

## Guia Folium

**Curso:** UFCD 10809

**UFCD/Módulo/Temática:** Visualização de dados em Python

**Ação:** 10809\_1L Visualização de dados em Python

**Formador/a:** Sandra Liliana Meira de Oliveira

# Visualização Geoespacial com Folium

## Introdução

Este tutorial apresenta as funcionalidades essenciais da biblioteca Folium para criação de mapas interativos com Python. Inclui marcadores, agrupamento com clusters, utilização de GeoJSON e mapas choropléticos.

## Funcionalidades Essenciais da Biblioteca Folium

Funcionalidade	Descrição	Exemplo de uso
<code>folium.Map()</code>	Cria o mapa base centrado em coordenadas específicas	<code>folium.Map(location=[45.4215, -75.6972], zoom_start=5)</code>
<code>folium.Marker()</code>	Adiciona marcadores ao mapa	<code>folium.Marker([45.4215, -75.6972], popup="Ottawa").add_to(m)</code>
<code>folium.Popup()</code>	Permite adicionar uma janela de informação ao clicar num marcador	<code>popup=folium.Popup("Informação detalhada", max_width=300)</code>
<code>folium.FeatureGroup()</code>	Agrupa elementos (marcadores, polígonos, etc.) para controlo separado	<code>fg = folium.FeatureGroup(name="Cidades")</code>
<code>folium.plugins.MarkerCluster()</code>	Agrupa marcadores automaticamente para melhor legibilidade	<code>MarkerCluster().add_to(m)</code>
<code>folium.Choropleth()</code>	Cria mapas temáticos (choropléticos) com base em ficheiros GeoJSON + dados	<code>folium.Choropleth(geo_data=geojson_path, data=df, columns=[...])</code>
<code>folium.LayerControl()</code>	Adiciona um controlo de camadas interativo ao mapa	<code>folium.LayerControl().add_to(m)</code>

## Instalação

Para instalar a biblioteca Folium, utiliza o seguinte comando no terminal:

```
pip install folium
```

## Funcionalidades Principais

1. **folium.Map()** - Cria o mapa base. Parâmetros: location, zoom\_start, tiles, control\_scale.

```
folium.Map(location=[latitude, longitude], zoom_start=10, tiles='OpenStreetMap',  
control_scale=True)
```

- **Parâmetros:**
- location (*obrigatório*): lista com as coordenadas [latitude, longitude] do centro do mapa.
- zoom\_start: nível de zoom inicial (inteiro). Valores típicos:  
1 (muito afastado, mundo) até 18 (muito aproximado).
- tiles: estilo do mapa base. Pode ser:
  - 'OpenStreetMap' (padrão)
  - 'Stamen Terrain'
  - 'Stamen Toner'
  - 'Stamen Watercolor'
  - 'CartoDB positron'
  - 'CartoDB dark\_matter'
  - Ou uma **URL personalizada** de tiles
- control\_scale: mostra a escala no canto do mapa (True ou False).

2. **folium.Marker()** - Adiciona um marcador. Parâmetros: location, popup, tooltip, icon.

```
folium.Marker(  
    location=[latitude, longitude],  
    popup='Texto informativo',  
    tooltip='Texto ao passar o rato',  
    icon=folium.Icon(color='blue', icon='info-sign')  
).add_to(mapa)
```

### Parâmetros:

- location: coordenadas [lat, lon]
- popup: texto ou objeto folium.Popup com informação detalhada
- tooltip: texto que aparece ao passar o cursor
- icon: define a aparência do ícone do marcador (usa **folium.Icon()**)

3. **folium.Icon()** - Personaliza ícones de marcadores. Parâmetros: color, icon, prefix.

```
folium.Icon(color='green', icon='home', prefix='fa')
```

**Parâmetros:**

- color: cor do marcador. Opções: 'red', 'blue', 'green', 'purple', 'orange', 'darkred', 'lightred', 'beige', 'darkblue', 'darkgreen', 'cadetblue', 'darkpurple', 'white', 'pink', 'lightblue', 'lightgreen', 'gray', 'black', 'lightgray'
- icon: ícone a usar (ex: 'info-sign', 'cloud', 'home'). Usa ícones do **Bootstrap Glyphicons** ou **FontAwesome**.
- prefix: 'glyphicon' (padrão) ou 'fa' (para FontAwesome)

