        dx = x1 - x0
        dy = y1 - y0
        iteration +=1
        print(x0, x1, dx, dy, d.sum(), iteration)
        x0 = x1
        y0 = y1

    return x1, y1
```

Working on getting the median center

[30]:
```
em = euclidean_median(pp.points)
em
```

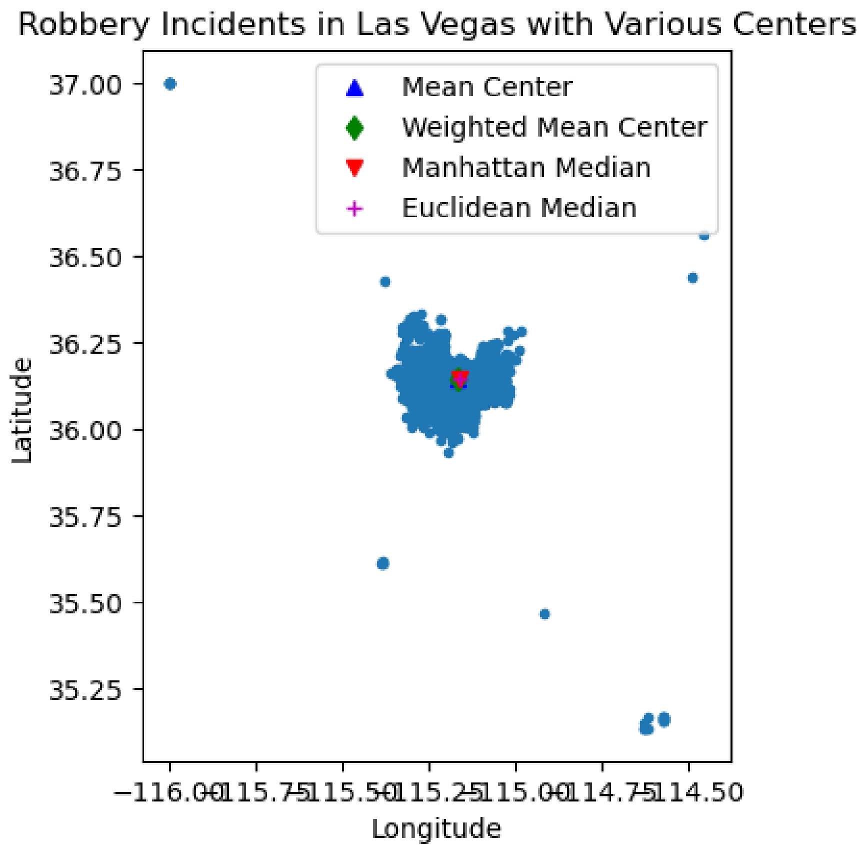
[30]:
```
array([-115.15952488,   36.14148731])
```

Code of the euclidean median

[31]:
```
import numpy as np
from pointpats import PointPattern, mean_center, weighted_mean_center,␣
 ↪euclidean_median
import matplotlib.pyplot as plt

points = lv_robb_gdf[['Longitude', 'Latitude']].values.tolist()
points = np.array(points)

pp = PointPattern(points)

mc = mean_center(pp.points)
weights = np.arange(1, len(points) + 1)
wmc = weighted_mean_center(pp.points, weights)
em = euclidean_median(pp.points)

def manhattan_median(points):
    points = np.asarray(points)
    median_x = np.median(points[:, 0])
    median_y = np.median(points[:, 1])
    return np.array([median_x, median_y])

mm = manhattan_median(pp.points)

plt.figure(figsize=(10, 8))
pp.plot()
plt.plot(mc[0], mc[1], 'b^', label='Mean Center')
plt.plot(wmc[0], wmc[1], 'gd', label='Weighted Mean Center')
plt.plot(mm[0], mm[1], 'rv', label='Manhattan Median')
plt.plot(em[0], em[1], 'm+', label='Euclidean Median')
```

```
plt.legend(numpoints=1)
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Robbery Incidents in Las Vegas with Various Centers')
plt.show()
```

```
<Figure size 1000x800 with 0 Axes>
```



The euclidean mean minimizes the sum of euclidean points to the other points. It is less effected by outliers than other methods. It is the same as the other center methods we had tried indicating symmetry.

[32]:
```
stdd = std_distance(pp.points)
stdd
```

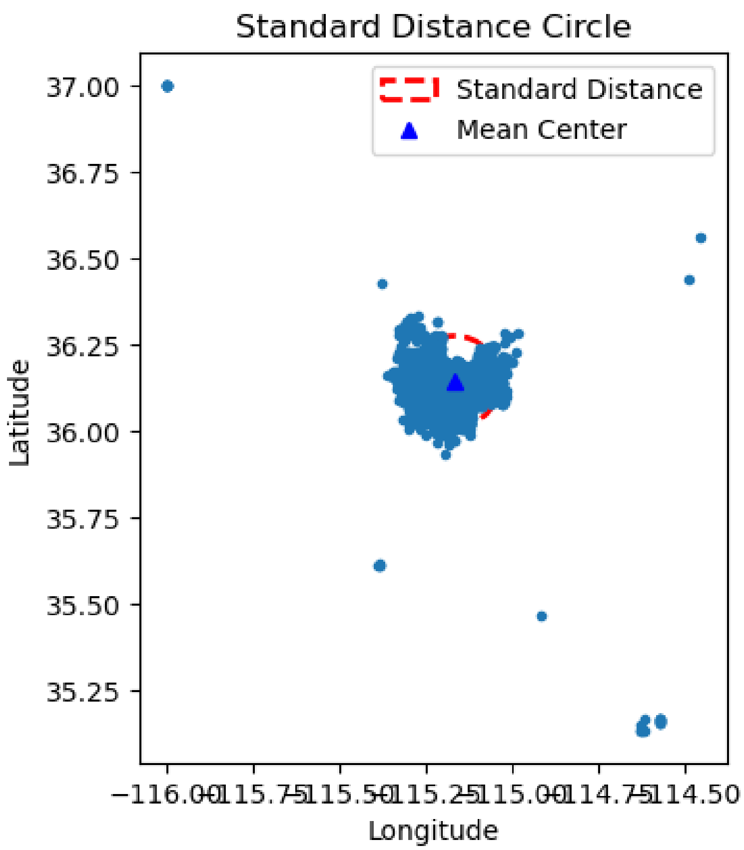
[32]: 0.13115103455048693

The standard distance of the points

Calculating of the standard distance of points

[33]:
```
import numpy as np
from pointpats import PointPattern, mean_center, std_distance
import matplotlib.pyplot as plt
import geopandas as gpd

points = lv_robb_gdf[['Longitude', 'Latitude']].values.tolist()

pp = PointPattern(points)

mc = mean_center(pp.points)

stdd = std_distance(pp.points)

plt.figure(figsize=(10, 8))
ax = pp.plot(get_ax=True, title='Standard Distance Circle')
circle1 = plt.Circle((mc[0], mc[1]), stdd, color='r', fill=False,␣
↪linestyle='--', linewidth=2, label='Standard Distance')
ax.add_artist(circle1)
plt.plot(mc[0], mc[1], 'b^', label='Mean Center')
ax.set_aspect('equal')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.legend(numpoints=1)
plt.show()
```

```
<Figure size 1000x800 with 0 Axes>
```



A standard distance circle can show standard deviation from the mean center. The mean center is the same it is important here that the red part in the southeast and the north of the points. Cold points exits in the red part of the plot.

[34]:
```
sx, sy, theta = ellipse(pp.points)
sx, sy, theta
```

[34]:
```
(0.0991538139932501, 0.09918332297709441, -0.7953565507910745)
```

Calculates a standard deviation ellipse that gives three important values to describe spatial distrubutuion of points

[35]:
```
theta_degree = np.degrees(theta)
theta_degree
```

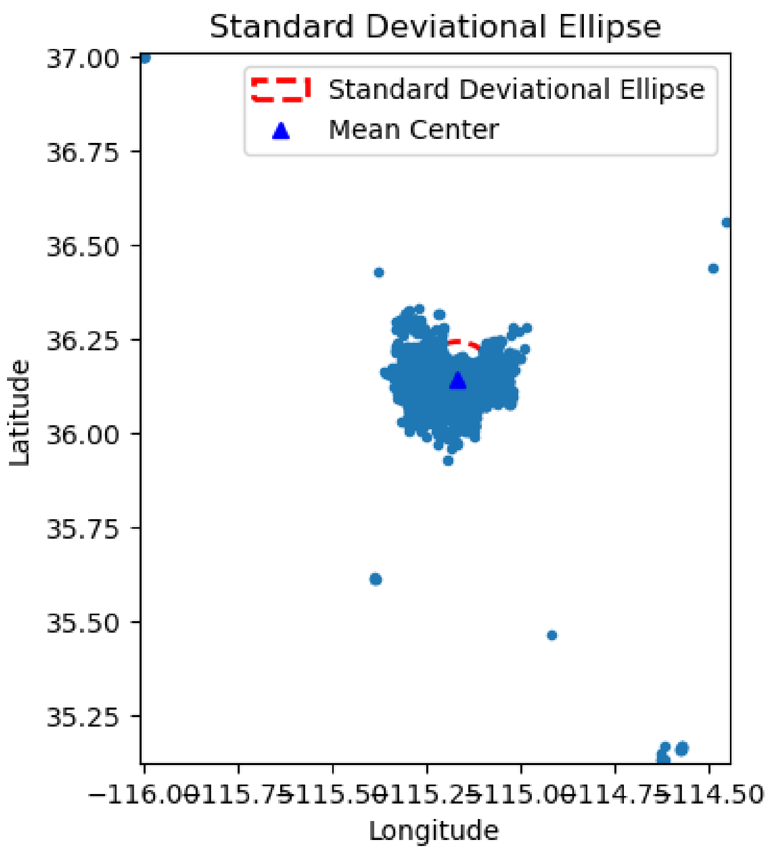
[35]:
```
-45.57057356841106
```

Converts from Radian to degrees to work with the standard deviation ellipse

[36]:
```
import numpy as np
from pointpats import PointPattern, mean_center, std_distance, ellipse
```

```
import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse

points = lv_robb_gdf[['Longitude', 'Latitude']].values.tolist()

pp = PointPattern(points)

mc = mean_center(pp.points)

sx, sy, theta = ellipse(pp.points)
theta_degree = np.degrees(theta)

plt.figure(figsize=(10, 8))
ax = pp.plot(get_ax=True, title='Standard Deviational Ellipse')

e = Ellipse(xy=(mc[0], mc[1]), width=sx*2, height=sy*2, angle=-theta_degree,
            edgecolor='red', facecolor='none', linestyle='--', linewidth=2, label='Standard Deviational Ellipse')
ax.add_artist(e)

ax.set_xlim(lv_robb_gdf['Longitude'].min() - 0.01, lv_robb_gdf['Longitude'].max() + 0.01)
ax.set_ylim(lv_robb_gdf['Latitude'].min() - 0.01, lv_robb_gdf['Latitude'].max() + 0.01)

plt.plot(mc[0], mc[1], 'b^', label='Mean Center')

