Geospatial Clustering

Based on two clustering methods 1) Local Morans 2) Markov Clustering Linkage

And runs feature importance for both methods.

Before running this notebook, make sure the following steps were done:

1. myVenv environment was activated
2. All required dependencies were installed with "pip install -r requirements.txt"
3. jupyter was opened in the myVenv environment

To run every cell, press shift+enter

In [1]:
```
import sys, os
import pandas as pd
import numpy as np
import geopandas as gpd

import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go

from sklearn.decomposition import PCA
from sklearn.cluster import AffinityPropagation
from sklearn.manifold import TSNE
from sklearn.metrics import rand_score

from sklearn.metrics.pairwise import cosine_similarity
import json
from sklearn.cluster import SpectralClustering
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import normalize
from sklearn.decomposition import PCA

from pysal.lib import weights
from pysal.lib import weights
from splot.libpysal import plot_spatial_weights
from esda.moran import Moran, Moran_Local
from splot.esda import moran_scatterplot, plot_local_autocorrelation, lisa_cluster
import folium
import matplotlib.colors as colors
from geopy.geocoders import Nominatim
from shapely.geometry import Point
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
import networkx as nx
import markov_clustering as mc
from scipy.sparse import csr_matrix
import kaleido
```

If the previous cell gives a "ModuleNotFoundError", run the following command:

pip install insert_module_name

In the case of geopandas specifically, run the following command on command line:

conda install geopandas

In [2]:
```
## Module Installation if needed:

# pip install insert_module_name
```

Step 1: Input the state of choice below

In [3]:
```
stateName = "MA"
```

Step 2: Essential Functions

When you run these, there is no output until you actually call the function later below. These functions merge together the EJI and zipcode mapping functions for the specific state.

In [4]:
```
def mergeDFs(eji, zipT):
    """
    Merged two dataframes of EJI And zipToCensusTract
    @param eji: environmental justice index dataframe
    - if specific state, should already be a subdataframe
    @param zipT: zip to census tract conversion (subdataframe if state)
    @return merged dataframe
    """
    cols = eji.columns[10:]
    ejiDict = eji.set_index('GEOID', drop=True)[cols].to_dict(orient="index")

    ## Sets up the ZIP to TRACT conversion
    zipDF = zipT[['TRACT','ZIP','USPS_ZIP_PREF_STATE']]
    zipDict = {}
    zipCity = {}
    for i in range(len(zipDF.index)):
        temp = zipDF.iloc[i]
        tempZip = temp['ZIP']
        zipCity[tempZip] = temp['USPS_ZIP_PREF_STATE']
        if tempZip in zipDict:
            zipDict[tempZip].append(temp['TRACT'])
        else:
            zipDict[tempZip] = [temp['TRACT']]

    ## Averages the EJI data across a zipcode
    total = []
    for z in zipDict:
        arrays = []
        for key in zipDict[z]:
            if key in ejiDict:
                arrays.append(np.array(list(ejiDict[key].values())))
        meanArray = [np.mean(k) for k in zip(*arrays)]
        total.append(meanArray)

    merged = pd.DataFrame(total)
    merged.columns = cols
    merged['ZIP'] = list(zipDict.keys())

    ## Adds state into the data too
    states = []
    for m in merged['ZIP']:
        states.append(zipCity[m])

    merged['STATE'] = states
```

```
    return merged.dropna()

def plotlyZip(state):
    """
    Gets the plotly data for zipcodes based on a specific state
    @param state: capitalized state string
    @return zipcodes: JSON of zipcode data
    """

    folder = "Data/State-zip-code-GeoJSON/"
    stateFile = ""
    for file in os.listdir(folder):
        if file.startswith(state.lower()):
            stateFile = file
    with open(folder+stateFile) as f:
        zipcodes = json.load(f)
    return zipcodes
```