4. **folium.Popup()** - Janela de informação clicável.

```
folium.Popup("Texto ou HTML", max_width=300)
```

**Parâmetros:**

- html ou str: conteúdo do popup
- max\_width: largura máxima em pixels (opcional)

5. **folium.Tooltip()** - Texto ao passar o cursor.

```
folium.Tooltip("Texto breve")
```

6. **folium.Circle() / CircleMarker()** - Desenho de círculos geográficos.

```
folium.Circle(  
    location=[lat, lon],  
    radius=1000,  
    color='blue',  
    fill=True,  
    fill_color='blue'  
) .add_to(mapa)
```

**Diferença:**

- Circle: raio em **metros** (escalável com zoom)

- CircleMarker: raio em **píxeis** (fixo)

## 7. **folium.FeatureGroup()** - Agrupamento lógico de camadas.

Agrupa vários objetos (marcadores, círculos, etc.) para ativação/desativação em conjunto.

```
fg = folium.FeatureGroup(name="Grupo A")
fg.add_child(folium.Marker([...]))
mapa.add_child(fg)
```

## 8. **folium.plugins.MarkerCluster()** - Agrupamento automático de marcadores.

Agrupa automaticamente marcadores próximos, formando “clusters”.

```
from folium.plugins import MarkerCluster
cluster = MarkerCluster().add_to(mapa)
folium.Marker([lat, lon], popup="Cidade").add_to(cluster)
```

## 9. **folium.GeoJson()** - Exibição de ficheiros GeoJSON.

Carrega e exibe um ficheiro GeoJSON (limites geográficos, polígonos, etc.)

```
folium.GeoJson('ficheiro.geojson', name="Limites").add_to(mapa)
```

## 10. **folium.Choropleth()** - Criação de mapas temáticos com dados regionais.

Cria **mapas temáticos** (choropléticos) associando dados a regiões.

```
folium.Choropleth(
    geo_data='provincias_canada.geojson',
    name='Imigração',
    data=df,
    columns=['Provincia', 'Imigrantes'],
    key_on='feature.properties.NOME_PROVINCIA',
    fill_color='YlOrRd',
    fill_opacity=0.7,
    line_opacity=0.2,
    legend_name='Número de Imigrantes'
).add_to(mapa)
```

### **Parâmetros:**

- geo\_data: caminho para o ficheiro .geojson ou dicionário

- data: DataFrame com os dados
- columns: lista com [chave\_região, valor] a mapear
- key\_on: campo do GeoJSON que corresponde à chave no DataFrame (ex: 'feature.properties.nome')
- fill\_color: esquema de cores:
  - 'YlOrRd', 'BuPu', 'PuRd', 'Greens', 'Reds', 'Blues', 'Oranges', 'Viridis', 'Spectral', etc.
- fill\_opacity, line\_opacity: opacidade de preenchimento e contorno
- legend\_name: nome apresentado na legenda

## 11. folium.LayerControl() - Controlo de visibilidade de camadas.

Adiciona um botão para mostrar/ocultar camadas (FeatureGroups, Choropleths, etc.)

```
folium.LayerControl(collapsed=False).add_to(mapa)
```

**collapsed:** False mantém o menu aberto; True colapsa por defeito.

## 12. mapa.save('mapa.html') - Exportação do mapa para ficheiro HTML.

### Exemplo 1: Mapa com Marcadores e Clusters

Este exemplo cria um mapa com três cidades do Canadá, utilizando marcadores com tooltip e popup, agrupados com MarkerCluster.

1. Criar DataFrame com cidades
2. Criar mapa com folium.Map()
3. Adicionar marcadores com folium.Marker()
4. Agrupar com folium.plugins.MarkerCluster()
5. Guardar com mapa.save()

```
import folium
from folium.plugins import MarkerCluster
import pandas as pd

mapa = folium.Map(location=[56.1304, -106.3468], zoom_start=4, tiles='CartoDB positron')

# Dados fictícios
dados = pd.DataFrame({
    'Cidade': ['Toronto', 'Vancouver', 'Montreal'],
    'Latitude': [43.651070, 49.282729, 45.501689],
```

```

'Longitude': [-79.347015, -123.120738, -73.567256],
'População': [2731571, 631486, 1704694]
})

cluster = MarkerCluster().add_to(mapa)

for i, row in dados.iterrows():
    folium.Marker(
        location=[row['Latitude'], row['Longitude']],
        popup=f'{row['Cidade']}  
População: {row['População']}',
        tooltip=row['Cidade']
    ).add_to(cluster)

folium.LayerControl().add_to(mapa)
mapa.save('mapa_canada.html')

```



## Exemplo 2: Mapa Choropleth com GeoJSON

Este exemplo cria um mapa temático com dados de imigração, usando polígonos de províncias simuladas em GeoJSON.