plt.legend(numpoints=1)
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
```

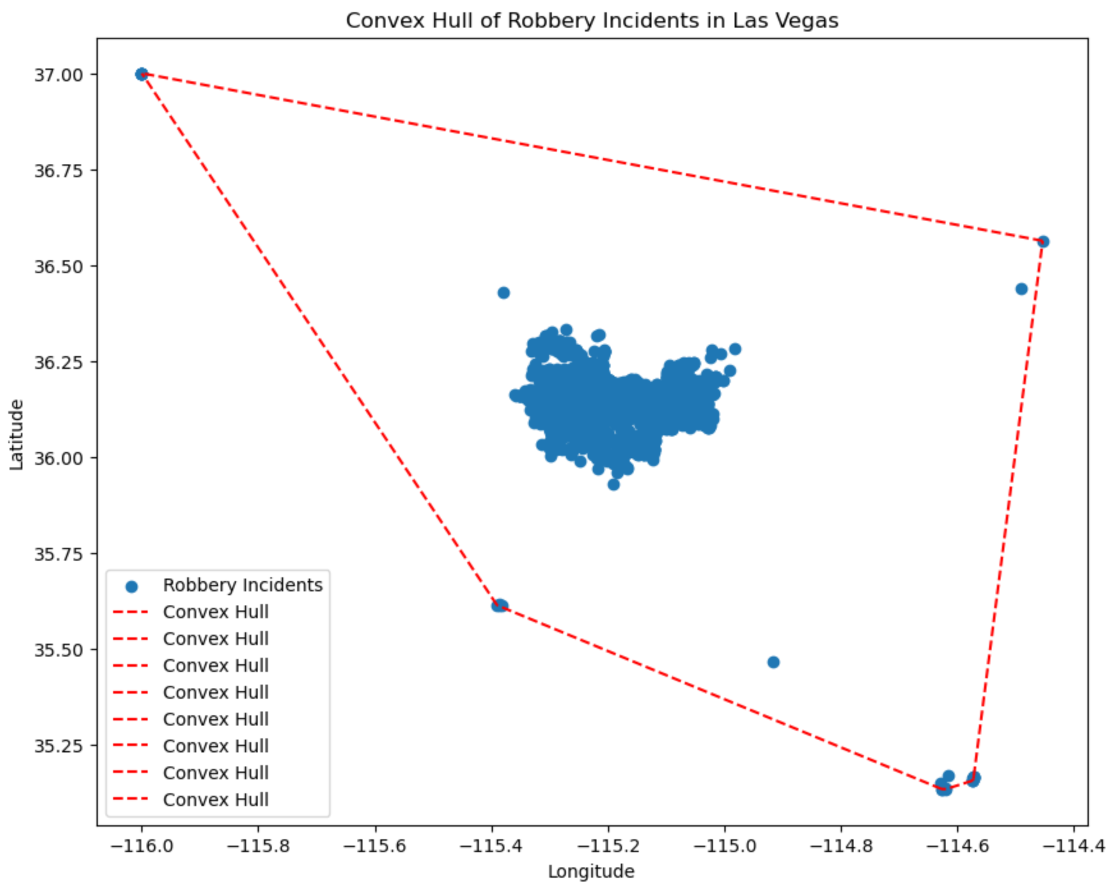
```
<Figure size 1000x800 with 0 Axes>
```



This plot features the standard deviation ellipse, the mean center, and the plots of the robberies. The ellipse illustrates the the data is more spread on the longitude.

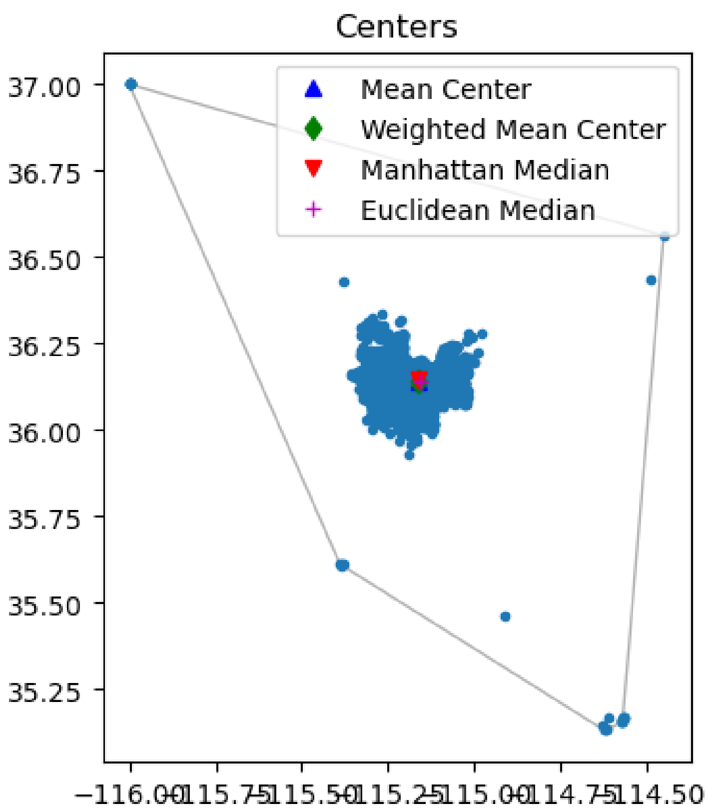
[37]:
```
import numpy as np
from scipy.spatial import ConvexHull
from pointpats import PointPattern
import matplotlib.pyplot as plt
from matplotlib.patches import Polygon as PolygonPatch

points = lv_robb_gdf[['Longitude', 'Latitude']].values.tolist()
points = np.array(points)

hull = ConvexHull(points)

plt.figure(figsize=(10, 8))
plt.scatter(points[:, 0], points[:, 1], label='Robbery Incidents')

for simplex in hull.simplices:
    plt.plot(points[simplex, 0], points[simplex, 1], 'r--', label='Convex Hull')
```

```
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Convex Hull of Robbery Incidents in Las Vegas')
plt.legend()
plt.show()
```



[38]:
```
pp.plot(title='Centers', hull=True )
plt.plot(mc[0], mc[1], 'b^', label='Mean Center')
plt.plot(wmc[0], wmc[1], 'gd', label='Weighted Mean Center')
plt.plot(mm[0], mm[1], 'rv', label='Manhattan Median')
plt.plot(em[0], em[1], 'm+', label='Euclidean Median')
plt.legend(numpoints=1)
```

[38]: <matplotlib.legend.Legend at 0x750f70b71ae0>



As refrenced earlier the centers represent symmetry. There is no real symmatry when it comes to outlier points.

[39]:
```
mbr(pp.points)
```

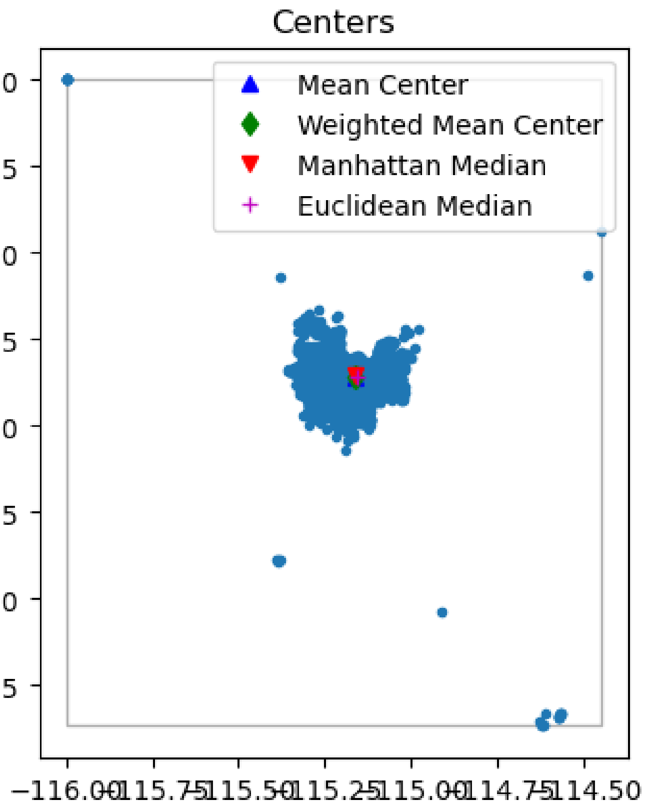
```
/tmp/ipykernel_2370128/2243439823.py:1: FutureWarning: This function will be
deprecated in the next release of pointpats.
  mbr(pp.points)
```

[39]:
```
(-116.0, 35.1329040000001, -114.4534905, 37.0000000000001)
```

The minimum bounding rectangle points

[40]:
```
pp.plot(title='Centers', window=True )
plt.plot(mc[0], mc[1], 'b^', label='Mean Center')
plt.plot(wmc[0], wmc[1], 'gd', label='Weighted Mean Center')
plt.plot(mm[0], mm[1], 'rv', label='Manhattan Median')
plt.plot(em[0], em[1], 'm+', label='Euclidean Median')
plt.legend(numpoints=1)
```

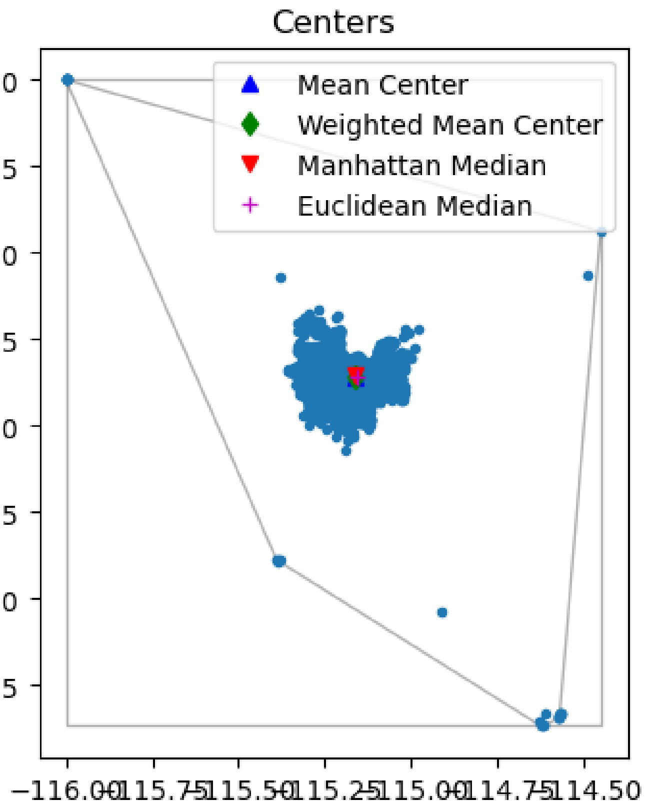
[40]:
```
<matplotlib.legend.Legend at 0x750f57f84340>
```



The rectangle is the minmium bounding rectangle. This is the smallest rectangle that can fit all the points.

[41]:
```
pp.plot(title='Centers',  hull=True , window=True )
plt.plot(mc[0], mc[1], 'b^', label='Mean Center')
plt.plot(wmc[0], wmc[1], 'gd', label='Weighted Mean Center')
plt.plot(mm[0], mm[1], 'rv', label='Manhattan Median')
plt.plot(em[0], em[1], 'm+', label='Euclidean Median')
plt.legend(numpoints=1)
```

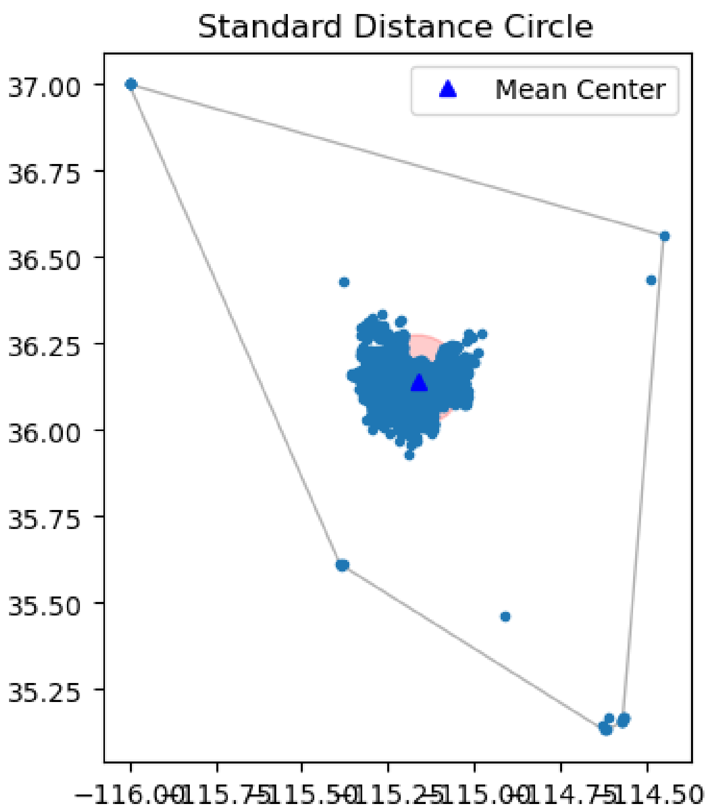
[41]:
```
<matplotlib.legend.Legend at 0x750f57ffc370>
```



This once agains bounds the points but this time it inlcudes a polygon inside of the rectangle. It represents what the rectangle may not show such as the large gap in the southwest circle.

[42]:
```
circle1=plt.Circle((mc[0], mc[1]),stdd,color='r',alpha=0.2)
ax = pp.plot(get_ax=True, title='Standard Distance Circle', hull=True)
ax.add_artist(circle1)
plt.plot(mc[0], mc[1], 'b^', label='Mean Center')
ax.set_aspect('equal')
plt.legend(numpoints=1)
```

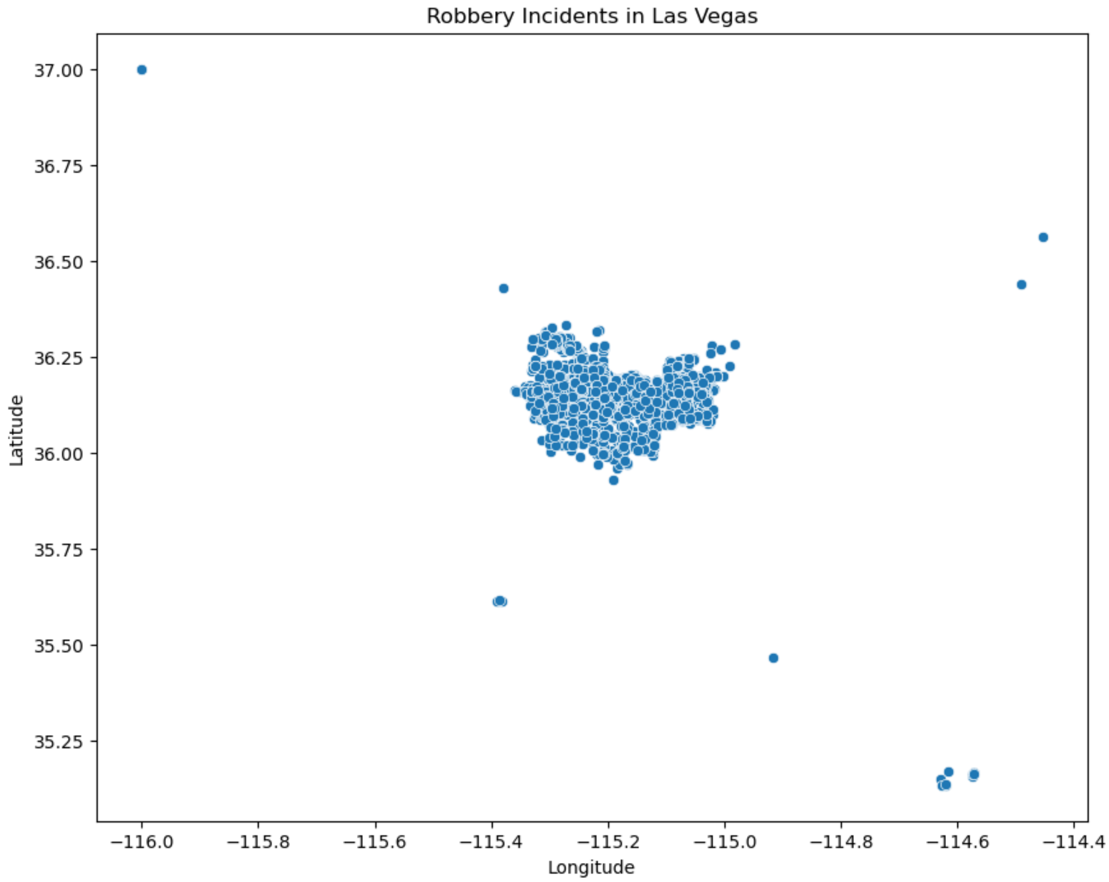
[42]:
```
<matplotlib.legend.Legend at 0x750f655bd030>
```



The red is a cold part and the boundrys are presented in the plot.

Quadrant Statistics

[43]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

df = lv_robb_gdf[['Longitude', 'Latitude']].rename(columns={'Longitude': 'x',␣
↪'Latitude': 'y'})

np.random.seed(12345)

plt.figure(figsize=(10, 8))
sns.scatterplot(x='x', y='y', data=df)
plt.title('Robbery Incidents in Las Vegas')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
```