These are the defined columns for the EJI that we use for analysis

In [5]:
```
health = ["EP_BPHIGH","EP_ASTHMA","EP_CANCER","EP_MHLTH","EP_DIABETES","EPL_BPHIGH",
          "EPL_ASTHMA","EPL_CANCER","EPL_DIABETES","EPL_MHLTH","F_BPHIGH","F_ASTHMA",
          "F_CANCER","F_MHLTH","F_DIABETES"]
env = ["EPL_OZONE","EPL_PM","EPL_DSLPM","EPL_TOTCR","SPL_EBM_THEME1","RPL_EBM_DOM1",
    "EPL_NPL","EPL_TRI","EPL_TSD","EPL_RMP","EPL_COAL","EPL_LEAD","SPL_EBM_THEME2",
    "RPL_EBM_DOM2","EPL_PARK","EPL_HOUAGE","EPL_WLKIND","SPL_EBM_THEME3","RPL_EBM_DOM3",
    "EPL_RAIL","EPL_ROAD","EPL_AIRPRT","SPL_EBM_THEME4","RPL_EBM_DOM4","EPL_IMPWTR",
    "SPL_EBM_THEME5","RPL_EBM_DOM5","E_OZONE","E_PM","E_DSLPM","E_TOTCR","E_NPL","E_TRI"
    "E_TSD","E_RMP","E_COAL","E_LEAD","E_PARK","E_HOUAGE","E_WLKIND","E_RAIL","E_ROAD",
    "E_AIRPRT","E_IMPWTR"]
soc = ["EPL_MINRTY","SPL_SVM_DOM1","RPL_SVM_DOM1","EPL_POV200","EPL_NOHSDP","EPL_UNEMP",
    "EPL_RENTER","EPL_HOUBDN","EPL_UNINSUR","EPL_NOINT","SPL_SVM_DOM2","RPL_SVM_DOM2",
    "EPL_AGE65","EPL_AGE17","EPL_DISABL","EPL_LIMENG","SPL_SVM_DOM3","RPL_SVM_DOM3",
    "EPL_MOBILE","EPL_GROUPQ","SPL_SVM_DOM4","RPL_SVM_DOM4","EP_MINRTY","EP_POV200",
    "EP_NOHSDP","EP_UNEMP","EP_RENTER","EP_HOUBDN","EP_UNINSUR","EP_NOINT","EP_AGE65",
    "EP_AGE17","EP_DISABL","EP_LIMENG","EP_MOBILE","EP_GROUPQ"]
```

In [6]:
```
# Inputs are: EJI File, zipT file, and state name abbreviation
eji = pd.read_csv("Data/ejiData.csv")
zipT = pd.read_csv("Data/TRACT_ZIP_122023.csv")
state = stateName.upper()
```

In [7]:
```
zipcodes = plotlyZip(state)
zipSub = zipT[zipT['USPS_ZIP_PREF_STATE']==state]
ejiSub = eji[eji['StateAbbr']==state]
```

In [8]:
```
# this is the final data set
zipEJI = mergeDFs(ejiSub, zipSub)
```

Step 3: Spatial Correlation with Local Moran's (Method 1)

In [9]:
```
def spatialCorr(zipEJI, cols, zipcodes, state):
    """
    Runs Local Moran's spatial geoclustering
    @param zipEJI: dataframe of zipcode-EJI merged
    @param cols: columns of choice (default is ALL)
    @param zipcodes: zipcode JSON dict for choropleth
    @param state: state name
    @return a couple figures (choropleth, weights, local morans)
    """
    filename = "Data/zipcodeXY.csv"
```

```
zipFile = pd.read_csv(filename)

df = pd.merge(zipEJI, zipFile, on='ZIP')

# Drop rows with missing lat/lon
df = df.dropna(subset=['LAT', 'LNG'])

# Create a GeoDataFrame
geometry = [Point(xy) for xy in zip(df['LNG'], df['LAT'])]
gdf = gpd.GeoDataFrame(df, geometry=geometry)

# Optionally set the coordinate reference system (CRS) to WGS84 (EPSG:4326)
gdf.set_crs(epsg=32130, inplace=True)

w = weights.contiguity.Queen.from_dataframe(gdf, use_index=False)

plot_spatial_weights(w, gdf)
plt.savefig("Figures/%s/%s_spatialWeights.png" %(state, state))

## LOCAL SPATIAL AUTOCORRELATION

# Transforming weights into binary (if it's 1 = is neighbor, 0 = not neighbor)
w.transform = "B"

x = PCA(n_components=1).fit_transform(df[cols+["LAT","LNG"]].to_numpy())[:,0]
df['PCA'] = x

# Local Moran's I
pop_count_local_moran = Moran_Local(x, w)
moran = Moran(x, w)
print("Local Morans Results: ")
print("Moran Statistic:", moran.I)
print("Ranges from -1 to 1, closer to 1 indicates spatial clustered variable")
print("Morans P-value:",moran.p_sim)
print("If the p-value is below 0.05, we can reject null hypothesis and interpret thi
print()

# Plotting Local Moran's I scatterplot of pop_count
fig, ax = moran_scatterplot(pop_count_local_moran, p=0.05)

plt.text(1.95, 1, 'HH', fontsize=25)
plt.text(1.95, -1.0, 'HL', fontsize=25)
plt.text(-1.5, 1, 'LH', fontsize=25)
plt.text(-1.5, -1, 'LL', fontsize=25)
plt.savefig("Figures/%s/%s_localmorans.png" %(state, state))

# creating column with local_moran classification
df['pop_count_local_moran'] = pop_count_local_moran.q

# Dict to map local moran's classification codes
local_moran_classification = {1: 'HH', 2: 'LH', 3: 'LL', 4: 'HL'}

# Mapping local moran's classification codes
df['pop_count_local_moran'] = df['pop_count_local_moran'].map(local_moran_classifica

# p-value for each observation/neighbor pair
df['pop_count_local_moran_p_sim'] = pop_count_local_moran.p_sim

# If p-value > 0.05 it is not statistical significant
df['pop_count_local_moran'] = np.where(df['pop_count_local_moran_p_sim'] > 0.05, 'ns

# creates the choropleth map
newZips = []
zips = df['ZIP'].values.tolist()
for i in range(len(zips)):
    if len(str(zips[i]))==4:
```

```
            newZips.append("0"+str(zips[i]))
        else:
            newZips.append(str(zips[i]))
    df['Zipcode'] = newZips
    fig = px.choropleth(df,
                    geojson=zipcodes,
                    locations='Zipcode',
                    color='pop_count_local_moran',
                    color_continuous_scale="Viridis",
                    range_color=(min(df['pop_count_local_moran']),max(df['pop_count_loca
                    featureidkey="properties.ZCTA5CE10",
                    scope="usa",
                    labels={'Cluster':'Cluster_Category'}
                          )
    fig.update_geos(fitbounds="locations", visible=False)
    fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})

    fig.write_image("Figures/%s/%s_moranMap.png" %(state, state), scale=2)
    fig.show()

    # run feature importance also
    featureImportance(df[cols],df['pop_count_local_moran'].values.tolist(), 'Morans', st

    return w, gdf, df

def featureImportance(X,y, method, state):
    """
    Analyzes feature importance using a random forest model
    @param X: the input data as a dataframe
    @param y: the clusters
    @param method: a string of the method name
    @param state: state name
    @return figure "<method>_featureImp.png" in Figures folder
    """
    clf = RandomForestClassifier(n_estimators=100, random_state=42)  # Use 100 trees
    clf.fit(X, y)

    importances = clf.feature_importances_

    feature_importance_df = pd.DataFrame({
        'Feature': X.columns,
        'Importance': importances
    }).sort_values(by='Importance', ascending=False).head(10)

    fig = px.bar(feature_importance_df, x='Importance', y='Feature', orientation='h',
             title='Top 10 Feature Importances')
    fig.write_image("Figures/%s/%s_%s_featureImp.png" %(state, state, method))
    fig.show()
    return
```