1. Criar DataFrame com dados de imigração
2. Definir estrutura GeoJSON simplificada
3. Criar mapa base
4. Usar folium.Choropleth() para ligar os dados ao mapa
5. Adicionar legenda

```

import json

# Criar dados simulados de imigração por província
imigracao_df = pd.DataFrame({
    'Provincia': ['Ontario', 'British Columbia', 'Quebec'],
    'Imigrantes': [150000, 90000, 120000]
})

# Criar um GeoJSON simplificado (simulado) com três províncias
geojson_data = {
    "type": "FeatureCollection",
    "features": [
        {
            "type": "Feature",
            "properties": {"Provincia": "Ontario"},
            "geometry": {
                "type": "Polygon",
                "coordinates": [
                    [
                        [-95.0, 49.0],
                        [-79.0, 49.0],
                        [-79.0, 42.0],
                        [-95.0, 42.0],
                        [-95.0, 49.0]
                    ]
                ]
            }
        },
        {
            "type": "Feature",
            "properties": {"Provincia": "British Columbia"},
            "geometry": {
                "type": "Polygon",
                "coordinates": [
                    [
                        [-125.0, 55.0],
                        [-114.0, 55.0],
                        [-114.0, 48.0],
                        [-125.0, 48.0],
                        [-125.0, 55.0]
                    ]
                ]
            }
        },
        {
            "type": "Feature",
            "properties": {"Provincia": "Quebec"},
            "geometry": {
                "type": "Polygon",
                "coordinates": [
                    [
                        [-80.0, 55.0],
                        [-60.0, 55.0],
                        [-60.0, 45.0],
                        [-80.0, 45.0],
                        [-80.0, 55.0]
                    ]
                ]
            }
        }
    ]
}