This is a graph that provides a good overview of the robberys

Morans

[44]:
```
import pandas as pd
import geopandas as gpd
import numpy as np
from esda.moran import Moran
from libpysal.weights import Queen
from shapely.geometry import Point

lv_robb_df = pd.read_csv("Robberies/LV_Robb.csv")

if 'Longitude' not in lv_robb_df.columns or 'Latitude' not in lv_robb_df.
 ↪columns:
    raise ValueError("CSV file must contain 'Longitude' and 'Latitude' columns")

geometry = [Point(xy) for xy in zip(lv_robb_df['Longitude'],␣
 ↪lv_robb_df['Latitude'])]
lv_robb_gdf = gpd.GeoDataFrame(lv_robb_df, geometry=geometry)
```

```
lv_robb_gdf = lv_robb_gdf.set_crs('EPSG:4326')

w = Queen.from_dataframe(lv_robb_gdf)
w.transform = 'r'

lv_robb_gdf['robberies'] = 1

np.random.seed(12345)
mc = Moran(lv_robb_gdf['robberies'], w, transformation='r')

print(f"Moran's I: {mc.I}")
print(f"Expected I: {mc.EI}")
print(f"P-value (normal): {mc.p_norm}")
print(f"P-value (simulated): {mc.p_sim}")
```

```
Moran's I: nan
Expected I: -0.0002626740215392698
P-value (normal): nan
P-value (simulated): 0.001
```

Morans I is the value that shows how much spatial autocorrelation is present in the data set. The expected I is the vlaue that is guessed with the null hypothesis of no spatial autocorrelation. Here the Morans is close to 0 which means with the null hypothesis we do not think it will show spatial clustering or dispersion. The P value (normal) is how correct the test if the distribution is normal.The P value simulated is the test signifcance based on simulated distribution. The nan P valuebeing close to 0 means the most likely there is spatial autocorrelation. For the nan answears there is issues with the morans due to other issues.

[45]:
```
print(lv_robb_gdf.info())
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 3808 entries, 0 to 3807
Data columns (total 23 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   OBJECTID              3808 non-null   int64
 1   Event Number          3808 non-null   object
 2   Reported On Date      3808 non-null   object
 3   Location              3808 non-null   object
 4   CSZ                   3808 non-null   object
 5   Area Command          3808 non-null   object
 6   Beat                  3808 non-null   object
 7   Offense Group         3808 non-null   object
 8   Crime Against         3808 non-null   object
 9   Offense Category      3808 non-null   object
 10  Offense               3808 non-null   object
 11  NIBRS Offense Code    3808 non-null   int64
```

```
 12  Violent Crime           3808 non-null   bool
 13  ShootingVictims         3808 non-null   object
 14  Shooting Victim Count   3808 non-null   int64
 15  Weapons                 3614 non-null   object
 16  Longitude               3808 non-null   float64
 17  Latitude                3808 non-null   float64
 18  Days From Report Ending 3808 non-null   int64
 19  x                       3808 non-null   float64
 20  y                       3808 non-null   float64
 21  geometry                3808 non-null   geometry
 22  robberies               3808 non-null   int64
dtypes: bool(1), float64(4), geometry(1), int64(5), object(12)
memory usage: 658.3+ KB
None
```

Gives information on the LV Robb csv file

[46]:
```python
import pandas as pd
import numpy as np
from scipy.spatial.distance import cdist

df = pd.read_csv('Robberies/LV_Robb.csv')

print(df.columns)

df = df.rename(columns={'Longitude': 'x', 'Latitude': 'y'})

coordinates = df[['x', 'y']].values

distance_matrix = cdist(coordinates, coordinates, metric='euclidean')

distance_matrix_df = pd.DataFrame(distance_matrix, columns=df.index, index=df.
 ↪index)

print(distance_matrix_df)

distance_x = df['x'].values[:, np.newaxis] - df['x'].values
print(distance_x)
```

```
Index(['OBJECTID', 'Event Number', 'Reported On Date', 'Location', 'CSZ',
       'Area Command', 'Beat', 'Offense Group', 'Crime Against',
       'Offense Category', 'Offense', 'NIBRS Offense Code', 'Violent Crime',
       'ShootingVictims', 'Shooting Victim Count', 'Weapons', 'Longitude',
       'Latitude', 'Days From Report Ending', 'x', 'y'],
      dtype='object')
            0         1         2         3         4         5         6     \
0      0.000000  0.090324  0.037686  0.091514  0.103376  0.104062  0.027521
1      0.090324  0.000000  0.072498  0.141337  0.102909  0.116675  0.114532
```

```
2     0.037686  0.072498  0.000000  0.126171  0.067235  0.070845  0.064400
3     0.091514  0.141337  0.126171  0.000000  0.193398  0.195420  0.074913
4     0.103376  0.102909  0.067235  0.193398  0.000000  0.017410  0.127664
…        …        …        …        …        …        …        …
3803  0.268890  0.196742  0.265823  0.254732  0.295548  0.311100  0.282432
3804  0.130034  0.121406  0.093768  0.219933  0.026660  0.032301  0.154150
3805  0.167013  0.135068  0.129405  0.253298  0.067588  0.077261  0.192881
3806  0.149712  0.062605  0.123666  0.202391  0.122079  0.139267  0.175448
3807  0.070362  0.062621  0.033254  0.154817  0.044408  0.055287  0.097492

             7         8         9  …      3798      3799      3800  \
0     0.128067  0.008323  0.108695  …  0.174787  0.096513  0.252992
1     0.090206  0.088594  0.120536  …  0.151185  0.091078  0.237664
2     0.136833  0.030333  0.075575  …  0.137728  0.114375  0.274060
3     0.109969  0.099830  0.200083  …  0.263307  0.070261  0.182065
4     0.188437  0.095238  0.019470  …  0.071995  0.174861  0.332912
…        …        …        … …      …        …        …
3803  0.147954  0.271321  0.314462  …  0.322229  0.188627  0.175722
3804  0.209728  0.121889  0.029725  …  0.046005  0.198952  0.355666
3805  0.225150  0.159302  0.074884  …  0.019609  0.222073  0.372639
3806  0.128743  0.146474  0.141518  …  0.139238  0.144595  0.269975
3807  0.144487  0.063431  0.059536  …  0.108976  0.130679  0.288506

          3801      3802      3803      3804      3805      3806      3807
0     0.049211  0.029482  0.268890  0.130034  0.167013  0.149712  0.070362
1     0.047305  0.081461  0.196742  0.121406  0.135068  0.062605  0.062621
2     0.025335  0.011043  0.265823  0.093768  0.129405  0.123666  0.033254
3     0.124902  0.120267  0.254732  0.219933  0.253298  0.202391  0.154817
4     0.075728  0.073895  0.295548  0.026660  0.067588  0.122079  0.044408
…        …        …        …        …        …        …        …
3803  0.240933  0.272562  0.000000  0.308026  0.302780  0.183027  0.259045
3804  0.100393  0.100555  0.308026  0.000000  0.045162  0.128515  0.068462
3805  0.128941  0.138020  0.302780  0.045162  0.000000  0.119856  0.098522
3806  0.101707  0.134302  0.183027  0.128515  0.119856  0.000000  0.099425
3807  0.031964  0.043722  0.259045  0.068462  0.098522  0.099425  0.000000

[3808 rows x 3808 columns]
[[[ 0.          0.        ]
  [ 0.00773693  0.00773693]
  [-0.01838557 -0.01838557]
  …
  [-0.08618257 -0.08618257]
  [-0.00520057 -0.00520057]
  [-0.02977057 -0.02977057]]

 [[-0.00773693 -0.00773693]
  [ 0.          0.        ]
  [-0.0261225  -0.0261225 ]
```

```
 …
 [-0.0939195  -0.0939195 ]
 [-0.0129375  -0.0129375 ]
 [-0.0375075  -0.0375075 ]]

[[ 0.01838557  0.01838557]
 [ 0.0261225   0.0261225 ]
 [ 0.          0.        ]
 …
 [-0.067797   -0.067797  ]
 [ 0.013185    0.013185  ]
 [-0.011385   -0.011385  ]]

…

[[ 0.08618257  0.08618257]
 [ 0.0939195   0.0939195 ]
 [ 0.067797    0.067797  ]
 …
 [ 0.          0.        ]
 [ 0.080982    0.080982  ]
 [ 0.056412    0.056412  ]]

[[ 0.00520057  0.00520057]
 [ 0.0129375   0.0129375 ]
 [-0.013185   -0.013185  ]
 …
 [-0.080982   -0.080982  ]
 [ 0.          0.        ]
 [-0.02457    -0.02457   ]]

[[ 0.02977057  0.02977057]
 [ 0.0375075   0.0375075 ]
 [ 0.011385    0.011385  ]
 …
 [-0.056412   -0.056412  ]
 [ 0.02457     0.02457   ]
 [ 0.          0.        ]]]
```

This is using the nearest naighbor approach to look at our data. It represents that the minimum non-zero value in each row of the distance matrix. Looking at our results the points are often fairly highly clustered.