Step 5: Running spatial correlations and interpreting results

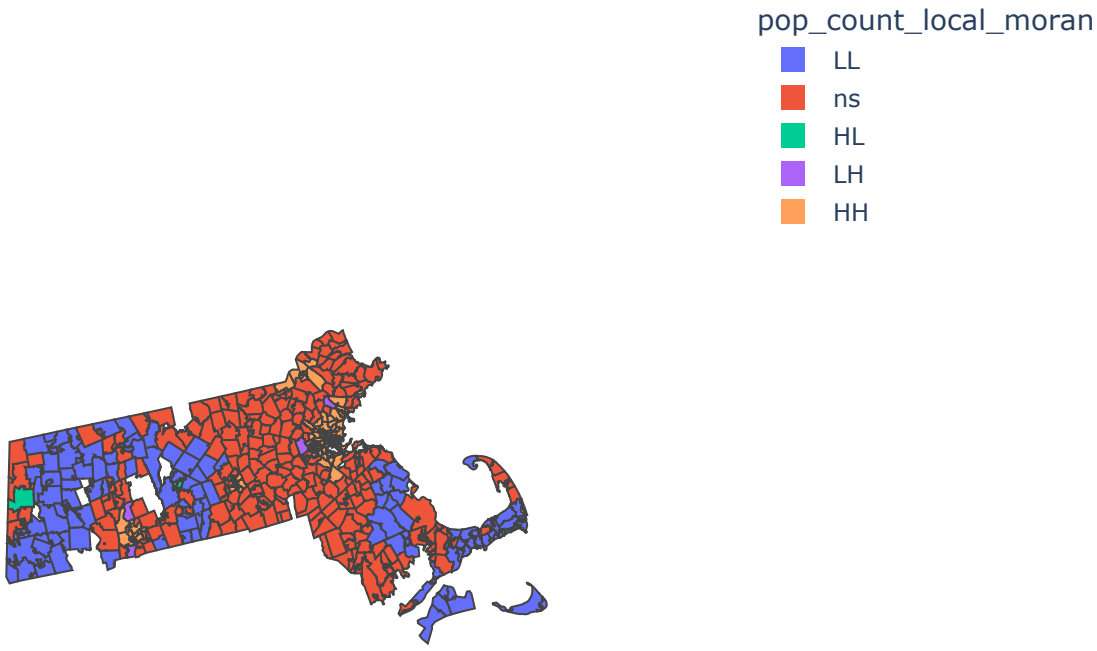
The following results show up with this code:

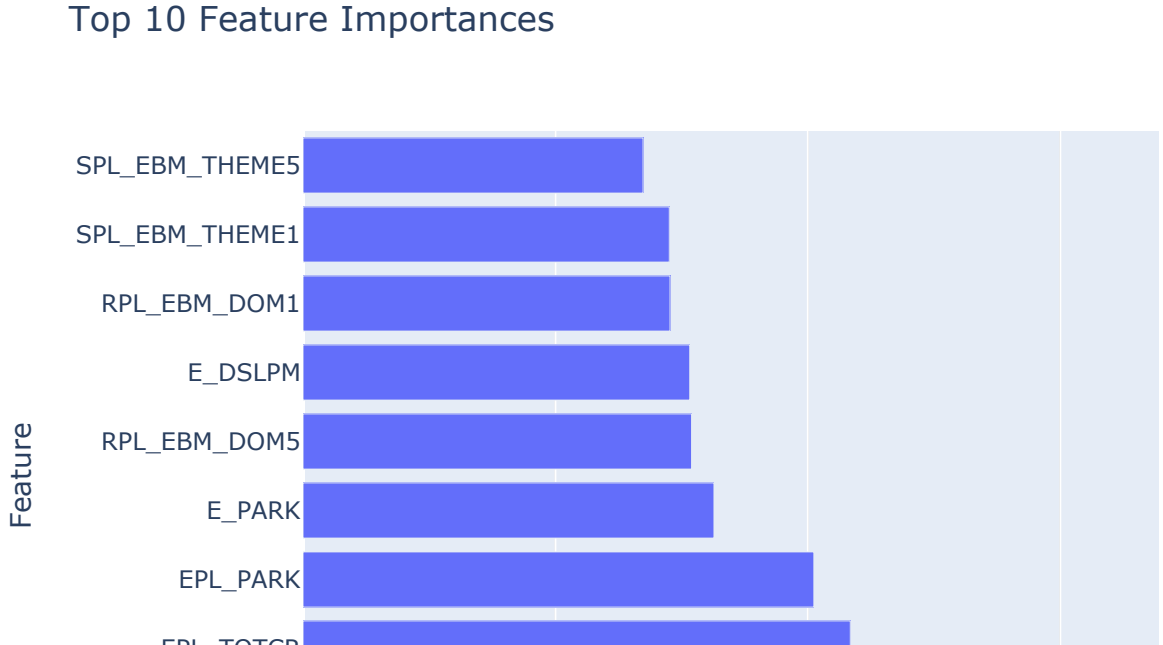
1. Local Moran's Results
2. A map of local moran's clusters
3. Top 10 feature importance of moran's clusters
4. A network map of weights
5. Local Moran's scatter plot

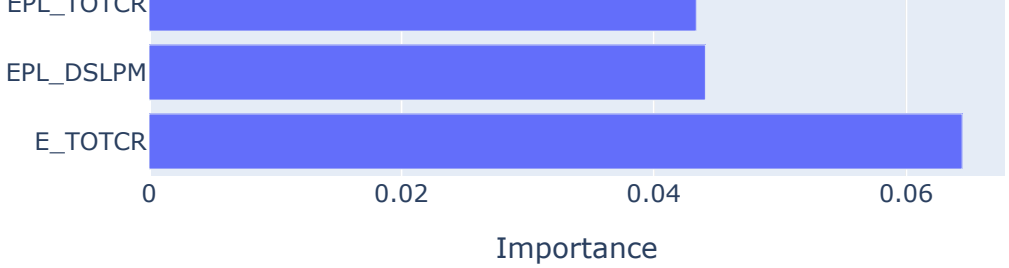
In [10]:
```
if not os.path.isdir("Figures/%s" %(state)):
```

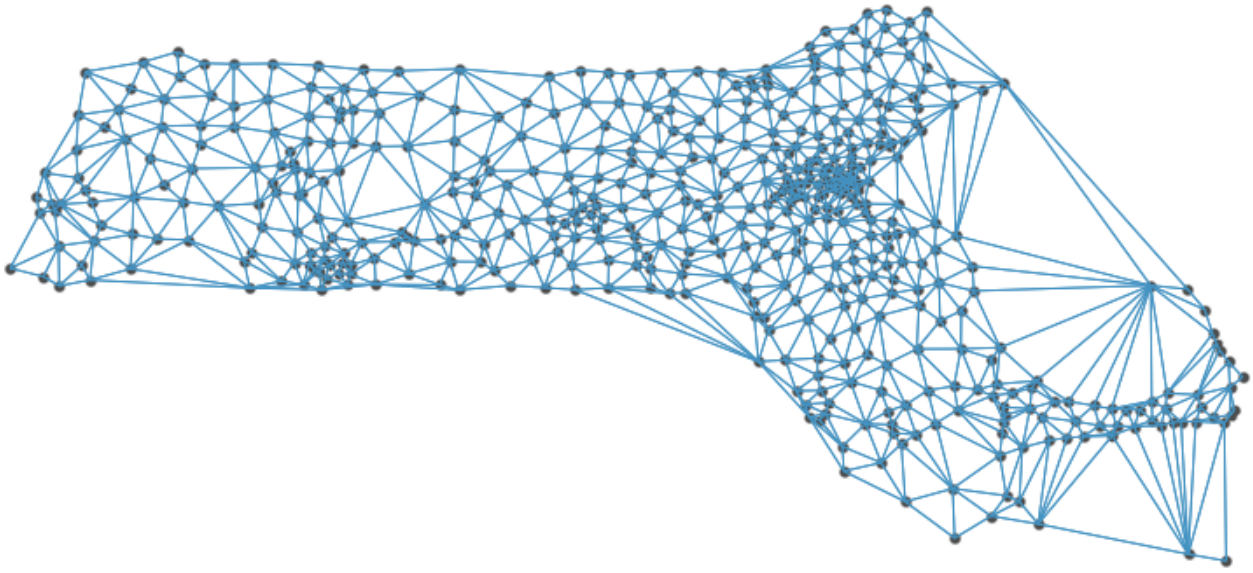
```
    os.mkdir("Figures/%s" %(state))
w, gdf, df = spatialCorr(zipEJI, health+env+soc, zipcodes, state)
```