```

```

        ]
    }

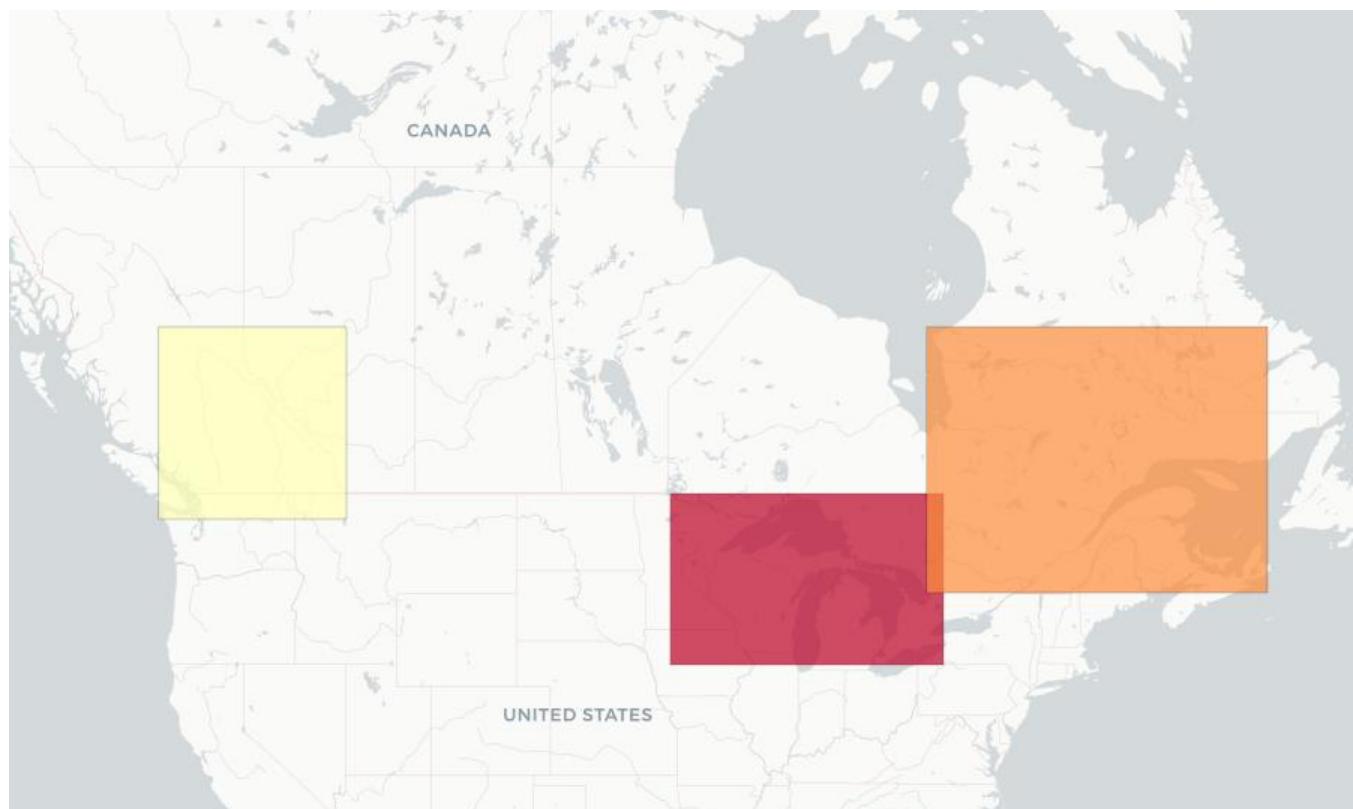
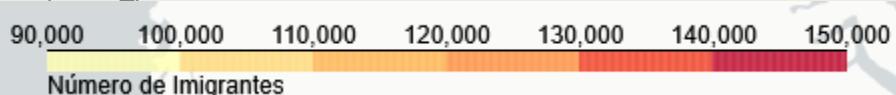
# Criar o mapa base
mapa_choropleth = folium.Map(location=[56.1304, -106.3468], zoom_start=4, tiles='CartoDB
positron')

# Adicionar o choropleth
folium.Choropleth(
    geo_data=geojson_data,
    name='Imigração por Província',
    data=imigracao_df,
    columns=['Província', 'Imigrantes'],
    key_on='feature.properties.Província',
    fill_color='YlOrRd',
    fill_opacity=0.7,
    line_opacity=0.2,
    legend_name='Número de Imigrantes'
).add_to(mapa_choropleth)

# Guardar como ficheiro HTML
choropleth_path = "/mnt/data/mapa_choropleth_canada.html"
mapa_choropleth.save(choropleth_path)

```

choropleth\_path



**A biblioteca Folium é poderosa e versátil para criar mapas interativos em Python. Pode ser integrada facilmente com Pandas, GeoJSON e outras bibliotecas para análise geoespacial.**

## GeoJson

O **GeoJSON** é um **formato aberto baseado em JSON** (JavaScript Object Notation) usado para **representar dados geoespaciais**. É muito utilizado em aplicações de **geoprocessamento, web mapping** e sistemas de informação geográfica (**SIG/GIS**) por ser leve, fácil de interpretar e compatível com várias bibliotecas e ferramentas modernas, como Leaflet, Mapbox, QGIS, e PostGIS.

**GeoJSON** descreve **objetos geográficos** (pontos, linhas, polígonos) e seus **atributos**. Baseia-se na especificação JSON e suporta os seguintes tipos de geometria:

- Point
- LineString
- Polygon
- MultiPoint
- MultiLineString
- MultiPolygon
- GeometryCollection
- Feature
- FeatureCollection

```
{  
  "type": "Feature",  
  "geometry": {  
    "type": "Point",  
    "coordinates": [-8.611, 41.149]  
  },  
  "properties": {  
    "nome": "Porto"  
  }  
}
```

### 1. Exemplo simples de um ficheiro GeoJSON (manual)

```
{  
  "type": "FeatureCollection",  
  "features": [  
    {  
      "type": "Feature",  
      "geometry": {  
        "type": "Point",  
        "coordinates": [-8.611, 41.149]  
      },  
      "properties": {  
        "cidade": "Porto"  
      }  
    }  
  ]  
}
```

## Passos:

- **type**: O tipo principal é FeatureCollection (coleção de elementos geográficos).
- **features**: Lista de objetos do tipo Feature.
- Cada Feature tem uma **geometry** (forma geográfica) e **properties** (atributos, como nome, população, etc).
- As **coordinates** seguem o padrão [longitude, latitude].