[47]:
```
from geosnap import DataStore
import geopandas as gpd

datasets = DataStore()
```

Import to have acssess to more datasets

[48]:
```
from geosnap.io import get_acs
```

Accsess ACS data

[49]:
```
from geosnap import DataStore
```

imports data store

[50]:
```
datasets = DataStore("/srv/data/geosnap")
```

Load in more datasets

[51]:
```
dir(datasets)
```

[51]:
```
['acs',
 'bea_regions',
 'blocks_2000',
 'blocks_2010',
 'blocks_2020',
 'codebook',
 'counties',
 'ejscreen',
 'lodes_codebook',
 'ltdb',
 'msa_definitions',
 'msas',
 'ncdb',
 'nces',
 'seda',
 'show_data_dir',
 'states',
 'tracts_1990',
 'tracts_2000',
 'tracts_2010',
 'tracts_2020']
```

[52]:
```
from geosnap import io as gio
```

Helps us use the spatial data

[53]:
```
import pandas as pd
import geopandas as gpd
import numpy as np
from esda.moran import Moran, Moran_Local
from libpysal.weights import Queen
from shapely.geometry import Point
```

```
lv_robb_df = pd.read_csv("Robberies/LV_Robb.csv")

if 'Longitude' not in lv_robb_df.columns or 'Latitude' not in lv_robb_df.
 ↪columns:
    raise ValueError("CSV file must contain 'Longitude' and 'Latitude' columns")

geometry = [Point(xy) for xy in zip(lv_robb_df['Longitude'],␣
 ↪lv_robb_df['Latitude'])]
lv_robb_gdf = gpd.GeoDataFrame(lv_robb_df, geometry=geometry)
lv_robb_gdf = lv_robb_gdf.set_crs('EPSG:4326')

w = Queen.from_dataframe(lv_robb_gdf)
w.transform = 'r'

lv_robb_gdf['robberies'] = 1
```

Set up the code for morans and local morans. It loads data, checks collumns, checks geometry, sets crs, creates a spatial weights matrix, and adds a robberies collumn.

[54]:
```
np.random.seed(12345)

mc = Moran(lv_robb_gdf['robberies'], w, transformation='r')

print(f"Moran's I: {mc.I}")
print(f"Expected I: {mc.EI}")
print(f"P-value (normal): {mc.p_norm}")
print(f"P-value (simulated): {mc.p_sim}")
```

```
Moran's I: nan
Expected I: -0.0002626740215392698
P-value (normal): nan
P-value (simulated): 0.001
```

The morans I is non indicating no varability. The expected I is to help us understand the morans I. The p value is a nan indicating no varability. The simulated p value is off of permutation simulations that can give a solid signifcance test. Here the simulated P value indicates that this is not randomly distibuted. It indicates distinct clusters.

[55]:
```
li = Moran_Local(lv_robb_gdf['robberies'], w)


print(f"Local Moran's I values:\n{li.Is}")
print(f"P-values:\n{li.p_sim}")
```

```
Local Moran's I values:
[nan nan nan … nan nan nan]
P-values:
[0.001 0.001 0.001 … 0.001 0.001 0.001]
```

The nan values may be from the robberies does not have variance which certain parts of Las Vegas or there are some points with no naighbors. The P values being so low means the statistics are accurate. This is evidence that there is likely spatial autocorrelation. Due to it being all nan values we are unable to get a plot.

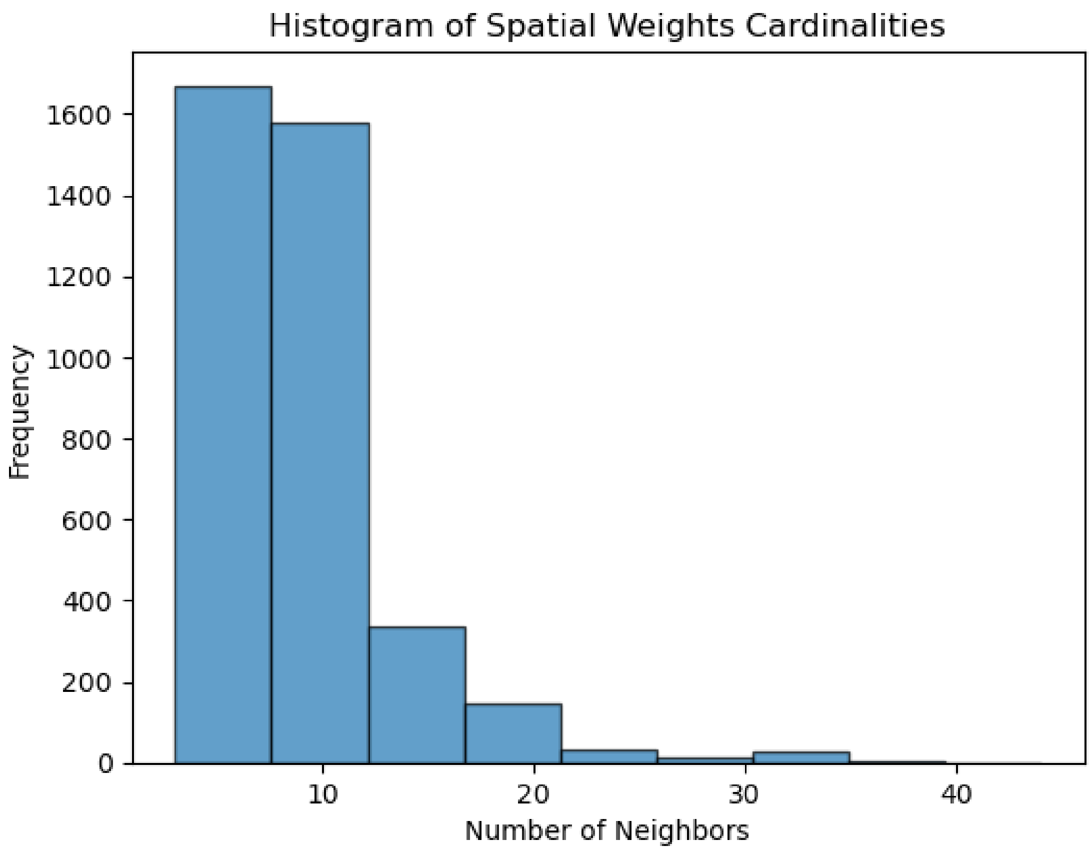
[56]:
```
print(li.Is)
print(np.isnan(li.Is).sum())
```

```
[nan nan nan … nan nan nan]
3808
```

Inspects the morans and finds that 3808 are nan which is all of the points

[57]:
```
import pandas as pd
import geopandas as gpd
import numpy as np
import matplotlib.pyplot as plt
from esda.moran import Moran_Local
from libpysal.weights import Queen
from shapely.geometry import Point

lv_robb_df = pd.read_csv("Robberies/LV_Robb.csv")

if 'Longitude' not in lv_robb_df.columns or 'Latitude' not in lv_robb_df.
 ↪columns:
    raise ValueError("CSV file must contain 'Longitude' and 'Latitude' columns")

geometry = [Point(xy) for xy in zip(lv_robb_df['Longitude'],␣
 ↪lv_robb_df['Latitude'])]
lv_robb_gdf = gpd.GeoDataFrame(lv_robb_df, geometry=geometry)


lv_robb_gdf = lv_robb_gdf.set_crs('EPSG:4326')

w = Queen.from_dataframe(lv_robb_gdf)
w.transform = 'r'

lv_robb_gdf['robberies'] = 1  #

np.random.seed(12345)
local_moran = Moran_Local(lv_robb_gdf['robberies'], w)

lv_robb_gdf['local_moran_I'] = local_moran.Is
lv_robb_gdf['p_value'] = local_moran.p_sim
lv_robb_gdf['significant'] = local_moran.p_sim < 0.05

pd.Series(w.cardinalities).plot.hist(bins=9, edgecolor='k', alpha=0.7)
plt.title('Histogram of Spatial Weights Cardinalities')
```

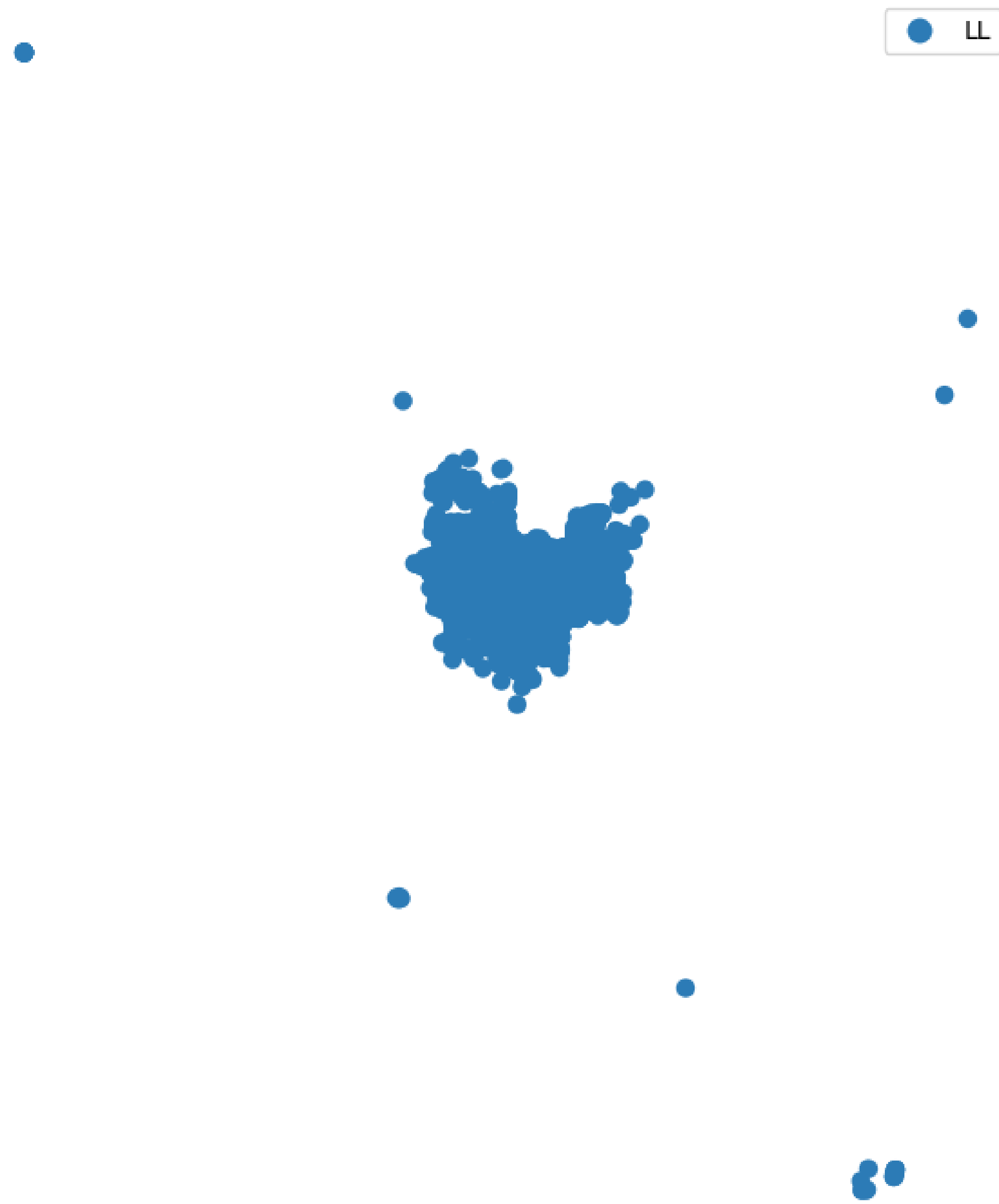
```
plt.xlabel('Number of Neighbors')
plt.ylabel('Frequency')
plt.show()
```