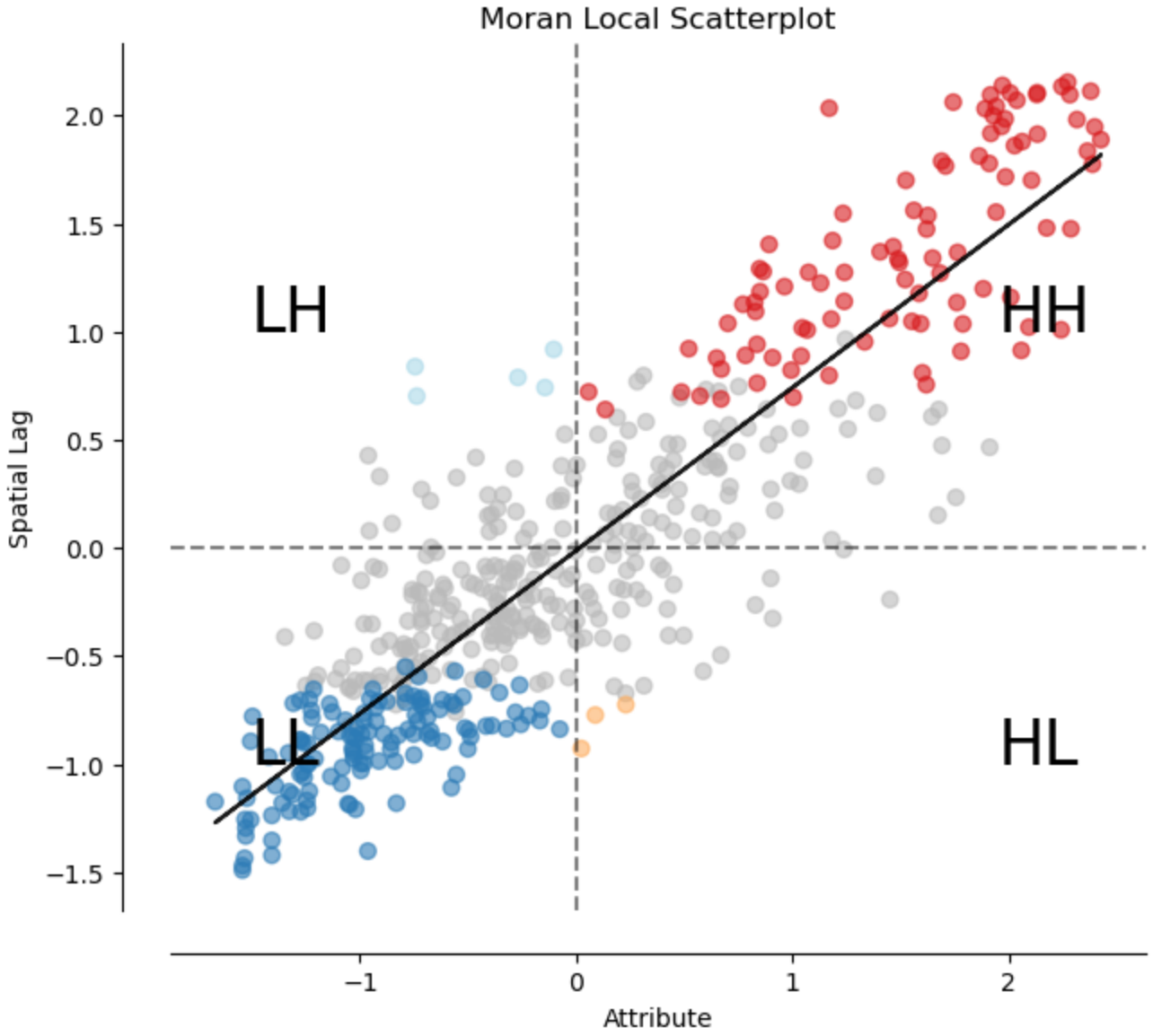
```
Local Morans Results:
Moran Statistic: 0.753910802325287
Ranges from -1 to 1, closer to 1 indicates spatial clustered variable
Morans P-value: 0.001
If the p-value is below 0.05, we can reject null hypothesis and interpret this variable
is clustered spatially.
```