## 2. Ler e visualizar GeoJSON em Python

```
import json

# Abrir e ler um ficheiro GeoJSON
with open('porto.geojson', 'r') as f:
    data = json.load(f)

# Ver os dados
print(json.dumps(data, indent=2))

Aqui estamos a usar a biblioteca padrão json para carregar e imprimir os dados.
```

## 3. Criar e guardar GeoJSON com GeoPandas

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

# Criar uma GeoDataFrame com dados de cidades
dados = {
    'cidade': ['Porto', 'Lisboa'],
    'geometry': [Point(-8.611, 41.149), Point(-9.139, 38.722)]
}

gdf = gpd.GeoDataFrame(dados, crs="EPSG:4326")

# Guardar como ficheiro GeoJSON
gdf.to_file("cidades.geojson", driver="GeoJSON")

# Ver o GeoDataFrame
print(gdf)
```

## Explicação:

- Criamos pontos (Point) com coordenadas geográficas.
- crs="EPSG:4326" define o sistema de coordenadas (latitude/longitude padrão).
- to\_file(..., driver="GeoJSON") guarda como GeoJSON.

## 4. Visualizar GeoJSON num mapa interativo com folium

```
import folium
import geopandas as gpd

# Carregar o ficheiro GeoJSON criado antes
gdf = gpd.read_file("cidades.geojson")

# Criar mapa centrado em Portugal
mapa = folium.Map(location=[39.5, -8.0], zoom_start=6)

# Adicionar camada GeoJSON ao mapa
folium.GeoJson(gdf).add_to(mapa)
```

```
# Mostrar mapa (em Jupyter Notebook ou JupyterLab)
mapa
```

## Se quiseres guardar:

```
mapa.save("mapa_cidades.html")
```

## 5. Exemplos mais complexos

Vamos agora ver **exemplos práticos com linhas (LineString) e polígonos (Polygon)** em GeoJSON, com o passo a passo para:

1. Criar linhas (ex: uma estrada entre cidades)
2. Criar polígonos (ex: zona urbana ou área de um parque)
3. Guardar como GeoJSON
4. Visualizar com folium

- **Exemplo com LineString (linha entre Porto e Lisboa)**

```
import geopandas as gpd
from shapely.geometry import LineString

# Coordenadas de Porto e Lisboa
linha = LineString([
    (-8.611, 41.149), # Porto
    (-9.139, 38.722) # Lisboa
])

# Criar GeoDataFrame com a linha
gdf_linha = gpd.GeoDataFrame({'nome': ['Estrada Porto-Lisboa'], 'geometry': [linha]}, crs="EPSG:4326")

# Guardar como GeoJSON
gdf_linha.to_file("estrada.geojson", driver="GeoJSON")
```

- **Exemplo com Polygon (zona urbana imaginária)**

```
from shapely.geometry import Polygon

# Coordenadas dos vértices do polígono (formando um quadrado)
poligono = Polygon([
    (-9.14, 38.72),
    (-9.13, 38.72),
    (-9.13, 38.73),
    (-9.14, 38.73),
    (-9.14, 38.72) # Fechar o polígono (volta ao início)
])

# Criar GeoDataFrame com o polígono
gdf_poligono = gpd.GeoDataFrame({'zona': ['Zona A'], 'geometry': [poligono]}, crs="EPSG:4326")
```

```
# Guardar como GeoJSON
gdf_poligono.to_file("zona.geojson", driver="GeoJSON")
```

- **Visualizar tudo num mapa com folium**