As we can see by this histogram the majority of the points have 0-20 naighbors. It is an urban area so it makes sense for this to be the case due to the density of Las Vegas. The areas with the most robberies are the ones with the most naighbors. This is evidence of clustering.

[58]:
```
from splot.esda import lisa_cluster
lv_robb_df = pd.read_csv("Robberies/LV_Robb.csv")


if 'Longitude' not in lv_robb_df.columns or 'Latitude' not in lv_robb_df.
 ↪columns:
    raise ValueError("CSV file must contain 'Longitude' and 'Latitude' columns")


geometry = [Point(xy) for xy in zip(lv_robb_df['Longitude'],␣
 ↪lv_robb_df['Latitude'])]
lv_robb_gdf = gpd.GeoDataFrame(lv_robb_df, geometry=geometry)
```

```
lv_robb_gdf = lv_robb_gdf.set_crs('EPSG:4326')


w = Queen.from_dataframe(lv_robb_gdf)
w.transform = 'r'

lv_robb_gdf['robberies'] = 1

np.random.seed(12345)
local_moran = Moran_Local(lv_robb_gdf['robberies'], w)


lv_robb_gdf['local_moran_I'] = local_moran.Is
lv_robb_gdf['p_value'] = local_moran.p_sim
lv_robb_gdf['significant'] = local_moran.p_sim < 0.05

lisa_cluster(local_moran, lv_robb_gdf, p=0.05, figsize=(9,9))
plt.show()
```



Lisa cluster that shows all of the plots. I good overview of the points.

[59]:
```
import matplotlib.pyplot as plt
from shapely.geometry import Point
import folium
from folium.plugins import HeatMap
```

```
lv_robb_df = pd.read_csv("Robberies/LV_Robb.csv")

if 'Longitude' not in lv_robb_df.columns or 'Latitude' not in lv_robb_df.
↪columns:
    raise ValueError("CSV file must contain 'Longitude' and 'Latitude' columns")

geometry = [Point(xy) for xy in zip(lv_robb_df['Longitude'],␣
↪lv_robb_df['Latitude'])]
lv_robb_gdf = gpd.GeoDataFrame(lv_robb_df, geometry=geometry)

lv_robb_gdf = lv_robb_gdf.set_crs('EPSG:4326')

m = folium.Map(location=[lv_robb_gdf.geometry.y.mean(), lv_robb_gdf.geometry.x.
↪mean()], zoom_start=12)

heat_data = [[point.xy[1][0], point.xy[0][0]] for point in lv_robb_gdf.geometry]

HeatMap(heat_data).add_to(m)

m.save("lv_robb_heatmap.html")
m
```

[59]: <folium.folium.Map at 0x750f57ac7430>

The heatmap represents that the most of the Robberies are in urban Las Vegas as opposed to rual parts. When zoomed out the robberies appear to be normally distributed.

[60]:
```
from pointpats import (
    distance_statistics,
    QStatistic,
    random,
    PointPattern,
)
```

Point Pats gives more tools to work on the data.

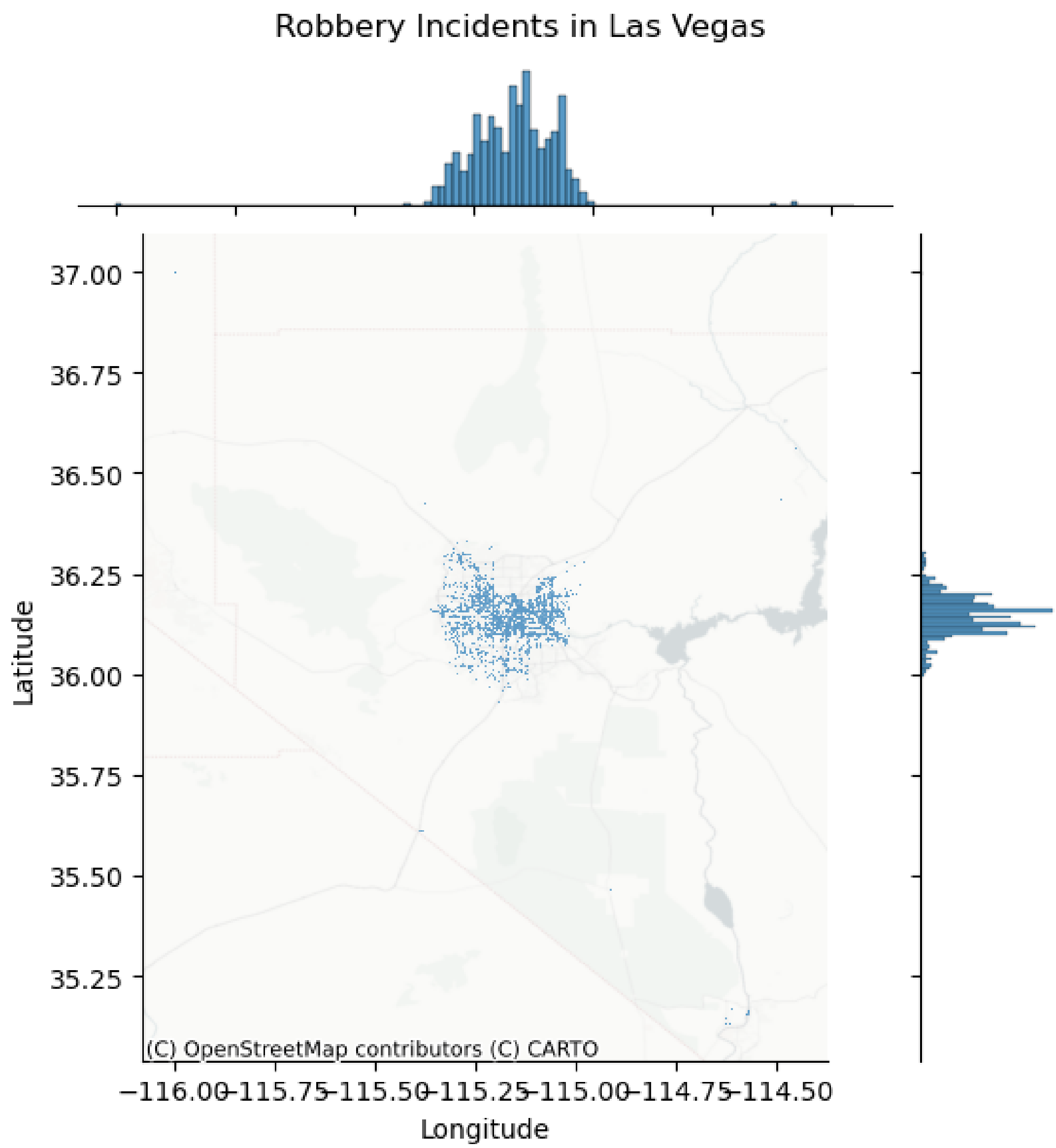
[61]:
```
import pandas as pd
db = pd.read_csv("Robberies/LV_Robb.csv")
db.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3808 entries, 0 to 3807
Data columns (total 21 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   OBJECTID                  3808 non-null   int64
 1   Event Number              3808 non-null   object
```

```
 2   Reported On Date         3808 non-null   object
 3   Location                 3808 non-null   object
 4   CSZ                      3808 non-null   object
 5   Area Command             3808 non-null   object
 6   Beat                     3808 non-null   object
 7   Offense Group            3808 non-null   object
 8   Crime Against            3808 non-null   object
 9   Offense Category         3808 non-null   object
 10  Offense                  3808 non-null   object
 11  NIBRS Offense Code       3808 non-null   int64
 12  Violent Crime            3808 non-null   bool
 13  ShootingVictims          3808 non-null   object
 14  Shooting Victim Count    3808 non-null   int64
 15  Weapons                  3614 non-null   object
 16  Longitude                3808 non-null   float64
 17  Latitude                 3808 non-null   float64
 18  Days From Report Ending  3808 non-null   int64
 19  x                        3808 non-null   float64
 20  y                        3808 non-null   float64
dtypes: bool(1), float64(4), int64(4), object(12)
memory usage: 598.8+ KB
```

This gives us a summary for the data frame

[62]:
```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import contextily as ctx

lv_robb = pd.read_csv('Robberies/LV_Robb.csv')

joint_axes = sns.jointplot(
    x="Longitude", y="Latitude", data=lv_robb, kind="scatter", s=0.5
)

plt.suptitle('Robbery Incidents in Las Vegas', y=1.02)
joint_axes.set_axis_labels('Longitude', 'Latitude')

ctx.add_basemap(
    joint_axes.ax_joint,
    crs="EPSG:4326",
    source=ctx.providers.CartoDB.PositronNoLabels
)

plt.show()
```



This is a scatter plot over a basemap

[63]:
```
import pandas as pd
import geopandas as gpd
from shapely.geometry import Point

lv_robb = pd.read_csv('Robberies/LV_Robb.csv')

robberies = lv_robb[lv_robb['Offense Category'] == 'Robbery']
```

```
geometry = [Point(xy) for xy in zip(robberies['Longitude'],␣
 ↪robberies['Latitude'])]
robberies_gdf = gpd.GeoDataFrame(robberies, geometry=geometry)

robberies_gdf.set_crs('EPSG:4326', inplace=True)
```

[63]:
```
Empty GeoDataFrame
Columns: [OBJECTID, Event Number, Reported On Date, Location, CSZ, Area Command,
Beat, Offense Group, Crime Against, Offense Category, Offense, NIBRS Offense
Code, Violent Crime, ShootingVictims, Shooting Victim Count, Weapons, Longitude,
Latitude, Days From Report Ending, x, y, geometry]
Index: []

[0 rows x 22 columns]
```