EPL_TOTCR
EPL_DSLPM
E_TOTCR
0
0.02
0.04
0.06
Importance





Step 6: Network Clustering

In [11]:

```
def network(w, gdf, cols, df, zipcodes, state):
    """
    Network based clustering using markovian clustering
    @param w: weights object from Queen
    @param gdf: pandas geodataframe
    @param cols: columns of choice (default is ALL)
    @param df: pandas dataframe of gdf
    @param zipcodes: zipcodes dictionary for choropleth
    @param state: state input
    """
    # Step 1: creates new networkx graph
    G = nx.Graph()

    for region, neighbors in w.neighbors.items():
        for neighbor in neighbors:
            G.add_edge(region, neighbor)

    for idx, row in gdf.iterrows():
        G.nodes[idx]['geometry'] = row['geometry']
        for c in cols:
            G.nodes[idx][c] = row[c]

    adj_matrix = nx.to_scipy_sparse_array(G)
    sparse_csr_matrix = csr_matrix(adj_matrix)
```

```python
# Step 2: Perform the Markov Clustering (MCL)
result = mc.run_mcl(sparse_csr_matrix)  # Run MCL algorithm
clusters = mc.get_clusters(result)  # Get clusters

# Step 3: Assign each node to its respective cluster
# Create a dictionary where the key is the node and value is the cluster number
cluster_dict = {}
for i, cluster in enumerate(clusters):
    for node in cluster:
        cluster_dict[node] = i

# Step 4: Define a color for each cluster
# Generate random colors for each cluster
num_clusters = len(clusters)
colors = ['#%06X' % np.random.randint(0, 0xFFFFFF) for _ in range(num_clusters)]

# Step 5: Extract node geometries (we will use centroids)
gdf['centroid'] = gdf.geometry.centroid
node_positions = {i: (geom.x, geom.y) for i, geom in gdf['centroid'].items()}

# Step 6: Prepare node traces for plotting
node_x = []
node_y = []
node_color = []
for node, (x, y) in node_positions.items():
    node_x.append(x)
    node_y.append(y)
    node_color.append(colors[cluster_dict[node]])

# Step 7: Prepare edge traces for plotting
edge_x = []
edge_y = []
for edge in G.edges():
    x0, y0 = node_positions[edge[0]]
    x1, y1 = node_positions[edge[1]]
    edge_x.append(x0)
    edge_x.append(x1)
    edge_x.append(None)  # To create breaks between segments
    edge_y.append(y0)
    edge_y.append(y1)
    edge_y.append(None)  # To create breaks between segments

# Step 8: Plot the edges
edge_trace = go.Scatter(
    x=edge_x, y=edge_y,
    line=dict(width=1, color='black'),
    hoverinfo='none',
    mode='lines'
)

# Step 9: Plot the nodes
node_trace = go.Scatter(
    x=node_x, y=node_y,
    mode='markers',
    hoverinfo='text',
    marker=dict(
        showscale=False,
        color=node_color,
        size=10,
        line_width=2
    ),
    text=[f"Node {i} - Cluster {cluster_dict[i]}" for i in G.nodes()]  # Optional: A
)

# Step 10: Combine both node and edge traces into a Plotly figure
```