```
import folium

# Mapa centrado em Portugal
mapa = folium.Map(location=[39.5, -8.0], zoom_start=6)

# Adicionar GeoJSON da estrada
folium.GeoJson("estrada.geojson", name="Estrada").add_to(mapa)

# Adicionar GeoJSON da zona
folium.GeoJson("zona.geojson", name="Zona").add_to(mapa)

# Adicionar camadas de controlo
folium.LayerControl().add_to(mapa)

# Mostrar (em Jupyter Notebook) ou guardar
mapa.save("mapa_completo.html")
```

O ficheiro mapa\_completo.html terá:

- A linha entre Porto e Lisboa (LineString)
- Um polígono desenhado sobre Lisboa (Polygon)
- Camadas interativas com nomes

Vamos agora incluir:

- **Multipolígonos (MultiPolygon)** (ex: várias ilhas ou zonas) - útil para representar regiões compostas por várias áreas (ex: arquipélagos, parques com zonas separadas)
- **Estilo personalizado** (cores, espessura, popup com info) - mudar cor, espessura, transparência
- **Popups com info dos atributos** nas features GeoJSON - mostrar atributos no mapa ao clicar

### Multipolígonos (MultiPolygon)

```
from shapely.geometry import MultiPolygon, Polygon
import geopandas as gpd

# Criar dois pequenos polígonos
p1 = Polygon([(-9.2, 38.7), (-9.1, 38.7), (-9.1, 38.75), (-9.2, 38.75), (-9.2, 38.7)])
p2 = Polygon([(-9.0, 38.7), (-8.9, 38.7), (-8.9, 38.75), (-9.0, 38.75), (-9.0, 38.7)])

# Criar multipolíгоно
multi = MultiPolygon([p1, p2])

# Criar GeoDataFrame
gdf_multi = gpd.GeoDataFrame({'zona': ['Zonas Separadas'], 'geometry': [multi]}, crs="EPSG:4326")

# Guardar como GeoJSON
gdf_multi.to_file("multipoligono.geojson", driver="GeoJSON")
```

## Estilo personalizado no folium

```
import folium

# Criar o mapa
mapa = folium.Map(location=[38.72, -9.1], zoom_start=10)

# Adicionar multipolígono com estilo
folium.GeoJson(
    "multipoligono.geojson",
    name="Zonas",
    style_function=lambda feature: {
        'fillColor': '#0080ff',
        'color': '#003366',
        'weight': 2,
        'fillOpacity': 0.5
    },
    tooltip=folium.GeoJsonTooltip(fields=["zona"], aliases=["Zona:])
).add_to(mapa)

# Adicionar controlo de camadas
folium.LayerControl().add_to(mapa)

# Guardar
mapa.save("mapa_multipolygon.html")
```

## Popups com info (usando atributos do GeoJSON)

```
folium.GeoJson(
    "cidades.geojson",
    name="Cidades",
    popup=folium.GeoJsonPopup(fields=["cidade"], aliases=["Cidade:")
).add_to(mapa)
```

## Script Python completo que:

Cria:

- Pontos (cidades com população)
- Linha (estrada entre Porto e Lisboa)
- Polígonos (zona urbana de Lisboa)
- Multipolígonos (duas zonas separadas)

Adiciona:

- Estilos personalizados para cada camada
- Popups com dados fictícios
- Tooltip para facilitar a visualização
- Controlo de camadas

Gera:

- Ficheiros GeoJSON
- Mapa HTML interativo com tudo

```

import geopandas as gpd
from shapely.geometry import Point, LineString, Polygon, MultiPolygon
import folium

# -----
# Criar dados GeoEspaciais
# -----

# 1. Cidades (Pontos)
cidades = gpd.GeoDataFrame({
    'cidade': ['Porto', 'Lisboa'],
    'populacao': [230000, 500000],
    'geometry': [Point(-8.611, 41.149), Point(-9.139, 38.722)]
}, crs="EPSG:4326")
cidades.to_file("cidades.geojson", driver="GeoJSON")

# 2. Estrada (Linha)
linha = gpd.GeoDataFrame({
    'nome': ['Estrada Porto-Lisboa'],
    'tipo': ['Autoestrada A1'],
    'geometry': [LineString([(-8.611, 41.149), (-9.139, 38.722)])]
}, crs="EPSG:4326")
linha.to_file("estrada.geojson", driver="GeoJSON")

# 3. Zona Urbana (Polígono)
zona = gpd.GeoDataFrame({
    'zona': ['Urbana Centro'],
    'uso': ['Residencial'],
    'geometry': [Polygon([
        (-9.14, 38.72), (-9.13, 38.72),
        (-9.13, 38.73), (-9.14, 38.73),
        (-9.14, 38.72)
    ])]
}, crs="EPSG:4326")
zona.to_file("zona.geojson", driver="GeoJSON")