Making sure all of the data is in order

[64]:
```
import pandas as pd

robbery_data = pd.read_csv('Robberies/LV_Robb.csv')
population_data = pd.read_csv('census/?Census?.csv')


print(robbery_data.columns)
print(population_data.columns)
```

```
Index(['OBJECTID', 'Event Number', 'Reported On Date', 'Location', 'CSZ',
       'Area Command', 'Beat', 'Offense Group', 'Crime Against',
       'Offense Category', 'Offense', 'NIBRS Offense Code', 'Violent Crime',
       'ShootingVictims', 'Shooting Victim Count', 'Weapons', 'Longitude',
       'Latitude', 'Days From Report Ending', 'x', 'y'],
      dtype='object')
Index(['Label (Grouping)', 'Las Vegas city, Nevada!!Total!!Estimate',
       'Las Vegas city, Nevada!!Total!!Margin of Error',
       'Las Vegas city, Nevada!!Percent!!Estimate',
       'Las Vegas city, Nevada!!Percent!!Margin of Error',
       'Las Vegas city, Nevada!!Male!!Estimate',
       'Las Vegas city, Nevada!!Male!!Margin of Error',
       'Las Vegas city, Nevada!!Percent Male!!Estimate',
       'Las Vegas city, Nevada!!Percent Male!!Margin of Error',
       'Las Vegas city, Nevada!!Female!!Estimate',
       'Las Vegas city, Nevada!!Female!!Margin of Error',
       'Las Vegas city, Nevada!!Percent Female!!Estimate',
       'Las Vegas city, Nevada!!Percent Female!!Margin of Error'],
      dtype='object')
```

Prints columns of both data sets to prepare spatial join

Population / census data for spatial join

[65]:
```python
import pandas as pd
import geopandas as gpd
from shapely.geometry import Point

robbery_data = pd.read_csv('Robberies/LV_Robb.csv')

robberies = robbery_data[robbery_data['Offense Category'] == 'Robbery']

geometry = [Point(xy) for xy in zip(robberies['Longitude'],␣
 ↪robberies['Latitude'])]
robberies_gdf = gpd.GeoDataFrame(robberies, geometry=geometry)

robberies_gdf.set_crs('EPSG:4326', inplace=True)
```

[65]:
```
Empty GeoDataFrame
Columns: [OBJECTID, Event Number, Reported On Date, Location, CSZ, Area Command,
Beat, Offense Group, Crime Against, Offense Category, Offense, NIBRS Offense
Code, Violent Crime, ShootingVictims, Shooting Victim Count, Weapons, Longitude,
Latitude, Days From Report Ending, x, y, geometry]
Index: []

[0 rows x 22 columns]
```

Filtering robberies and create a geo data frame setting the cordinate system

[66]:
```python
population_data = pd.read_csv('census/?Census?.csv')

population_data.rename(columns={
    'Las Vegas city, Nevada!!Total!!Estimate': 'population'
}, inplace=True)
```

Rename collumns to make it easier to work with

[67]:
```python
import pandas as pd
import geopandas as gpd
from shapely.geometry import Point

robbery_data = pd.read_csv('Robberies/LV_Robb.csv')



robberies = robbery_data[robbery_data['Offense Category'] == 'Robbery']

geometry = [Point(xy) for xy in zip(robberies['Longitude'],␣
 ↪robberies['Latitude'])]
robberies_gdf = gpd.GeoDataFrame(robberies, geometry=geometry)

robberies_gdf.set_crs('EPSG:4326', inplace=True)
```

[67]: Empty GeoDataFrame
Columns: [OBJECTID, Event Number, Reported On Date, Location, CSZ, Area Command, Beat, Offense Group, Crime Against, Offense Category, Offense, NIBRS Offense Code, Violent Crime, ShootingVictims, Shooting Victim Count, Weapons, Longitude, Latitude, Days From Report Ending, x, y, geometry]
Index: []

[0 rows x 22 columns]

Filtered to robberies and made a geodata frame

100,000 Groups

[68]:
```
population_data = pd.read_csv('census/?Census?.csv')

population_data['population'] = pd.to_numeric(population_data['Las Vegas city,␣
↪Nevada!!Total!!Estimate'], errors='coerce')

population_data.dropna(subset=['population'], inplace=True)
```

Got the data in and cleaned the population data, made the population estimates numeric values then took away the rows without proper population data.

[69]:
```
grouped_data = []
current_group = []
current_population = 0

for index, row in population_data.iterrows():
    if current_population + row['population'] > 100000:
        grouped_data.append(current_group)
        current_group = []
        current_population = 0
    current_group.append(row)
    current_population += row['population']
if current_group:
    grouped_data.append(current_group)

geometries = []
for group in grouped_data:
    points = [Point(xy) for xy in zip(robberies['Longitude'],␣
↪robberies['Latitude']) if any(robberies['Offense Category'] == row['Label␣
↪(Grouping)'] for row in group)]
    if points:
        centroid = gpd.GeoSeries(points).unary_union.centroid
        geometries.append(centroid)

centroid_gdf = gpd.GeoDataFrame(geometry=geometries)
```

```
centroid_gdf.set_crs('EPSG:4326', inplace=True)

print(centroid_gdf.head())

centroid_gdf.to_file('centroid_groups.shp')
```

```
Empty GeoDataFrame
Columns: [geometry]
Index: []
```

Sorts the population into 100,000 and make centroids and saves the centroids to a shapefile

Sptial Join

[70]:
```
import pandas as pd
import geopandas as gpd
from shapely.geometry import Point

lv_robb_df = pd.read_csv("Robberies/LV_Robb.csv")

if 'Longitude' not in lv_robb_df.columns or 'Latitude' not in lv_robb_df.
↪columns:
    raise ValueError("CSV file must contain 'Longitude' and 'Latitude' columns")

geometry = [Point(xy) for xy in zip(lv_robb_df['Longitude'],␣
↪lv_robb_df['Latitude'])]
lv_robb_gdf = gpd.GeoDataFrame(lv_robb_df, geometry=geometry)

lv_robb_gdf = lv_robb_gdf.set_crs('EPSG:4326')
```

Make sure everything is proper for a spatial join

[71]:
```
census_data = pd.read_csv('census/?Census?.csv')

census_data['population'] = pd.to_numeric(census_data['Las Vegas city, Nevada!!
↪Total!!Estimate'], errors='coerce')

census_data.dropna(subset=['population'], inplace=True)
```

Coverted census estimates to numeric values and gets rid of missing population data

[72]:
```
grouped_data = []
current_group = []
current_population = 0

for index, row in census_data.iterrows():
    if current_population + row['population'] > 100000:
        grouped_data.append(current_group)
        current_group = []
```

```
        current_population = 0
    current_group.append(row)
    current_population += row['population']
if current_group:
    grouped_data.append(current_group)

geometries = []

for group in grouped_data:
    group_points = lv_robb_gdf.sample(n=len(group), random_state=1)

    if not group_points.empty:
        centroid = group_points.geometry.unary_union.centroid
        geometries.append(centroid)

centroid_gdf = gpd.GeoDataFrame(geometry=geometries)

centroid_gdf.set_crs('EPSG:4326', inplace=True)

centroid_gdf['robberies'] = 1
```

Geodata frames of centroids of population groups of 100,000

Polygons

[73]:
```
import geopandas as gpd
import pandas as pd
from shapely.geometry import Point

tracts_url = 'https://www2.census.gov/geo/tiger/TIGER2022/TRACT/
 ↪tl_2022_32_tract.zip'
tracts_gdf = gpd.read_file(tracts_url)

tracts_gdf = tracts_gdf.to_crs('EPSG:4326')
```

Load in a census shapefile and get the coordinates the same that I have been working with

[74]:
```
import zipfile
import os

zip_file_path = 'census/tl_2022_32_tract.zip'
extraction_dir = 'extracted_tracts'

if not os.path.exists(extraction_dir):
    os.makedirs(extraction_dir)

with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extraction_dir)
```

```
print("Extraction completed.")
```

```
Extraction completed.
```

Extracted a zip file with the census tract data

[75]:
```
import geopandas as gpd

tracts_gdf = gpd.read_file(os.path.join(extraction_dir, 'tl_2022_32_tract.shp'))

tracts_gdf = tracts_gdf.to_crs('EPSG:4326')
```

Reporjected the EPSG to the coordinates I want to work with

[76]:
```
import pandas as pd
from shapely.geometry import Point

lv_robb_df = pd.read_csv("Robberies/LV_Robb.csv")

if 'Longitude' not in lv_robb_df.columns or 'Latitude' not in lv_robb_df.
 ↪columns:
    raise ValueError("CSV file must contain 'Longitude' and 'Latitude' columns")

geometry = [Point(xy) for xy in zip(lv_robb_df['Longitude'],␣
 ↪lv_robb_df['Latitude'])]
lv_robb_gdf = gpd.GeoDataFrame(lv_robb_df, geometry=geometry, crs='EPSG:4326')
```

Made the spatial data sturcutred properly

[77]:
```
import geopandas as gpd
import os

extraction_dir = 'extracted_tracts'

tracts_gdf = gpd.read_file(os.path.join(extraction_dir, 'tl_2022_32_tract.shp'))

tracts_gdf = tracts_gdf.to_crs('EPSG:4326')

print(tracts_gdf.head())
```

```
  STATEFP COUNTYFP TRACTCE        GEOID     NAME             NAMELSAD  MTFCC  \
0      32      023  960412  32023960412  9604.12  Census Tract 9604.12  G5020
1      32      031  003519  32031003519    35.19    Census Tract 35.19  G5020
2      32      031  002303  32031002303    23.03    Census Tract 23.03  G5020
3      32      031  003516  32031003516    35.16    Census Tract 35.16  G5020
4      32      031  003520  32031003520    35.20    Census Tract 35.20  G5020

  FUNCSTAT      ALAND   AWATER     INTPTLAT      INTPTLON  \
```

```
0        S  152862412  306262  +36.2390207  -115.9339153
1        S   23848402       0  +39.6864764  -119.7205844
2        S   20036766       0  +39.5533581  -119.9349098
3        S  100816740  855173  +39.6494961  -119.6445694
4        S    5693788   68150  +39.6252419  -119.7092346

                                            geometry
0  POLYGON ((-116.05086 36.29284, -116.0506 36.29…
1  POLYGON ((-119.7613 39.66884, -119.75788 39.66…
2  POLYGON ((-119.96486 39.52666, -119.96486 39.5…
3  POLYGON ((-119.70414 39.6625, -119.7033 39.666…
4  POLYGON ((-119.72799 39.61672, -119.72632 39.6…
```

Plotting the polygons making sure the shapefiles and crs are correct

[78]:
```python
import geopandas as gpd

robberies_in_tracts = gpd.sjoin(lv_robb_gdf, tracts_gdf, how="left",␣
 ↪predicate="within")

print(robberies_in_tracts.head())
```

```
   OBJECTID     Event Number     Reported On Date                Location  \
0    190758  LLV220100001394  1/1/2022 3:11:32 PM       100 Block Fremont St
1    151444  LLV220100001498  1/1/2022 3:48:25 PM     4500 Block PARADISE RD
2    101348  LLV220100008942  1/3/2022 5:50:49 AM    1700 Block E OAKEY BLVD
3    298940  LLV220100009295  1/3/2022 7:09:44 AM  1700 Block N Decatur Blvd
4    298477  LLV220100009802  1/3/2022 9:35:21 AM     4100 Block BOULDER HWY

                 CSZ Area Command Beat Offense Group Crime Against  \
0  LAS VEGAS, NV  89101         DTAC   A1             A      Property
1  LAS VEGAS, NV  89119         CCAC   M3             A      Property
2  LAS VEGAS, NV  89104         DTAC   C4             A      Property
3  LAS VEGAS, NV  89108          BAC   U1             A      Property
4  LAS VEGAS, NV  89121         SEAC   H1             A      Property

  Offense Category  … TRACTCE        GEOID   NAME           NAMELSAD  \
0          ROBBERY  …  000700  32003000700      7     Census Tract 7
1          ROBBERY  …  002604  32003002604  26.04  Census Tract 26.04
2          ROBBERY  …  001402  32003001402  14.02  Census Tract 14.02
3          ROBBERY  …  003423  32003003423  34.23  Census Tract 34.23
4          ROBBERY  …  001608  32003001608  16.08  Census Tract 16.08

   MTFCC FUNCSTAT    ALAND  AWATER     INTPTLAT      INTPTLON
0  G5020        S  1478745       0  +36.1701274  -115.1411884
1  G5020        S  1009369       0  +36.1047800  -115.1571534
2  G5020        S  1746750       0  +36.1478779  -115.1156435
3  G5020        S  1975418       0  +36.1894852  -115.2165348
```

```
4   G5020         S   1288439       0   +36.1396206   -115.0886809

[5 rows x 35 columns]
```