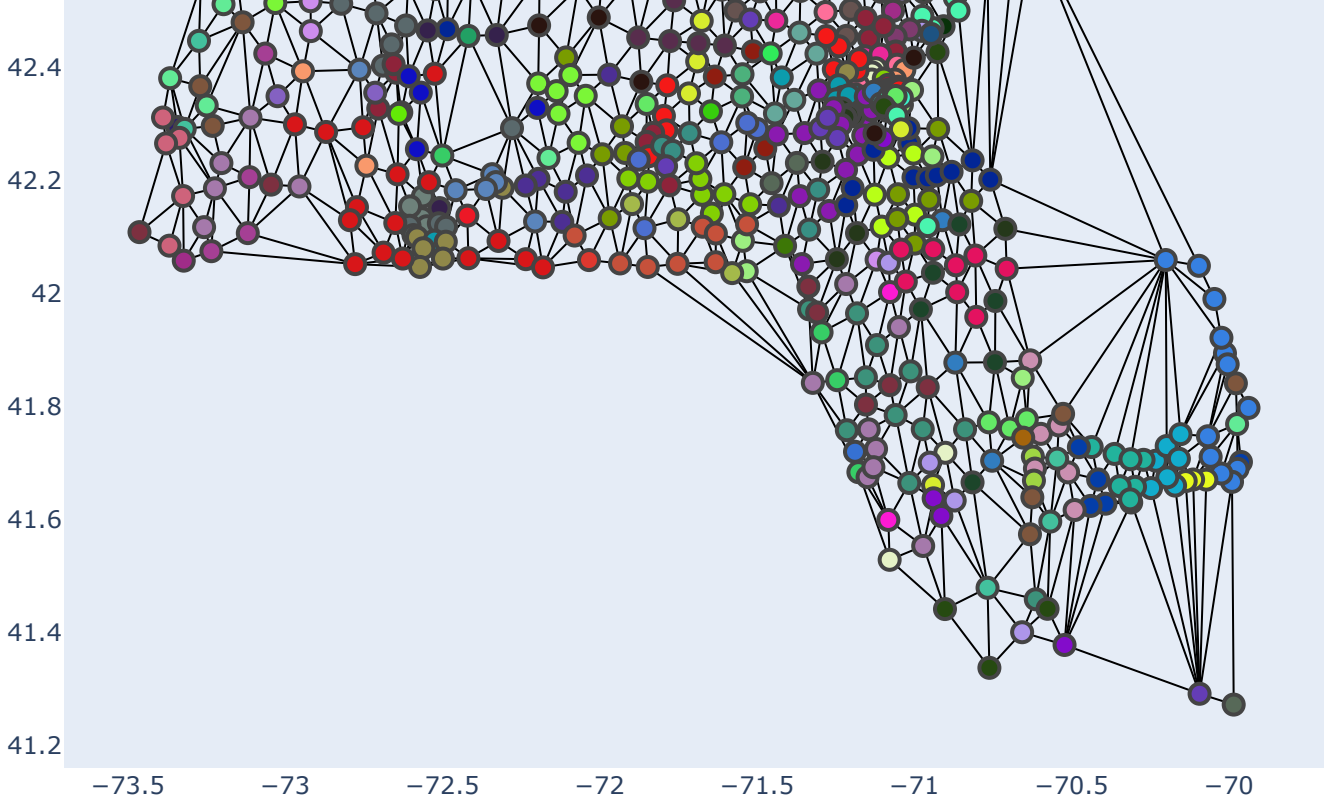
```
    fig = go.Figure(data=[edge_trace, node_trace])

    # Step 11: Update layout for better display
    fig.update_layout(
        title="NetworkX Graph with Node Geometries",
        titlefont_size=16,
        showlegend=False,
        hovermode='closest',
        margin=dict(b=0, l=0, r=0, t=0),
        xaxis=dict(showgrid=False, zeroline=False),
        yaxis=dict(showgrid=False, zeroline=False)
    )

    # Show the plot
    fig.write_image("Figures/%s/%s_network.png" %(state, state), scale=2)
    fig.show()

    # Step 12: Create the choropleth map
    clusters = []
    for idx, row in gdf.iterrows():
        clusters.append(cluster_dict[idx])
    df['MCL'] = clusters
    print("Markov Clustering: ")
    print("Number of Clusters: ", max(clusters))
    print()
    fig2 = px.choropleth(df,
                    geojson=zipcodes,
                    locations='Zipcode',
                    color='MCL',
                    color_continuous_scale="Viridis",
                    range_color=(min(df['MCL']),max(df['MCL'])),
                    featureidkey="properties.ZCTA5CE10",
                    scope="usa",
                    labels={'Cluster':'Cluster_Category'}
                          )
    fig2.update_geos(fitbounds="locations", visible=False)
    fig2.update_layout(margin={"r":0,"t":0,"l":0,"b":0})

    fig2.write_image("Figures/%s/%s_mclMap.png" %(state, state))
    fig2.show()

    # Step 13: Runs the feature importance also
    featureImportance(df[cols],clusters, 'MCL', state)
    return
```