# 4. MultiPolígono (Duas zonas separadas)
p1 = Polygon([(-9.2, 38.7), (-9.1, 38.7), (-9.1, 38.75), (-9.2, 38.75), (-9.2, 38.7)])
p2 = Polygon([(-9.0, 38.7), (-8.9, 38.7), (-8.9, 38.75), (-9.0, 38.75), (-9.0, 38.7)])
multi = MultiPolygon([p1, p2])
multipoligono = gpd.GeoDataFrame({
    'zona': ['Reserva Natural'],
    'uso': ['Protegido'],
    'geometry': [multi]
}, crs="EPSG:4326")
multipoligono.to_file("multipoligono.geojson", driver="GeoJSON")

# -----
# Criar Mapa com Folium
# -----

mapa = folium.Map(location=[39.5, -8.0], zoom_start=6)

# Adicionar cidades (pontos)
folium.GeoJson(
    "cidades.geojson",
    name="Cidades",
    style_function=lambda f: {'color': 'blue'},
    marker=folium.CircleMarker(radius=6),
    tooltip=folium.GeoJsonTooltip(fields=["cidade", "populacao"], aliases=["Cidade:", "População:"]),
    popup=folium.GeoJsonPopup(fields=["cidade", "populacao"])
).add_to(mapa)

# Adicionar estrada (linha)

```

```

folium.GeoJson(
    "estrada.geojson",
    name="Estrada",
    style_function=lambda f: {'color': 'red', 'weight': 3},
    tooltip=folium.GeoJsonTooltip(fields=["nome", "tipo"], aliases=["Nome:", "Tipo:"]),
    popup=folium.GeoJsonPopup(fields=["nome", "tipo"]))
).add_to(mapa)

# Adicionar zona urbana (polígono)
folium.GeoJson(
    "zona.geojson",
    name="Zona Urbana",
    style_function=lambda f: {'fillColor': 'orange', 'color': 'brown', 'fillOpacity': 0.5},
    tooltip=folium.GeoJsonTooltip(fields=["zona", "uso"], aliases=["Zona:", "Uso:"]),
    popup=folium.GeoJsonPopup(fields=["zona", "uso"]))
).add_to(mapa)

# Adicionar multipolígono
folium.GeoJson(
    "multipolygon.geojson",
    name="Reserva Natural",
    style_function=lambda f: {'fillColor': 'green', 'color': 'darkgreen', 'fillOpacity': 0.4},
    tooltip=folium.GeoJsonTooltip(fields=["zona", "uso"], aliases=["Zona:", "Uso:"]),
    popup=folium.GeoJsonPopup(fields=["zona", "uso"]))
).add_to(mapa)

# Adicionar controlo de camadas
folium.LayerControl().add_to(mapa)

# Guardar o mapa
mapa.save("mapa_interativo_completo.html")

```

## Bibliografia e Recursos

### Artigos e documentação:

- **RFC 7946 – The GeoJSON Format**  
<https://tools.ietf.org/html/rfc7946>  
→ Documento oficial que define o formato GeoJSON.
- **GeoJSON Specification (GitHub)**  
<https://github.com/geojson/geojson-spec>  
→ Repositório da especificação no GitHub com exemplos e discussões.
- **PostGIS Docs – Working with GeoJSON**  
[https://postgis.net/docs/ST\\_AsGeoJSON.html](https://postgis.net/docs/ST_AsGeoJSON.html)  
→ Mostra como manipular GeoJSON dentro de uma base de dados espacial.
- **Leaflet – GeoJSON Layer Documentation**  
<https://leafletjs.com/examples/geojson/>  
→ Explica como usar GeoJSON em mapas interativos com Leaflet.

### Livros:

1. **"Geographic Information Systems and Science"** – Paul A. Longley, Michael F. Goodchild, David J. Maguire, David W. Rhind  
ISBN: 978-1118676950  
→ Livro abrangente sobre SIG, com menções ao uso de formatos como o GeoJSON.

2. "**GIS Tutorial for Python Scripting**" – David W. Allen  
ISBN: 978-1589483569  
→ Apresenta o uso de GeoJSON com bibliotecas Python como geopandas e shapely.
3. "**Mastering Geospatial Analysis with Python**" – Silas Toms  
ISBN: 978-1788293334  
→ Inclui várias aplicações práticas de GeoJSON com Python e bibliotecas GIS.