Spatial join and checks the results

[79]:
```python
import geopandas as gpd
import pandas as pd
from shapely.geometry import Point
import os

extraction_dir = 'extracted_tracts'
tracts_gdf = gpd.read_file(os.path.join(extraction_dir, 'tl_2022_32_tract.shp'))
tracts_gdf = tracts_gdf.to_crs('EPSG:4326')

lv_robb_df = pd.read_csv("Robberies/LV_Robb.csv")

if 'Longitude' not in lv_robb_df.columns or 'Latitude' not in lv_robb_df.
↪columns:
    raise ValueError("CSV file must contain 'Longitude' and 'Latitude' columns")

geometry = [Point(xy) for xy in zip(lv_robb_df['Longitude'],␣
↪lv_robb_df['Latitude'])]
lv_robb_gdf = gpd.GeoDataFrame(lv_robb_df, geometry=geometry, crs='EPSG:4326')

robberies_in_tracts = gpd.sjoin(lv_robb_gdf, tracts_gdf, how="left",␣
↪predicate="within")

print(robberies_in_tracts.head())
```

```
   OBJECTID      Event Number      Reported On Date                  Location  \
0    190758  LLV220100001394  1/1/2022 3:11:32 PM       100 Block Fremont St
1    151444  LLV220100001498  1/1/2022 3:48:25 PM     4500 Block PARADISE RD
2    101348  LLV220100008942  1/3/2022 5:50:49 AM    1700 Block E OAKEY BLVD
3    298940  LLV220100009295  1/3/2022 7:09:44 AM  1700 Block N Decatur Blvd
4    298477  LLV220100009802  1/3/2022 9:35:21 AM     4100 Block BOULDER HWY

                 CSZ Area Command Beat Offense Group Crime Against  \
0  LAS VEGAS, NV  89101         DTAC   A1             A      Property
1  LAS VEGAS, NV  89119         CCAC   M3             A      Property
2  LAS VEGAS, NV  89104         DTAC   C4             A      Property
3  LAS VEGAS, NV  89108          BAC   U1             A      Property
4  LAS VEGAS, NV  89121         SEAC   H1             A      Property

  Offense Category  … TRACTCE        GEOID   NAME           NAMELSAD  \
0          ROBBERY  …  000700  32003000700      7      Census Tract 7
1          ROBBERY  …  002604  32003002604  26.04  Census Tract 26.04
2          ROBBERY  …  001402  32003001402  14.02  Census Tract 14.02
```

```
3          ROBBERY  …  003423  32003003423  34.23  Census Tract 34.23
4          ROBBERY  …  001608  32003001608  16.08  Census Tract 16.08

   MTFCC FUNCSTAT    ALAND  AWATER     INTPTLAT      INTPTLON
0  G5020        S  1478745       0  +36.1701274  -115.1411884
1  G5020        S  1009369       0  +36.1047800  -115.1571534
2  G5020        S  1746750       0  +36.1478779  -115.1156435
3  G5020        S  1975418       0  +36.1894852  -115.2165348
4  G5020        S  1288439       0  +36.1396206  -115.0886809

[5 rows x 35 columns]
```

Spatial join to do Morans with

[80]:
```python
print(census_data.columns)
```

```
Index(['Label (Grouping)', 'Las Vegas city, Nevada!!Total!!Estimate',
       'Las Vegas city, Nevada!!Total!!Margin of Error',
       'Las Vegas city, Nevada!!Percent!!Estimate',
       'Las Vegas city, Nevada!!Percent!!Margin of Error',
       'Las Vegas city, Nevada!!Male!!Estimate',
       'Las Vegas city, Nevada!!Male!!Margin of Error',
       'Las Vegas city, Nevada!!Percent Male!!Estimate',
       'Las Vegas city, Nevada!!Percent Male!!Margin of Error',
       'Las Vegas city, Nevada!!Female!!Estimate',
       'Las Vegas city, Nevada!!Female!!Margin of Error',
       'Las Vegas city, Nevada!!Percent Female!!Estimate',
       'Las Vegas city, Nevada!!Percent Female!!Margin of Error',
       'population'],
      dtype='object')
```

[81]:
```python
import geopandas as gpd
import pandas as pd
from shapely.geometry import Point
import os

extraction_dir = 'extracted_tracts'
tracts_gdf = gpd.read_file(os.path.join(extraction_dir, 'tl_2022_32_tract.shp'))
tracts_gdf = tracts_gdf.to_crs('EPSG:4326')

lv_robb_df = pd.read_csv("Robberies/LV_Robb.csv")

if 'Longitude' not in lv_robb_df.columns or 'Latitude' not in lv_robb_df.
  ↪columns:
    raise ValueError("CSV file must contain 'Longitude' and 'Latitude' columns")

geometry = [Point(xy) for xy in zip(lv_robb_df['Longitude'],␣
  ↪lv_robb_df['Latitude'])]
```

```
lv_robb_gdf = gpd.GeoDataFrame(lv_robb_df, geometry=geometry, crs='EPSG:4326')

robberies_in_tracts = gpd.sjoin(lv_robb_gdf, tracts_gdf, how="left",␣
↪predicate="within")

print(robberies_in_tracts.head())
```

```
   OBJECTID      Event Number      Reported On Date                  Location  \
0    190758  LLV220100001394  1/1/2022 3:11:32 PM       100 Block Fremont St
1    151444  LLV220100001498  1/1/2022 3:48:25 PM     4500 Block PARADISE RD
2    101348  LLV220100008942  1/3/2022 5:50:49 AM    1700 Block E OAKEY BLVD
3    298940  LLV220100009295  1/3/2022 7:09:44 AM  1700 Block N Decatur Blvd
4    298477  LLV220100009802  1/3/2022 9:35:21 AM     4100 Block BOULDER HWY

                 CSZ Area Command Beat Offense Group Crime Against  \
0  LAS VEGAS, NV  89101         DTAC   A1             A       Property
1  LAS VEGAS, NV  89119         CCAC   M3             A       Property
2  LAS VEGAS, NV  89104         DTAC   C4             A       Property
3  LAS VEGAS, NV  89108          BAC   U1             A       Property
4  LAS VEGAS, NV  89121         SEAC   H1             A       Property

  Offense Category … TRACTCE        GEOID   NAME             NAMELSAD  \
0          ROBBERY …  000700  32003000700      7       Census Tract 7
1          ROBBERY …  002604  32003002604  26.04  Census Tract 26.04
2          ROBBERY …  001402  32003001402  14.02  Census Tract 14.02
3          ROBBERY …  003423  32003003423  34.23  Census Tract 34.23
4          ROBBERY …  001608  32003001608  16.08  Census Tract 16.08

   MTFCC FUNCSTAT    ALAND  AWATER     INTPTLAT      INTPTLON
0  G5020        S  1478745       0  +36.1701274  -115.1411884
1  G5020        S  1009369       0  +36.1047800  -115.1571534
2  G5020        S  1746750       0  +36.1478779  -115.1156435
3  G5020        S  1975418       0  +36.1894852  -115.2165348
4  G5020        S  1288439       0  +36.1396206  -115.0886809

[5 rows x 35 columns]
```

Spatial Join with approiate code

[82]:
```
tract_robberies = robberies_in_tracts.groupby('GEOID').size().
↪reset_index(name='robberies')

print(tract_robberies.head())
```

```
         GEOID  robberies
0  32003000101         15
1  32003000103         15
2  32003000105         21
```

```
3   32003000106           5
4   32003000107          13
```

Counts the robberies in the tracts

[83]:
```
merged_tracts = tracts_gdf.merge(tract_robberies, on='GEOID', how='left')
merged_tracts['robberies'].fillna(0, inplace=True)

merged_tracts = merged_tracts.merge(census_data[['Label (Grouping)',␣
 ↪'population']], left_on='GEOID', right_on='Label (Grouping)', how='left')

merged_tracts['robbery_rate'] = (merged_tracts['robberies'] /␣
 ↪merged_tracts['population']) * 1000

print(merged_tracts[['GEOID', 'population', 'robberies', 'robbery_rate']].
 ↪head())
```

```
         GEOID  population  robberies  robbery_rate
0  32023960412         NaN        0.0           NaN
1  32031003519         NaN        0.0           NaN
2  32031002303         NaN        0.0           NaN
3  32031003516         NaN        0.0           NaN
4  32031003520         NaN        0.0           NaN
```

Merges with the correct collumns

[84]:
```
merged_tracts = tracts_gdf.merge(tract_robberies, on='GEOID', how='left')
merged_tracts['robberies'].fillna(0, inplace=True)



merged_tracts = merged_tracts.merge(census_data[['Label (Grouping)',␣
 ↪'population']], left_on='GEOID', right_on='Label (Grouping)', how='left')



merged_tracts['robbery_rate'] = (merged_tracts['robberies'] /␣
 ↪merged_tracts['population']) * 1000



print(merged_tracts[['GEOID', 'population', 'robberies', 'robbery_rate']].
 ↪head())
```

```
         GEOID  population  robberies  robbery_rate
0  32023960412         NaN        0.0           NaN
1  32031003519         NaN        0.0           NaN
2  32031002303         NaN        0.0           NaN
3  32031003516         NaN        0.0           NaN
4  32031003520         NaN        0.0           NaN
```

This merges robbery counts to the geo data frame of the cenus tract. Then is merges with population data. Next it caluclates robbery rate for 1000 people then it displays rows.

[85]:
```python
from libpysal.weights import Queen

w = Queen.from_dataframe(merged_tracts)
w.transform = 'r'
```

Creates a spatial weights matrix

[86]:
```python
from esda.moran import Moran

moran = Moran(merged_tracts['robbery_rate'], w)
print(f"Global Moran's I: {moran.I}")
print(f"Expected I: {moran.EI}")
print(f"P-value: {moran.p_norm}")
```

```
Global Moran's I: nan
Expected I: -0.0012853470437017994
P-value: nan
```

This is code for global morans. This expected I makes represents there is no spatial autocorrelation in the data. If the plots where placed randomly it would be close to this value.

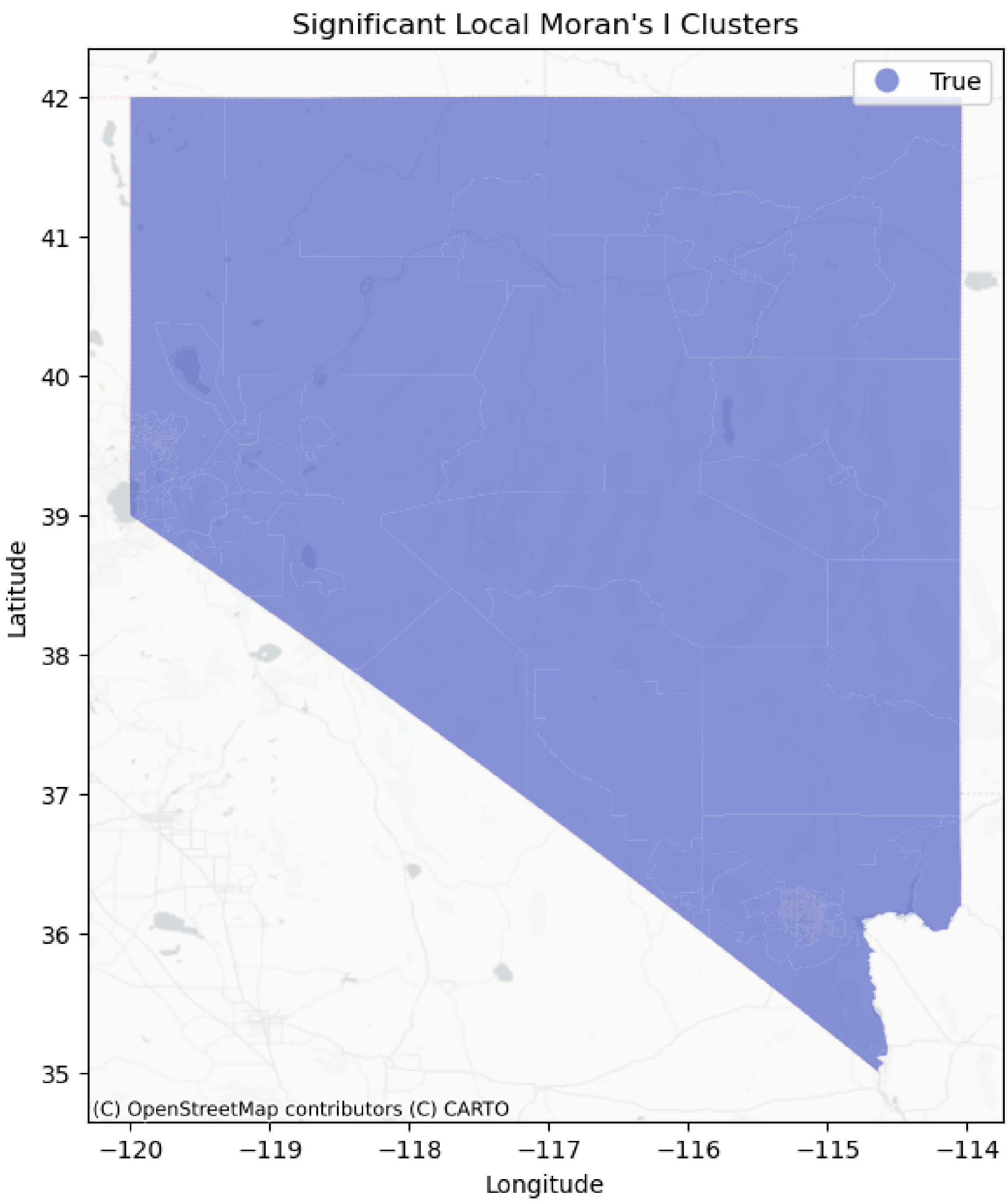
[87]:
```python
from esda.moran import Moran_Local
import numpy as np

np.random.seed(12345)

local_moran = Moran_Local(merged_tracts['robbery_rate'], w)
merged_tracts['local_moran_I'] = local_moran.Is
merged_tracts['p_value'] = local_moran.p_sim

merged_tracts['significant'] = merged_tracts['p_value'] < 0.05

print(merged_tracts[['GEOID', 'local_moran_I', 'p_value', 'significant']].
 ↪head())
```

```
         GEOID  local_moran_I  p_value  significant
0  32023960412            NaN    0.001         True
1  32031003519            NaN    0.001         True
2  32031002303            NaN    0.001         True
3  32031003516            NaN    0.001         True
4  32031003520            NaN    0.001         True
```

This is the code for local morans. Significant clustering is indicated but there is nan values.

[88]:
```python
import matplotlib.pyplot as plt
import contextily as ctx
```

```
fig, ax = plt.subplots(1, 1, figsize=(10, 8))
merged_tracts.plot(column='significant', cmap='coolwarm', legend=True, alpha=0.
    ↪6, ax=ax)
ctx.add_basemap(ax, crs=merged_tracts.crs.to_string(), source=ctx.providers.
    ↪CartoDB.PositronNoLabels)

plt.title("Significant Local Moran's I Clusters")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.show()
```

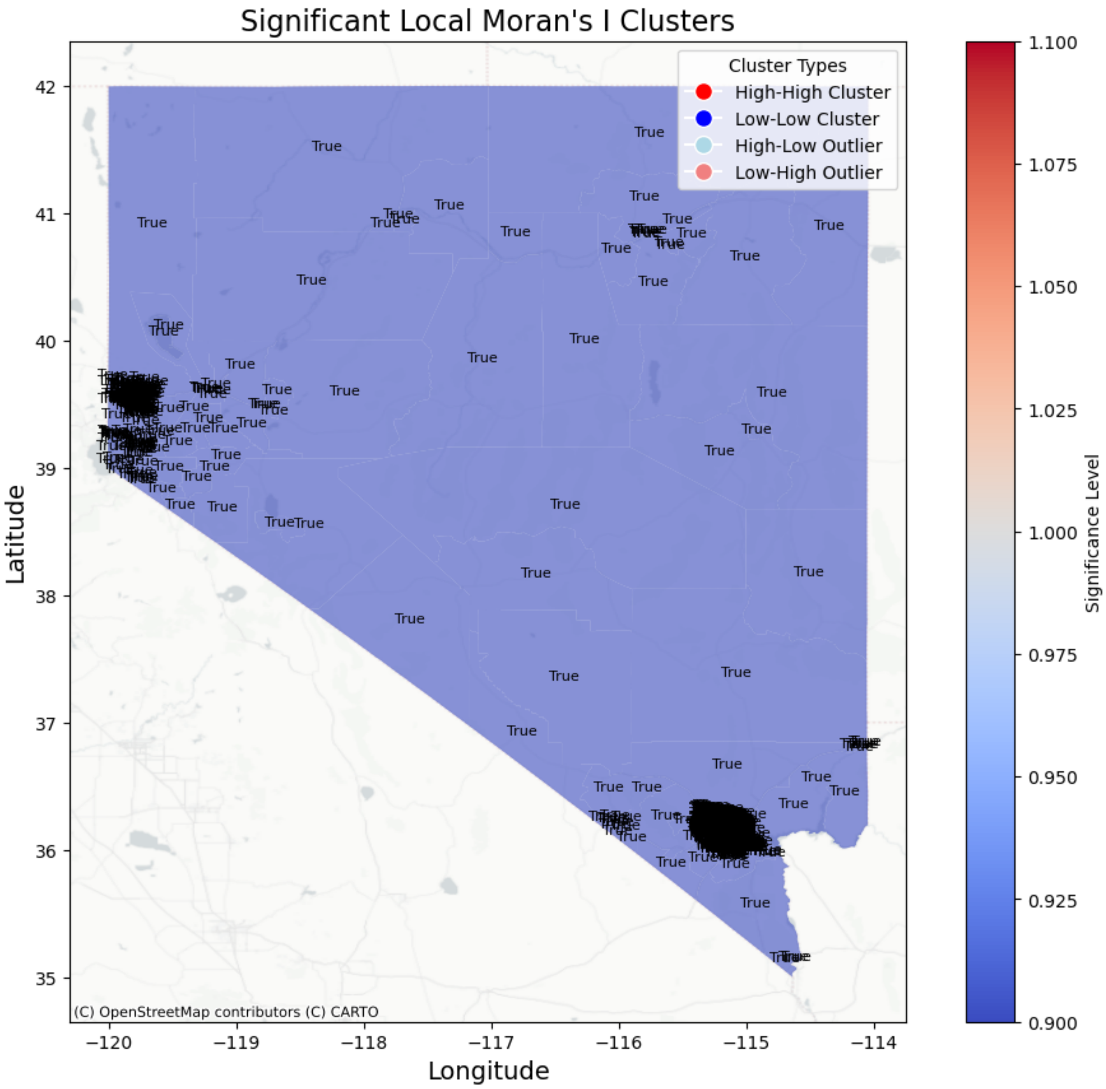


Significant clusters are plotted and the whole state is signifcant

More Detailed Lisa Map

[91]:
```
import matplotlib.pyplot as plt
import contextily as ctx
import geopandas as gpd
```

```
legend_labels = {
    1: 'High-High Cluster',
    2: 'Low-Low Cluster',
    3: 'High-Low Outlier',
    4: 'Low-High Outlier'
}

fig, ax = plt.subplots(1, 1, figsize=(12, 10))

cmap = 'coolwarm'
merged_tracts.plot(column='significant', cmap=cmap, legend=True, alpha=0.6, ax=ax)

ctx.add_basemap(ax, crs=merged_tracts.crs.to_string(), source=ctx.providers.CartoDB.PositronNoLabels)

plt.title("Significant Local Moran's I Clusters", fontsize=16)
plt.xlabel("Longitude", fontsize=14)
plt.ylabel("Latitude", fontsize=14)

handles = [plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='red', markersize=10, label='High-High Cluster'),
           plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='blue', markersize=10, label='Low-Low Cluster'),
           plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='lightblue', markersize=10, label='High-Low Outlier'),
           plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='lightcoral', markersize=10, label='Low-High Outlier')]
plt.legend(handles=handles, title='Cluster Types')

for x, y, label in zip(merged_tracts.geometry.centroid.x, merged_tracts.geometry.centroid.y, merged_tracts['significant']):
    plt.text(x, y, str(label), fontsize=8, ha='center')

sm = plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(vmin=merged_tracts['significant'].min(), vmax=merged_tracts['significant'].max()))
sm._A = []
cbar = plt.colorbar(sm, ax=ax)
cbar.set_label('Significance Level')

plt.show()
```



Lisa Map that uses the information from the morans. Is more detailed than other other Lisa maps. The map shows a lot of significant polygons. Clusters are represented in the bottoum right and the middle left.

Conclusion The eye test of looking at the plots shows that there is a lot of clustering in the city of Las Vegas overall with no particular patterns. The whole city appears to be overrun by robberies. Overall the map is coated in robberies with some spots that appear to have a bit less. These overall maps often just show maps of population density. Naturally the places that more people live in will have more crime than communities of low density. Low income areas tend to have higher densities. The maps have a lot of areas with robberies and areas with less. The morans analysis is a great way to get information on our data. The global morans gives information about the whole dataset. For the morans the expected I is close to 0 showing it will not have spatial clustering or dispersion. Then it is shown that the nan P value being close to 0 means that most likely there is spatial autocorrelation. Local morans evaluate points in reference to their nearest neighbors. The local moran's I is non indicating no variability in the data set. In addition the p value is a non indicating no variability. This simulated p value is off of permutation simulations that can give a solid significance test. Like earlier the simulated P value indicates that this is not randomly distributed. It indicates distinct clusters within the points. For the local Moran's nan values are present. The P values being so low means the statistics are probably accurate. This is evidence that there is likely spatial autocorrelation. This morans is not as accurate as the one that gets run after a spatial join is completed. The spatial join is what solves the earlier problem of the data just reflecting the population of the city in a different way. For the post spatial join morans the expected I makes represents there is no spatial autocorrelation in the data. If the plots were placed randomly it would be close to this value according to this statistic. For the post spatial join local Moran's significant clustering is indicated but there are nan values. All of this considered it brings the conclusion that overall robberies are evenly distributed. Evenly distributed robberies reinforces that there is no spatial autocorrelation. The robberies are close to how they would be if randomly placed on the map. However in local tracts (areas) the robberies are clustered. Specific areas inside of a tract have robberies that are clustered. This is due to local factors in the area. A shopping center or some sort of neighborhood center. There are local reasons for Las Vegas having these specific trends. A big reason for the overall robberies not being clustered could be due to Las Vegas not having a good transit system with trains. It has one small train and many buses. In many cities crime is concentrated around transit centers that have trains. Las Vegas is a city that is different from a lot of places in the United States. The casinos and the fact that relative to other cities in the United States it is new gives it a different feel. It developed with more sprawl and roads than older cities on the East Coast. I learned a lot from this processes about spatial analysis. I got much better with the moran topics, centrography for point patterns, and spatial join tactics. Running the many different types of techniques and seeing what was worth keeping in the notebook was a good way to see the types of techniques that work. Data interpretation in the notebook felt more meaningful than the normal studios.

Bibliography in MLA

"LVMPD Reported Nibrs Robbery." LVMPD Open Data Portal, 25 Oct. 2021, opendata-lvmpd.hub.arcgis.com/datasets/9288d25383234c41a162444fc9c3e5be_0/explore.

"Las Vegas City, Nevada – Census Buraeu Tables". https://data.census.gov/table?q=Las%20Vegas%20city,%20Nev