Step 7: Running Network Clustering

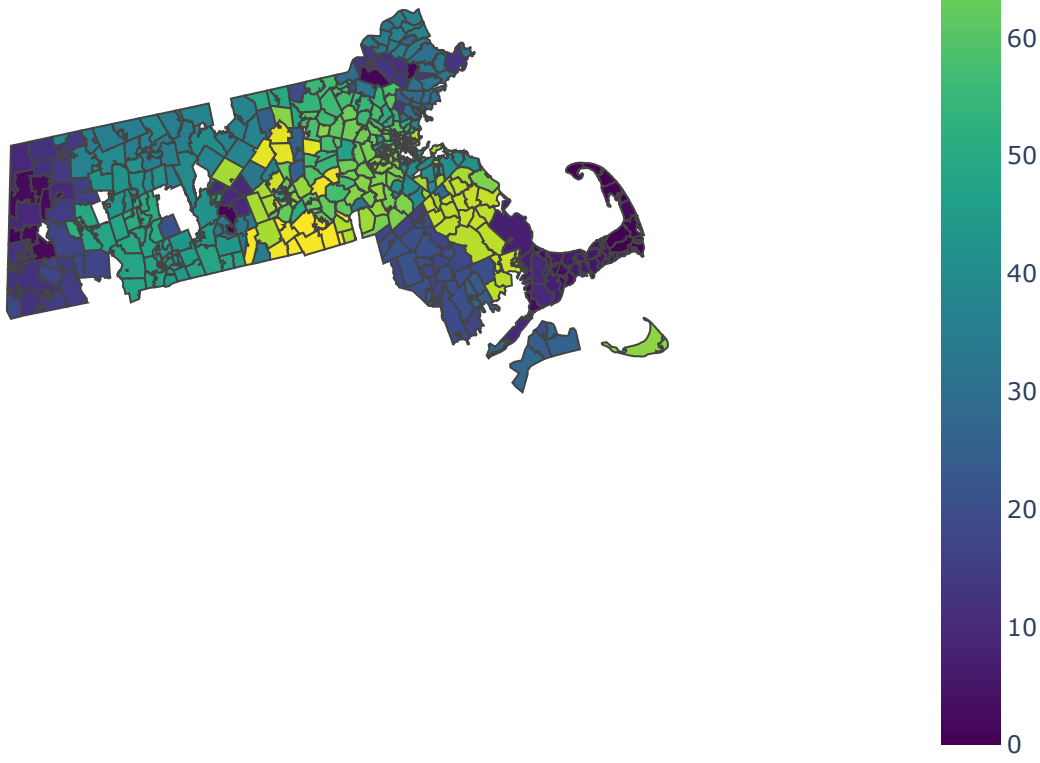
The following results show up here:

1. Network representation of the state
2. MCL Clusters
3. Top 10 features for MCL clusters

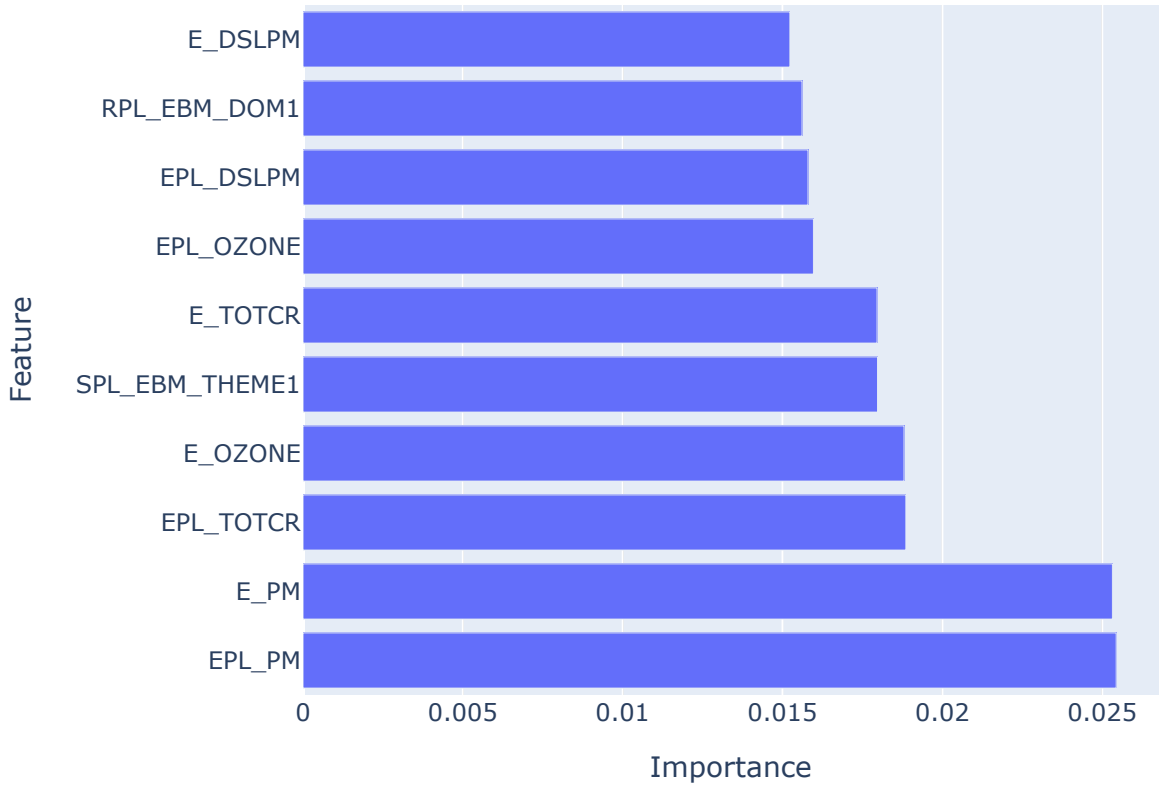
In [12]:
```
network(w, gdf, health+env+soc, df, zipcodes, state)
```



Markov Clustering:
Number of Clusters: 82



Top 10 Feature Importances



Step 8: Interpretation

At this point, you can add any code of interest to understand more about these results if you'd like.

In [ ]: