Libraries

In [134…

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

import geopandas as gpd
import h3
import folium
import random

from folium.plugins import MarkerCluster
from shapely.geometry import Polygon, MultiPolygon, Point

from IPython.display import display
from scipy.stats import gaussian_kde

from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

Carregar e explorar os arquivos

In [2]:

```
gdf_ICV = gpd.read_file("ICV_argentina.geojson")
```

In [40]:

```
gdf_ICV.head(15)
```

Out[40]:

	fid	link	ICV2010	geometry
0	1	068821704	5.714253	MULTIPOLYGON (((-59.03972 -34.10997, -59.03948...
1	2	060141201	5.407843	MULTIPOLYGON (((-60.16071 -38.16385, -60.15364...
2	3	060210206	6.903273	MULTIPOLYGON (((-60.40995 -34.87205, -60.40859...
3	4	060070107	7.255807	MULTIPOLYGON (((-63.33826 -36.79329, -63.33999...
4	5	060210609	7.042759	MULTIPOLYGON (((-60.26759 -34.98781, -60.25127...
5	6	060420501	7.118543	MULTIPOLYGON (((-58.43691 -36.45290, -58.41922...
6	7	060421102	6.635552	MULTIPOLYGON (((-58.52328 -37.11311, -58.50552...
7	8	060562708	7.335448	MULTIPOLYGON (((-62.20458 -38.75006, -62.20389...
8	9	060630706	6.771001	MULTIPOLYGON (((-58.26926 -37.81713, -58.25577...
9	10	060630802	7.339134	MULTIPOLYGON (((-58.29873 -37.92856, -58.31576...
10	11	060700302	6.885195	MULTIPOLYGON (((-59.57712 -33.78316, -59.56776...
11	12	060980403	5.569138	MULTIPOLYGON (((-57.84951 -34.88152, -57.84849...
12	13	060980612	5.604852	MULTIPOLYGON (((-57.90032 -34.89049, -57.90001...
13	14	061051002	6.985777	MULTIPOLYGON (((-61.12149 -36.41290, -61.10516...
14	15	061051201	7.138411	MULTIPOLYGON (((-61.22761 -36.48427, -61.16842...

In [41]:

```
# Consultar os valores únicos da coluna geometry
valores_geometry = gdf_ICV['geometry'].unique()

# Inicializar conjuntos vazios para armazenar os tipos de geometria
tipos_geometria = set()

# Iterar sobre os valores únicos e verificar o tipo de cada geometria
for geometria in valores_geometry:
    # Verificar se a geometria é um MultiPolygon
    if isinstance(geometria, MultiPolygon):
        tipos_geometria.add('MULTIPOLYGON')
    # Verificar se a geometria é um Polygon
    elif isinstance(geometria, Polygon):
        tipos_geometria.add('POLYGON')
    # Se for outro tipo, adicionar ao conjunto de outros tipos
    else:
        tipos_geometria.add(type(geometria).__name__)

# Exibir os tipos de geometria encontrados
print("Tipos de geometria presentes na coluna 'geometry':", tipos_geometria)
```

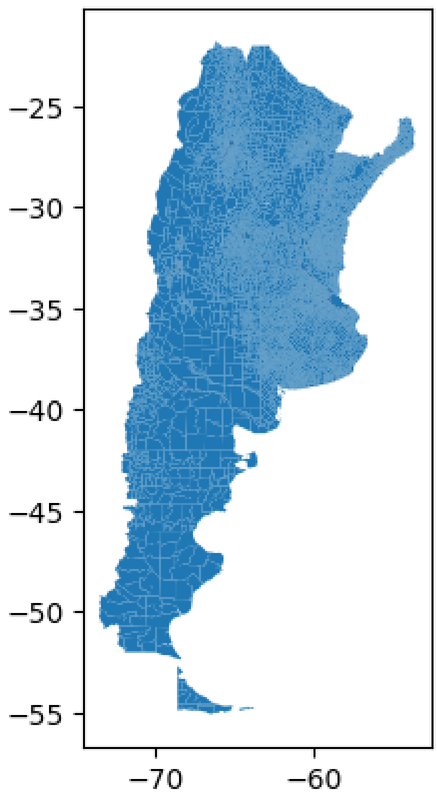
```
Tipos de geometria presentes na coluna 'geometry': {'MULTIPOLYGON'}
```

In [4]:

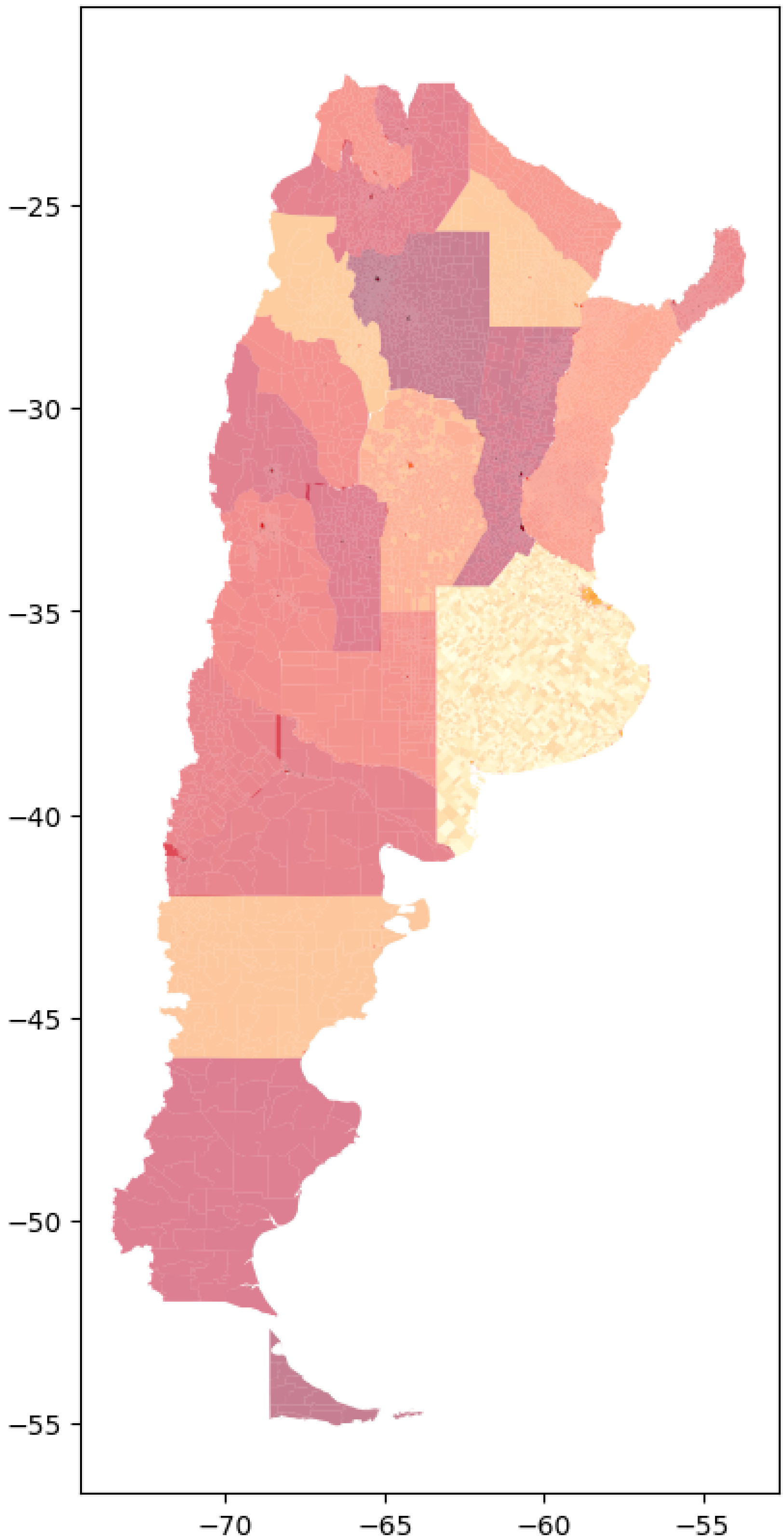
```
gdf_ICV.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 52408 entries, 0 to 52407
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   fid       52408 non-null  int64
 1   link      52408 non-null  object
 2   ICV2010   52395 non-null  float64
 3   geometry  52408 non-null  geometry
dtypes: float64(1), geometry(1), int64(1), object(1)
memory usage: 1.6+ MB
```

In [14]:
```python
# Visualização Simples
gdf.plot()
plt.show()
```



In [12]:
```python
# Mapa de Calor (Heatmap)
gdf.plot(figsize=(10, 10), cmap='YlOrRd', alpha=0.5)
plt.show()
```

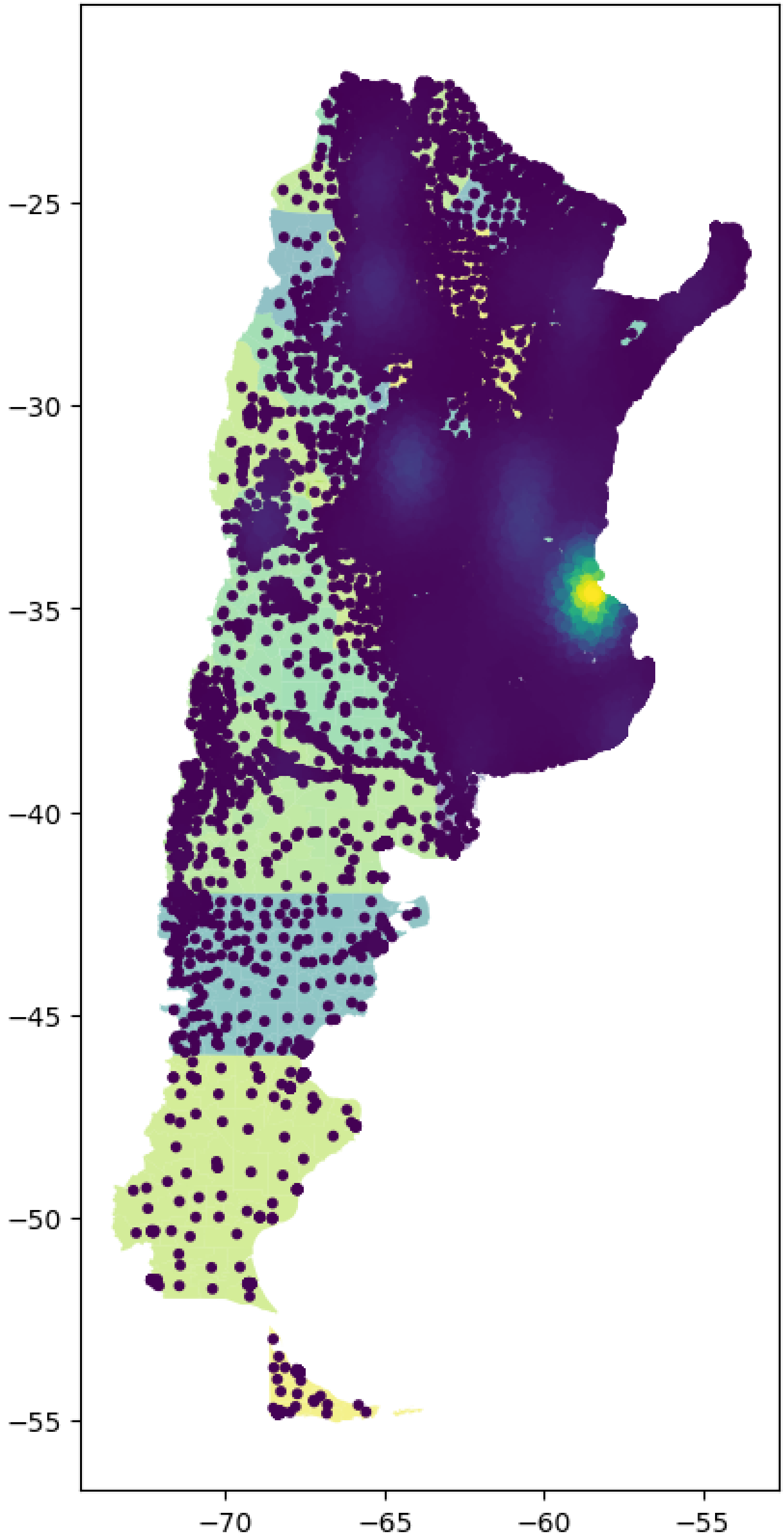


In [21]:

```
# Calcular densidade espacial usando os centroides
centroides_x = gdf.geometry.centroid.x
centroides_y = gdf.geometry.centroid.y

xy = np.vstack([centroides_x, centroides_y])
z = gaussian_kde(xy)(xy)

# Visualizar mapa de densidade
gdf.plot(figsize=(10, 10), cmap='viridis', alpha=0.5)
```

```python
plt.scatter(centroides_x, centroides_y, c=z, s=10, cmap='viridis')
plt.show()
```

```
C:\Users\lucci\AppData\Local\Temp\ipykernel_34876\77519814.py:2: UserWarning: Geometr
y is in a geographic CRS. Results from 'centroid' are likely incorrect. Use 'GeoSerie
s.to_crs()' to re-project geometries to a projected CRS before this operation.

  centroides_x = gdf.geometry.centroid.x
C:\Users\lucci\AppData\Local\Temp\ipykernel_34876\77519814.py:3: UserWarning: Geometr
y is in a geographic CRS. Results from 'centroid' are likely incorrect. Use 'GeoSerie
s.to_crs()' to re-project geometries to a projected CRS before this operation.

  centroides_y = gdf.geometry.centroid.y
```



Population

In [11]:
```
gdf_p = gpd.read_file("population_2010.geojson")
```

In [24]:
```
gdf_p.head()
```

Out[24]:

	fid	link	population_2010	geometry
0	1	068821704	2003.0	MULTIPOLYGON (((-59.03972 -34.10997, -59.03948...
1	2	060141201	56.0	MULTIPOLYGON (((-60.16071 -38.16385, -60.15364...
2	3	060210206	177.0	MULTIPOLYGON (((-60.40995 -34.87205, -60.40859...
3	4	060070107	216.0	MULTIPOLYGON (((-63.33826 -36.79329, -63.33999...
4	5	060210609	80.0	MULTIPOLYGON (((-60.26759 -34.98781, -60.25127...

In [25]:

```
gdf_p.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 52408 entries, 0 to 52407
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype   
---  ------           --------------  -----   
 0   fid              52408 non-null  int64   
 1   link             52408 non-null  object  
 2   population_2010  52379 non-null  float64 
 3   geometry         52408 non-null  geometry
dtypes: float64(1), geometry(1), int64(1), object(1)
memory usage: 1.6+ MB
```

Explorar e transformar dataset mobile_data_cordoba.csv (Latitude e Longitude)

In [21]:

```
def corrigir_coordenadas(latitude, longitude):
    try:
        lat = float(latitude)
        lon = float(longitude)
        if -90 <= lat <= 90 and -180 <= lon <= 180:
            return lat, lon
    except ValueError:
        pass
    return None, None

# Leia o dataset
df_mobile = pd.read_csv('mobile_data_cordoba.csv')

# Aplique a correção nas colunas de latitude e longitude
df_mobile[['latitude_corrigida', 'longitude_corrigida']] = df_mobile.apply(lambda row:

# Filtre as linhas com coordenadas válidas
df_valido = df_mobile.dropna(subset=['latitude_corrigida', 'longitude_corrigida'])

# Salve o dataset corrigido
df_valido.to_csv('dataset_corrigido.csv', index=False)  # Substitua 'dataset_corrigido
```

In [22]:

```
df_mobile.head()
```

Out[22]:

	Unnamed: 0	timestamp	device_aid	latitude	longitude	latitude_corrigida	longitude_corrigida
0	16	1680390000	d60c0141-709b-6708-a861-e96a90bad0f0	-31.404471	-64.195906	-31.404471	-64.19590
1	27	1680390000	b80f9ec1-619f-60fa-aa23-3c2e44f429df	-31.358208	-64.242614	-31.358208	-64.24261
2	43	1680390000	477fc43c-4cda-67d0-b3a2-61f144cf2af1	-31.377321	-64.211549	-31.377321	-64.21154
3	80	1680390000	bb07628a-9c04-6f1e-b114-dfaa90f022b6	-31.312654	-64.309352	-31.312654	-64.30935
4	125	1680390000	e0c32fa2-7f99-60b6-26b6-3b2c0ea72949	-31.409300	-64.201040	-31.409300	-64.20104

In [23]:

```
df_valido.head()
```

Out[23]:

	Unnamed: 0	timestamp	device_aid	latitude	longitude	latitude_corrigida	longitude_corrigida
0	16	1680390000	d60c0141-709b-6708-a861-e96a90bad0f0	-31.404471	-64.195906	-31.404471	-64.19590
1	27	1680390000	b80f9ec1-619f-60fa-aa23-3c2e44f429df	-31.358208	-64.242614	-31.358208	-64.24261
2	43	1680390000	477fc43c-4cda-67d0-b3a2-61f144cf2af1	-31.377321	-64.211549	-31.377321	-64.21154
3	80	1680390000	bb07628a-9c04-6f1e-b114-dfaa90f022b6	-31.312654	-64.309352	-31.312654	-64.30935
4	125	1680390000	e0c32fa2-7f99-60b6-26b6-3b2c0ea72949	-31.409300	-64.201040	-31.409300	-64.20104

Preparação/Junção df mobile com ICV 2010 Argentina (clustering)

In [24]:
```python
# Convertendo o DataFrame regular em um GeoDataFrame
geometry = [Point(xy) for xy in zip(df_valido['longitude_corrigida'], df_valido['latit
gdf_valido = gpd.GeoDataFrame(df_valido, geometry=geometry, crs="EPSG:4326")

# Realizar a junção espacial
resultado = gpd.sjoin(gdf_valido, gdf_ICV, how='left', op='within')

# Selecionar apenas as colunas necessárias do resultado
res_final_mob = resultado[['longitude_corrigida', 'latitude_corrigida', 'device_aid',

res_final_mob.head()
```

```
C:\Users\lucci\anaconda3\Lib\site-packages\IPython\core\interactiveshell.py:3445: Fut
ureWarning: The `op` parameter is deprecated and will be removed in a future release.
Please use the `predicate` parameter instead.
  if await self.run_code(code, result, async_=asy):
```

Out[24]:

	longitude_corrigida	latitude_corrigida	device_aid	timestamp	fid	link	ICV2010	geo
0	-64.195906	-31.404471	d60c0141-709b-6708-a861-e96a90bad0f0	1680390000	28642	140144712	8.767856	(-64 -31.
1	-64.242614	-31.358208	b80f9ec1-619f-60fa-aa23-3c2e44f429df	1680390000	29158	140142310	9.044272	(-64 -31.
2	-64.211549	-31.377321	477fc43c-4cda-67d0-b3a2-61f144cf2af1	1680390000	28940	140142509	8.130223	(-64 -31.
3	-64.309352	-31.312654	bb07628a-9c04-6f1e-b114-dfaa90f022b6	1680390000	28303	140210609	7.519565	(-64 -31.
4	-64.201040	-31.409300	e0c32fa2-7f99-60b6-26b6-3b2c0ea72949	1680390000	28629	140145810	8.898264	(-64 -31.

In [25]:
```python
res_final_mob.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
Int64Index: 273830 entries, 0 to 273829
Data columns (total 8 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   longitude_corrigida   273830 non-null  float64
 1   latitude_corrigida    273830 non-null  float64
 2   device_aid            273830 non-null  object
 3   timestamp             273830 non-null  int64
 4   fid                   273830 non-null  int64
 5   link                  273830 non-null  object
 6   ICV2010               273830 non-null  float64
 7   geometry              273830 non-null  geometry
dtypes: float64(3), geometry(1), int64(2), object(2)
memory usage: 18.8+ MB
```

Checar se device_aid ocorre mais de uma vez no dataset

In [41]:

```python
# Contagem de ocorrências de device_aid
device_aid_counts = res_final_mob['device_aid'].value_counts()

# Filtrar apenas as device_aid que aparecem mais de uma vez
duplicated_device_aid = device_aid_counts[device_aid_counts > 1]

# print("device_aid que aparecem mais de uma vez:")
print(duplicated_device_aid)
```

```
0c061210-0677-6781-ae8a-32d178b0ea20    1012
b99d74f1-2dc0-6f63-11b9-8ef4688060f6     890
4672c89d-8f4a-6037-1ee0-77c1a8a7f21a     872
c3f910e0-6129-62f4-b7e2-d79af90d2bab     770
80a4040f-ebe8-6c77-1240-c3c828a7faaa     678
                                        ...
2653e5da-eb4e-43ab-8158-9974e9efc8c8       2
80441c8d-27df-647c-b7bd-4da03afe021b       2
cf16f9af-95e1-4550-b215-6232f9ee69a9       2
a808fe80-8638-6eea-13b9-0ba476798ecb       2
f3116ed6-708e-6001-108d-f107eb760ec1       2
Name: device_aid, Length: 9129, dtype: int64
```

Explorar e transformar o dataset locations_cordoba.csv e poligonos

Transformar Latitude e Longitude

In [26]:

```python
def formatar_coordenadas(valor):
    try:
        # Tenta converter o valor para float
        return float(valor)
    except ValueError:
        # Se a conversão falhar, retorna NaN
        return float('nan')

def ajustar_intervalo(valor, limite_inferior, limite_superior):
    # Garante que o valor esteja dentro do intervalo especificado
    return max(min(valor, limite_superior), limite_inferior)
```

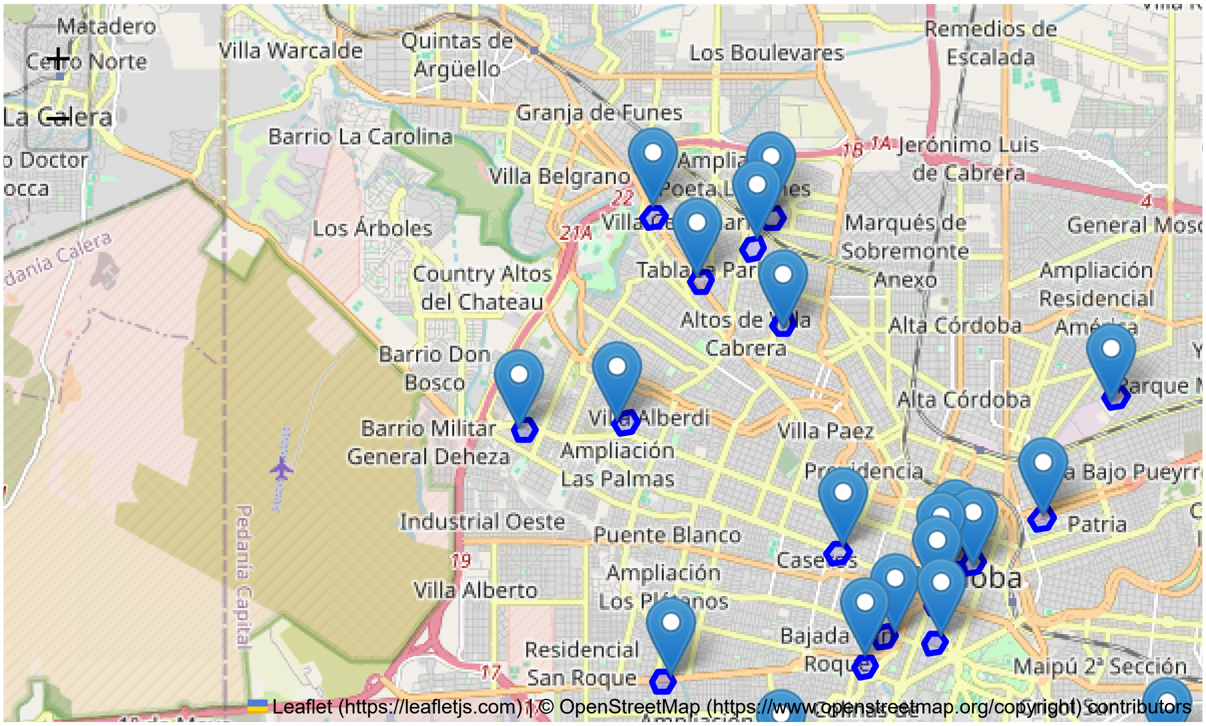
```python
def ajustar_latitude(valor):
    # Ajusta o valor para o intervalo de -90 a 90
    return ajustar_intervalo(valor, -90, 90)

def ajustar_longitude(valor):
    # Ajusta o valor para o intervalo de -180 a 180
    return ajustar_intervalo(valor, -180, 180)

# Carregue seu dataset usando o pandas (substitua 'seu_dataset.csv' pelo caminho corre
df_l = pd.read_csv('locations_cordoba.csv')

# Aplique as transformações nas colunas de Latitude e Longitude
df_l['Latitude'] = df_l['Latitude'].apply(formatar_coordenadas).apply(ajustar_latitude
df_l['Longitude'] = df_l['Longitude'].apply(formatar_coordenadas).apply(ajustar_longit

# Exiba o DataFrame resultante
df_l.head()
```

Out[26]:

	Unnamed: 0	Longitude	Latitude	H3_ID	H3_Geometry
0	0	-64.18179	-31.41539	89b243705dbffff	((-31.413365178428585, -64.1825952011923), (-3...
1	1	-64.18488	-31.41444	89b243704afffff	((-31.412117942732255, -64.18540729171282), (-...
2	2	-64.18799	-31.41968	89b2437042fffff	((-31.419424492502117, -64.18876200920899), (-...
3	3	-64.20396	-31.41267	89b2437045bffff	((-31.411939367704555, -64.20563685090073), (-...
4	4	-64.18743	-31.42574	89b24370087ffff	((-31.425126582326705, -64.18912374250645), (-...

In [27]:

```python
# Criar o mapa
M_loc = folium.Map(location=[df_l['Latitude'].mean(), df_l['Longitude'].mean()], zoom_

# Adicionar marcadores e polígonos para cada linha no dataset
for index, row in df_l.iterrows():
    # Adicionar marcador para o ponto inicial
    folium.Marker(location=[row['Latitude'], row['Longitude']], popup='Ponto Inicial')

    # Adicionar polígono ao mapa
    folium.Polygon(locations=eval(row['H3_Geometry']), color='blue', fill=True, fill_c

# Exibir o mapa
# mymap.save('mapa_interativo.html')
display(M_loc)
# M_loc
```



Preparação/Junção df locations com ICV 2010 Argentina (clustering)

In [30]:

```python
# Convertendo o DataFrame regular em um GeoDataFrame
geometry = [Point(xy) for xy in zip(df_l['Longitude'], df_l['Latitude'])]
gdf_loc = gpd.GeoDataFrame(df_l, geometry=geometry, crs="EPSG:4326")

# Realizar a junção espacial
resultado = gpd.sjoin(gdf_loc, gdf_ICV, how='left', op='within')

# Selecionar apenas as colunas necessárias do resultado
res_final_loc = resultado[['Longitude', 'Latitude', 'H3_ID', 'H3_Geometry', 'fid', 'li

res_final_loc.head()
```

```
C:\Users\lucci\anaconda3\Lib\site-packages\IPython\core\interactiveshell.py:3445: FutureWarning: The `op` parameter is deprecated and will be removed in a future release. Please use the `predicate` parameter instead.
  if await self.run_code(code, result, async_=asy):
```

Out[30]:

	Longitude	Latitude	H3_ID	H3_Geometry	fid	link	ICV2010	geometr
0	-64.18179	-31.41539	89b243705dbffff	((-31.413365178428585, -64.1825952011923), (-3...	29058	140145509	8.735894	POIN (-64.1817 -31.4153
1	-64.18488	-31.41444	89b243704afffff	((-31.412117942732255, -64.18540729171282), (-...	28988	140145708	8.773664	POIN (-64.1848 -31.4144
2	-64.18799	-31.41968	89b2437042fffff	((-31.419424492502117, -64.18876200920899), (-...	28525	140146704	9.336821	POIN (-64.1879 -31.4196
3	-64.20396	-31.41267	89b2437045bffff	((-31.411939367704555, -64.20563685090073), (-...	28552	140146505	9.315523	POIN (-64.2039 -31.4126
4	-64.18743	-31.42574	89b24370087ffff	((-31.425126582326705, -64.18912374250645), (-	28489	140147005	9.641398	POIN (-64.1874 -31.4257

In [31]:

```
res_final_loc.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
Int64Index: 22 entries, 0 to 21
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Longitude    22 non-null     float64
 1   Latitude     22 non-null     float64
 2   H3_ID        22 non-null     object
 3   H3_Geometry  22 non-null     object
 4   fid          22 non-null     int64
 5   link         22 non-null     object
 6   ICV2010      22 non-null     float64
 7   geometry     22 non-null     geometry
dtypes: float64(3), geometry(1), int64(1), object(3)
memory usage: 1.5+ KB
```

In [87]:

```
df_mob = res_final_mob[['longitude_corrigida','latitude_corrigida','fid','link','ICV20
df_loc = res_final_loc[['Longitude', 'Latitude','fid','link','ICV2010']].copy()
```

In [33]:

```
df_loc
```

Out[33]:

	Longitude	Latitude	fid	link	ICV2010
0	-64.18179	-31.41539	29058	140145509	8.735894
1	-64.18488	-31.41444	28988	140145708	8.773664
2	-64.18799	-31.41968	28525	140146704	9.336821
3	-64.20396	-31.41267	28552	140146505	9.315523
4	-64.18743	-31.42574	28489	140147005	9.641398
5	-64.19517	-31.42520	28496	140146911	8.447601
6	-64.18100	-31.45120	29278	140141501	9.545728
7	-64.21454	-31.38071	28933	140142516	9.174258
8	-64.21648	-31.36327	29581	140140212	8.706018
9	-64.21897	-31.36741	28948	140142501	9.373766
10	-64.22903	-31.37294	28955	140142407	9.202750
11	-64.23657	-31.36280	28949	140142414	9.127301
12	-64.24280	-31.39408	28771	140144308	5.830760
13	-64.25975	-31.39535	29149	140142216	8.506356
14	-64.23350	-31.43162	28138	140148513	7.118108
15	-64.21476	-31.44498	28049	140149013	6.771470
16	-64.16993	-31.40820	29126	140144908	8.106499
17	-64.13219	-31.42687	28742	140147705	8.051623
18	-64.14842	-31.44331	28445	140149307	7.318817
19	-64.15809	-31.39131	28907	140143419	8.110630
20	-64.20028	-31.42866	28172	140148306	7.608107
21	-64.18720	-31.41620	28526	140146703	9.197365

In [88]:

```
df_mob = df_mob.rename(columns={'longitude_corrigida': 'Longitude', 'latitude_corrigic
df_mob.head()
```

Out[88]:

	Longitude	Latitude	fid	link	ICV2010	device_aid	timestamp
0	-64.195906	-31.404471	28642	140144712	8.767856	d60c0141-709b-6708-a861-e96a90bad0f0	1680390000
1	-64.242614	-31.358208	29158	140142310	9.044272	b80f9ec1-619f-60fa-aa23-3c2e44f429df	1680390000
2	-64.211549	-31.377321	28940	140142509	8.130223	477fc43c-4cda-67d0-b3a2-61f144cf2af1	1680390000
3	-64.309352	-31.312654	28303	140210609	7.519565	bb07628a-9c04-6f1e-b114-dfaa90f022b6	1680390000
4	-64.201040	-31.409300	28629	140145810	8.898264	e0c32fa2-7f99-60b6-26b6-3b2c0ea72949	1680390000

Gerando hex_ids em df_mob (mapas centros comerciales)

In [89]:

```
# Normalizando as latitudes e longitudes
def normalize_coordinates(latitude, longitude, decimal_places):
    """
    Normaliza as coordenadas para o mesmo número de casas decimais.

    :param latitude: Latitude
    :param longitude: Longitude
    :param decimal_places: Número de casas decimais desejado
    :return: Tupla contendo as coordenadas normalizadas
    """
    lat_normalized = round(latitude, decimal_places)
    long_normalized = round(longitude, decimal_places)
    return lat_normalized, long_normalized

# Normalizando as coordenadas para 5 casas decimais
# df_mob[['latitude_normalized', 'longitude_normalized']] = df_mob.apply(lambda row: n
df_mob[['latitude_normalized', 'longitude_normalized']] = df_mob.apply(lambda row: pd.

df_mob.head()
```

Out[89]:

	Longitude	Latitude	fid	link	ICV2010	device_aid	timestamp	latitude_normalized
0	-64.195906	-31.404471	28642	140144712	8.767856	d60c0141-709b-6708-a861-e96a90bad0f0	1680390000	-31.40447
1	-64.242614	-31.358208	29158	140142310	9.044272	b80f9ec1-619f-60fa-aa23-3c2e44f429df	1680390000	-31.35821
2	-64.211549	-31.377321	28940	140142509	8.130223	477fc43c-4cda-67d0-b3a2-61f144cf2af1	1680390000	-31.37732
3	-64.309352	-31.312654	28303	140210609	7.519565	bb07628a-9c04-6f1e-b114-dfaa90f022b6	1680390000	-31.31265
4	-64.201040	-31.409300	28629	140145810	8.898264	e0c32fa2-7f99-60b6-26b6-3b2c0ea72949	1680390000	-31.40930

In [95]:

```
# Função para gerar o hex_id com base na latitude e longitude
def generate_hex_id(row):
    lat = row['Latitude']
    lon = row['Longitude']
    hex_id = h3.geo_to_h3(lat, lon, resolution=9)  # Escolha a resolução adequada
    return hex_id

# Gerar df_mob com hex_ids
# df_mob_hex = df_mob[['Longitude','Latitude','fid','link','ICV2010']].copy()
df_mob_hex = df_mob[['longitude_normalized','latitude_normalized','fid','link','ICV201
df_mob_hex = df_mob_hex.rename(columns={'longitude_normalized': 'Longitude', 'latitude

# Adicionar uma nova coluna com o hex_id
df_mob_hex['hex_id'] = df_mob_hex.apply(generate_hex_id, axis=1)

# Visualizar o dataframe com os hex_ids
# print(df)
df_mob_hex.head()
```

Out[95]:

	Longitude	Latitude	fid	link	ICV2010	device_aid	timestamp	hex_id
0	-64.19591	-31.40447	28642	140144712	8.767856	d60c0141-709b-6708-a861-e96a90bad0f0	1680390000	89b243704d7ffff
1	-64.24261	-31.35821	29158	140142310	9.044272	b80f9ec1-619f-60fa-aa23-3c2e44f429df	1680390000	89b24309a4bffff
2	-64.21155	-31.37732	28940	140142509	8.130223	477fc43c-4cda-67d0-b3a2-61f144cf2af1	1680390000	89b2437358bffff
3	-64.30935	-31.31265	28303	140210609	7.519565	bb07628a-9c04-6f1e-b114-dfaa90f022b6	1680390000	89b2434609bffff
4	-64.20104	-31.40930	28629	140145810	8.898264	e0c32fa2-7f99-60b6-26b6-3b2c0ea72949	1680390000	89b243704cbffff

In [36]:

```
duplicates = df_mob_hex['hex_id'].duplicated().any()

if duplicates:
    print("Existem hex_ids duplicados no dataframe.")
else:
    print("Não existem hex_ids duplicados no dataframe.")
```

```
Existem hex_ids duplicados no dataframe.
```

In [38]:

```
# Encontrar e contar os hex_ids duplicados
duplicates_mask = df_mob_hex.duplicated(subset=['hex_id'], keep=False)
duplicates_df = df_mob_hex[duplicates_mask]

# Contagem de hex_ids duplicados
duplicates_count = duplicates_df.groupby('hex_id').size()
duplicates_count = duplicates_count[duplicates_count > 1]

print("Hex_ids com mais de uma linha e sua contagem:")
print(duplicates_count)
```

```
Hex_ids com mais de uma linha e sua contagem:
hex_id
85b2430bfffffff     27685
85b24347fffffff     44940
85b24373fffffff    201205
dtype: int64
```

Clustering K-means

In [120…

```
# Preparando df mobile
df_mob1 = df_mob[['Longitude','Latitude','fid','link','ICV2010']].copy()

# Concatenar os DataFrames se necessário
df_conc = pd.concat([df_mob1, df_loc], ignore_index=True)
```

In [123…

```
# Limites geográficos de Córdoba, Argentina (aproximados)
cordoba_lat_min = -31.5
cordoba_lat_max = -31.3
cordoba_lon_min = -64.4
cordoba_lon_max = -64.1

# Filtrando o DataFrame para obter apenas os dados de Córdoba
cordoba_df = df_conc[(df_conc['Latitude'] >= cordoba_lat_min) & (df_conc['Latitude'] <
                (df_conc['Longitude'] >= cordoba_lon_min) & (df_conc['Longitude'] <= c
```

In [130…

```
# Normalizar os dados (opcional, dependendo do algoritmo de clustering escolhido)
scaler = StandardScaler()
df_normalized = scaler.fit_transform(cordoba_df)

# Escolher o número de clusters (ou utilizar algum método para determinar o número óti
num_clusters = 7

# Aplicar o algoritmo de clustering (K-Means, neste exemplo)
kmeans = KMeans(n_clusters=num_clusters)
kmeans.fit(df_normalized)

# Adicionar os rótulos dos clusters ao DataFrame concatenado
cordoba_df['Cluster'] = kmeans.labels_

cordoba_df
```

```
C:\Users\lucci\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarn
ing: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the valu
e of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Users\lucci\AppData\Local\Temp\ipykernel_50264\2626333650.py:13: SettingWithCopyWa
rning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  cordoba_df['Cluster'] = kmeans.labels_
```

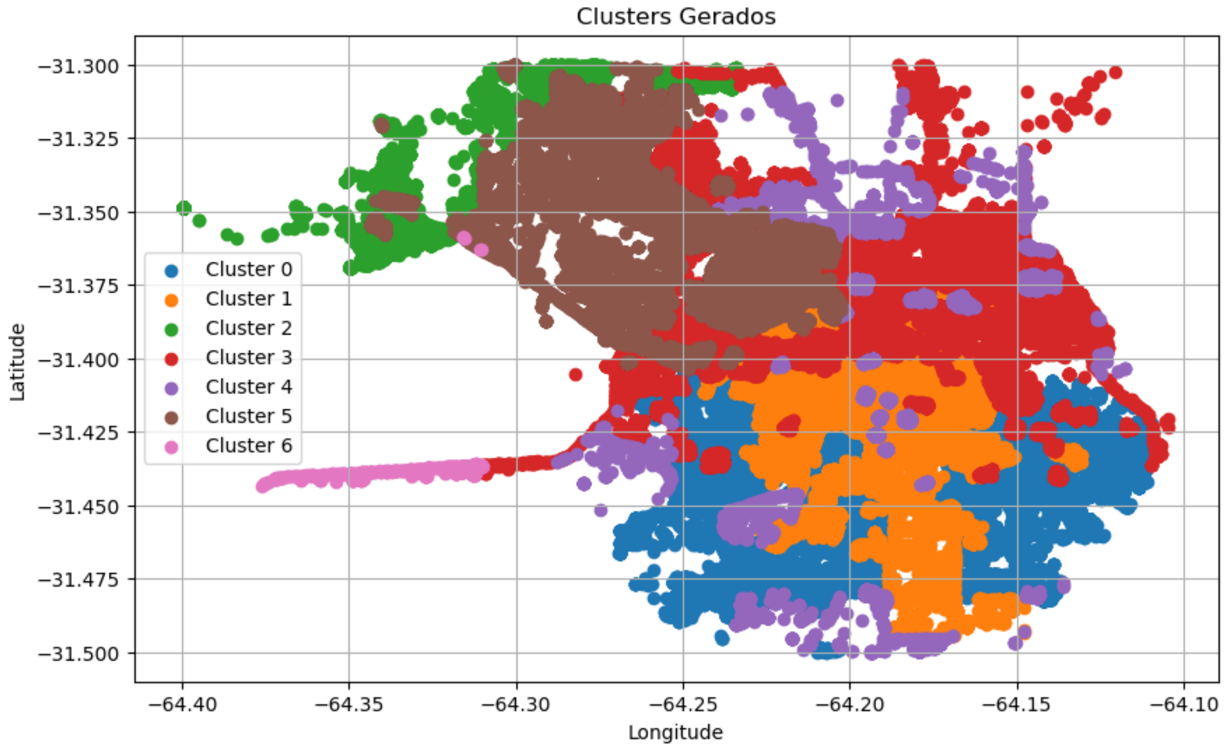
Out[130]:

	Longitude	Latitude	fid	link	ICV2010	Cluster
0	-64.195906	-31.404471	28642	140144712	8.767856	1
1	-64.242614	-31.358208	29158	140142310	9.044272	5
2	-64.211549	-31.377321	28940	140142509	8.130223	3
3	-64.309352	-31.312654	28303	140210609	7.519565	2
4	-64.201040	-31.409300	28629	140145810	8.898264	1
...	...	...	...	...	...	...
273847	-64.132190	-31.426870	28742	140147705	8.051623	0
273848	-64.148420	-31.443310	28445	140149307	7.318817	0
273849	-64.158090	-31.391310	28907	140143419	8.110630	3
273850	-64.200280	-31.428660	28172	140148306	7.608107	0
273851	-64.187200	-31.416200	28526	140146703	9.197365	1

271351 rows × 6 columns

In [131…

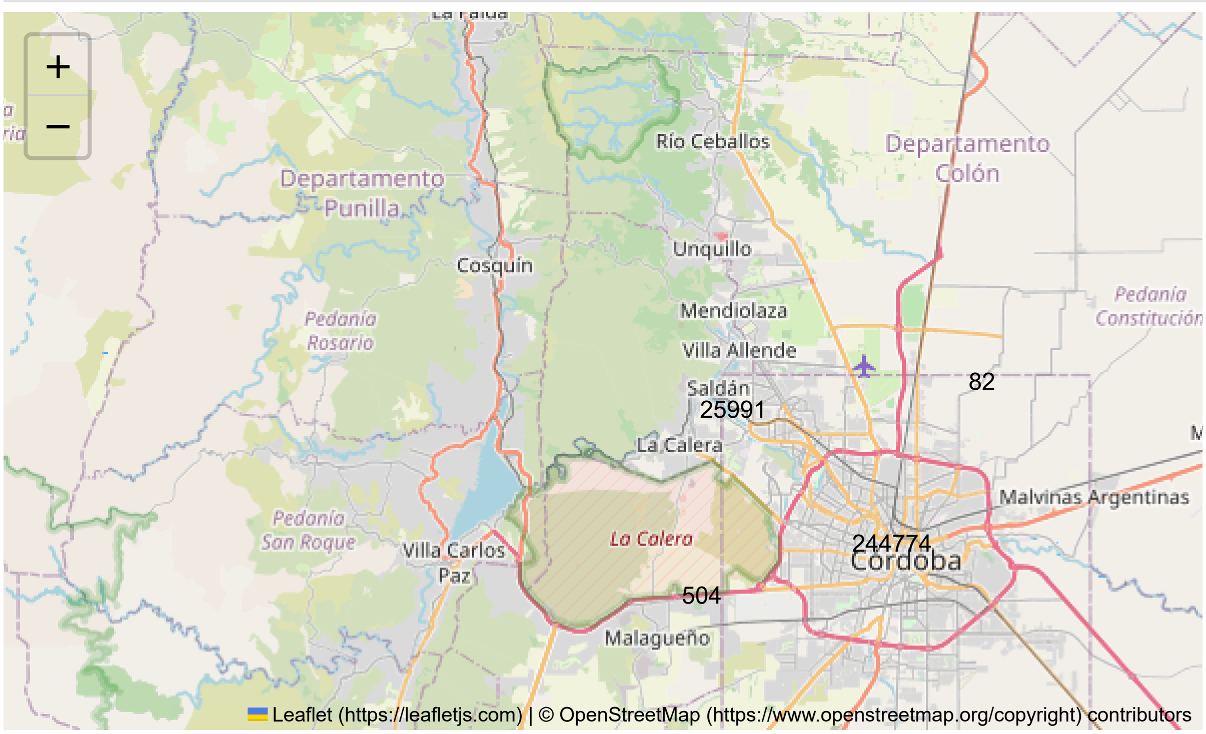
```
# Plotar os clusters
plt.figure(figsize=(10, 6))
for cluster in range(num_clusters):
    cluster_points = cordoba_df[cordoba_df['Cluster'] == cluster]
    plt.scatter(cluster_points['Longitude'], cluster_points['Latitude'], label=f'Clust
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Clusters Gerados')
plt.legend()
plt.grid(True)
plt.show()
```



In [133…

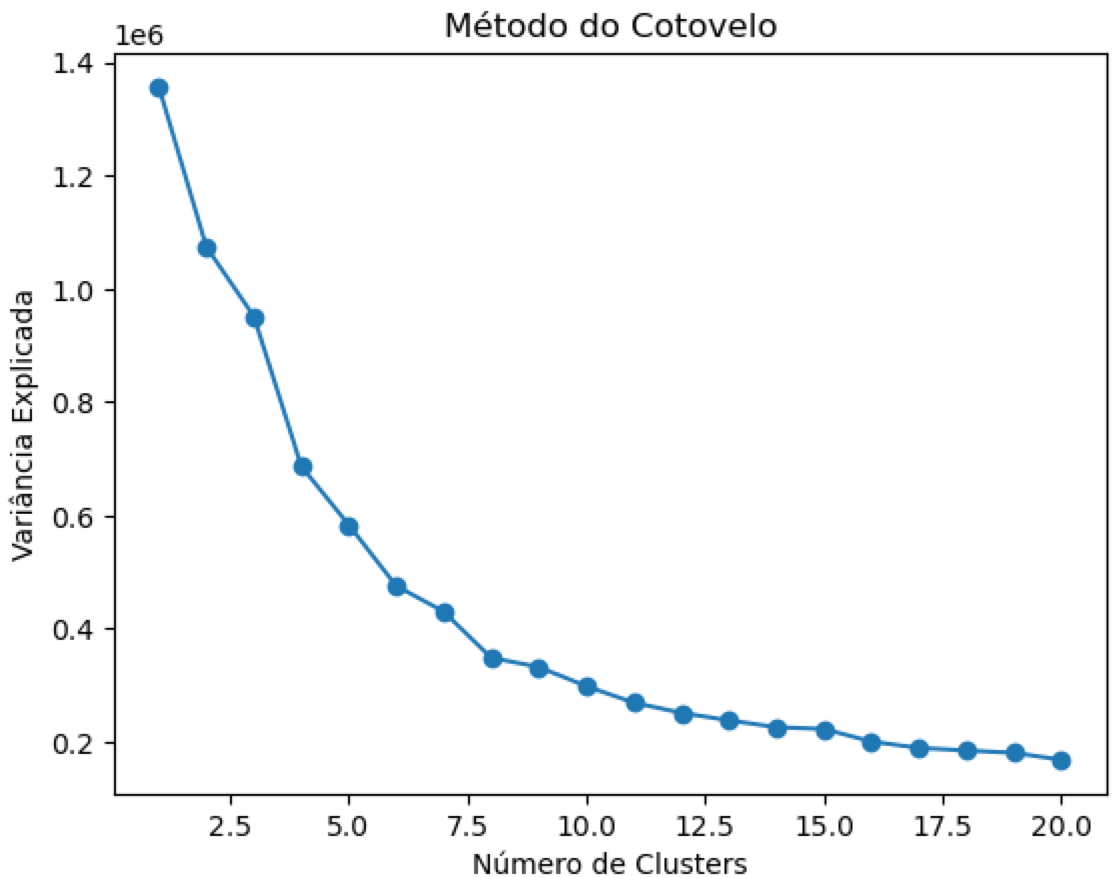
```
# Usar Folium MarkerCluster
mapa = folium.Map(location=[cordoba_df['Latitude'].mean(), cordoba_df['Longitude'].mea

# Adicionar marcadores para cada ponto
marker_cluster = MarkerCluster().add_to(mapa)
for index, row in cordoba_df.iterrows():  # Usar cordoba_df em vez de df_conc
    folium.Marker([row['Latitude'], row['Longitude']]).add_to(marker_cluster)

# Exibir o mapa
mapa
```

Out[133]:



In [127…

```
# Lista para armazenar a variância explicada
variancia_explicada = []

# Testar diferentes valores de k
for k in range(1, 21):
    kmeans = KMeans(n_clusters=k, n_init='auto')
    kmeans.fit(df_normalized)
    variancia_explicada.append(kmeans.inertia_)

# Plotar o gráfico do método do cotovelo
plt.plot(range(1, 21), variancia_explicada, marker='o')
plt.xlabel('Número de Clusters')
plt.ylabel('Variância Explicada')
plt.title('Método do Cotovelo')
plt.show()
```



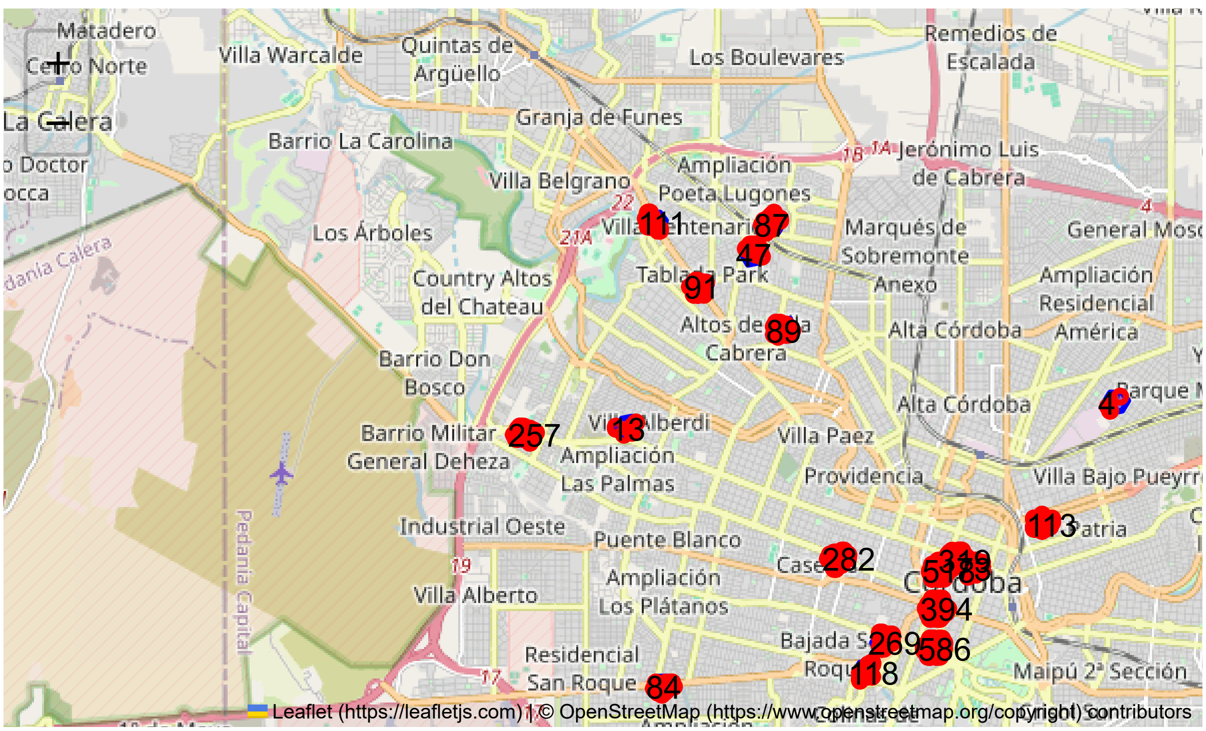
H3 library - Exploração e grupos

In [85]:

```
# Função para criar um mapa Folium com células hexagonais coloridas e pontos dentro de
def create_hex_map_with_points(df_hexagons, df_points):
    # Centraliza o mapa nas coordenadas médias das células hexagonais
    avg_lat = df_hexagons['Latitude'].mean()
    avg_lon = df_hexagons['Longitude'].mean()
    m = folium.Map(location=[avg_lat, avg_lon], zoom_start=12)

    # Itera sobre os hex_ids para adicionar células hexagonais ao mapa
    for hex_id in df_hexagons['H3_ID']:
        # Converte o hex_id para coordenadas geográficas do hexágono
        polygon = h3.h3_to_geo_boundary(hex_id)
```

```python
        # Adiciona o hexágono ao mapa
        folium.Polygon(locations=polygon, color='blue', fill_color='blue', fill_opacit

        # Identifica os pontos dentro do hexágono
        points_inside_hex = df_points[df_points['hex_id'] == hex_id]

        # Adiciona os pontos ao mapa
        for _, point in points_inside_hex.iterrows():
            folium.CircleMarker(location=[point['Latitude'], point['Longitude']], radi

        # Adiciona o número de pontos como um rótulo de texto
        folium.Marker(location=[polygon[0][0], polygon[0][1]], icon=folium.DivIcon(htm

    return m

# Cria um mapa Folium com as células hexagonais coloridas e pontos dentro de cada hexá
hex_map_with_points = create_hex_map_with_points(df_l, df_mob_hex)

# Salva o mapa em um arquivo HTML
hex_map_with_points.save('hex_map_with_points.html')

# Abre o arquivo HTML no navegador padrão
import webbrowser
webbrowser.open('hex_map_with_points.html')

hex_map_with_points
```

Out[85]:



Cria novo dataframe para responder perguntas

In [96]:

```python
# Função para criar um mapa Folium com células hexagonais coloridas e pontos dentro de
def create_hex_map_with_points(df_hexagons, df_points):
    # Centraliza o mapa nas coordenadas médias das células hexagonais
    avg_lat = df_hexagons['Latitude'].mean()
    avg_lon = df_hexagons['Longitude'].mean()
    m = folium.Map(location=[avg_lat, avg_lon], zoom_start=12)
```

```
    # Lista para armazenar os pontos com a nova coluna 'main_hex_id'
    new_points = []

    # Itera sobre os hex_ids para adicionar células hexagonais ao mapa
    for hex_id in df_hexagons['H3_ID']:
        # Converte o hex_id para coordenadas geográficas do hexágono
        polygon = h3.h3_to_geo_boundary(hex_id)

        # Adiciona o hexágono ao mapa
        folium.Polygon(locations=polygon, color='blue', fill_color='blue', fill_opacit

        # Identifica os pontos dentro do hexágono
        points_inside_hex = df_points[df_points['hex_id'] == hex_id]

        # Adiciona os pontos ao mapa e atualiza o DataFrame com a nova coluna 'main_he
        for _, point in points_inside_hex.iterrows():
            folium.CircleMarker(location=[point['Latitude'], point['Longitude']], radi
            point['main_hex_id'] = hex_id
            new_points.append(point)

    # Cria um novo DataFrame com os pontos e a nova coluna 'main_hex_id'
    df_new_points = pd.DataFrame(new_points)

    return m, df_new_points

# Cria um mapa Folium com as células hexagonais coloridas e pontos dentro de cada hexá
hex_map_with_points, df_new_points = create_hex_map_with_points(df_l, df_mob_hex)

# Salva o mapa em um arquivo HTML
hex_map_with_points.save('hex_map_with_points.html')

# Salva o novo DataFrame em um arquivo CSV
# df_new_points.to_csv('novo_dataframe_com_main_hex_id.csv', index=False)

# Abre o arquivo HTML no navegador padrão
# import webbrowser
# webbrowser.open('hex_map_with_points.html')
```

Out[96]: True

In [97]:
```
df_new_points
```

Out[97]:

	Longitude	Latitude	fid	link	ICV2010	device_aid	timestamp	hex_i
6742	-64.18099	-31.41577	29058	140145509	8.735894	bfaf22b8-8363-60b9-16d0-d7dda6b4a277	1680391547	89b243705dbfff
15087	-64.18237	-31.41569	29058	140145509	8.735894	20e10b93-fc1a-60f7-a2a3-92a6aa9b889d	1680386417	89b243705dbfff
15143	-64.18238	-31.41569	29058	140145509	8.735894	20e10b93-fc1a-60f7-a2a3-92a6aa9b889d	1680386427	89b243705dbfff
15223	-64.18241	-31.41568	29058	140145509	8.735894	20e10b93-fc1a-60f7-a2a3-92a6aa9b889d	1680386441	89b243705dbfff
17493	-64.18176	-31.41408	29058	140145509	8.735894	4e010821-1ce2-668f-2617-100ac8ff219a	1680386775	89b243705dbfff
...	...	...	...	...	...	...	...	.
271802	-64.18882	-31.41533	28527	140146702	9.521492	2ecc8520-001f-411d-8085-7d052ddc860e	1680908042	89b24370433fff
272166	-64.18788	-31.41537	28526	140146703	9.197365	4b9d0896-3c0f-6430-a791-3ece61af638f	1680908097	89b24370433fff
272811	-64.18642	-31.41390	28989	140145707	9.258276	a7918980-216b-64fa-bddf-9d733d302d40	1680908206	89b24370433fff
273419	-64.18682	-31.41481	28986	140145710	9.233548	daa0b6a4-8f04-6234-1a7a-ed2d4d3e47ec	1680908325	89b24370433fff
273825	-64.18839	-31.41558	28526	140146703	9.197365	792a4e6b-3b09-6d48-29f0-d3a0b911ecaa	1680908399	89b24370433fff

3740 rows × 9 columns

In [76]:

```
# Cria um novo DataFrame com o contador agrupando por "main_hex_id"
df_counter = df_new_points.groupby('main_hex_id').size().reset_index(name='contador')
```

```
# Ordena o DataFrame pelo contador em ordem decrescente
df_counter = df_counter.sort_values(by='contador', ascending=False)

# Visualiza o novo DataFrame
df_counter
```

Out[76]:

	main_hex_id	contador
3	89b24370087ffff	586
9	89b24370433ffff	518
8	89b2437042fffff	394
11	89b243704afffff	319
10	89b2437045bffff	282
5	89b243700d7ffff	269
21	89b2437366bffff	257
4	89b243700cfffff	118
16	89b2437232bffff	113
0	89b24309a6bffff	111
7	89b24370203ffff	95
19	89b24373417ffff	91
20	89b243735c7ffff	89
2	89b24309b57ffff	87
14	89b24371513ffff	84
13	89b24370c87ffff	79
12	89b243705dbffff	73
6	89b2437016bffff	69
1	89b24309b4fffff	47
17	89b24372867ffff	42
18	89b24373053ffff	13
15	89b24372157ffff	4

In [98]:

```
# Cria um novo DataFrame com o contador agrupando por "main_hex_id" e "device_aid"
df_counter2 = df_new_points.groupby(['main_hex_id', 'device_aid']).size().reset_index(

# Ordena o DataFrame pelo contador em ordem decrescente
df_counter2 = df_counter2.sort_values(by='contador', ascending=False)

# Visualiza o novo DataFrame
df_counter2
```

Out[98]:

	main_hex_id	device_aid	contador
78	89b24370087ffff	3a04cd62-f9d2-6698-a1c4-92092cf8077b	128
133	89b243700d7ffff	7d13b2be-663a-68f3-a9ef-0010434d6977	108
354	89b243704afffff	4f0fa23a-b639-6da0-204c-b0fcf2fdf423	96
79	89b24370087ffff	3bdff12c-4d41-6db7-244b-8be8d0c98e66	89
285	89b24370433ffff	6e210af1-d2ba-67fc-ac9b-b03a76270700	87
...	...	...	...
255	89b2437042fffff	ee12b44c-3230-433b-824c-32275a7786b0	1
254	89b2437042fffff	ecec29ce-0948-62b0-29d0-b917bd3333de	1
252	89b2437042fffff	eab23041-0ea9-698c-2f90-a847db648bf0	1
251	89b2437042fffff	ea3a4bb3-86f8-452f-81c1-cc2a8ce18601	1
585	89b2437366bffff	f1f99dba-b142-4ac4-8c17-db6e559a1866	1

586 rows × 3 columns

In [100…

```
# Primeira pergunta ------------
# ¿Cuántos dispositivos únicos circulan por cada hexágono de nivel 9?

# Cria um novo DataFrame com o contador de dispositivos únicos agrupados por "main_hex
df_unique_devices = df_new_points.groupby('main_hex_id')['device_aid'].nunique().reset

# Visualiza o novo DataFrame
print(df_unique_devices)
```

```
          main_hex_id  unique_devices_count
0     89b24309a6bffff                    28
1     89b24309b4fffff                    16
2     89b24309b57ffff                    18
3     89b24370087ffff                    55
4     89b243700cfffff                    10
5     89b243700d7ffff                    17
6     89b2437016bffff                    20
7     89b24370203ffff                    15
8     89b2437042fffff                    80
9     89b24370433ffff                    51
10    89b2437045bffff                    31
11    89b243704afffff                    49
12    89b243705dbffff                    23
13    89b24370c87ffff                    21
14    89b24371513ffff                    21
15    89b24372157ffff                     3
16    89b2437232bffff                    26
17    89b24372867ffff                    12
18    89b24373053ffff                     6
19    89b24373417ffff                    37
20    89b243735c7ffff                    21
21    89b2437366bffff                    26
```

In [103…

```
# Ordena o DataFrame pelo número de vezes que o mesmo dispositivo circulou por cada he
df_device_counts = df_device_counts.sort_values(by='device_count', ascending=False)
```

```python
# Visualiza o novo DataFrame ordenado
print(df_device_counts)
```

```
          main_hex_id                            device_aid  device_count
78   89b24370087ffff  3a04cd62-f9d2-6698-a1c4-92092cf8077b           128
133  89b243700d7ffff  7d13b2be-663a-68f3-a9ef-0010434d6977           108
354  89b243704afffff  4f0fa23a-b639-6da0-204c-b0fcf2fdf423            96
79   89b24370087ffff  3bdff12c-4d41-6db7-244b-8be8d0c98e66            89
285  89b24370433ffff  6e210af1-d2ba-67fc-ac9b-b03a76270700            87
..                ...                                   ...           ...
255  89b2437042fffff  ee12b44c-3230-433b-824c-32275a7786b0             1
254  89b2437042fffff  ecec29ce-0948-62b0-29d0-b917bd3333de             1
252  89b2437042fffff  eab23041-0ea9-698c-2f90-a847db648bf0             1
251  89b2437042fffff  ea3a4bb3-86f8-452f-81c1-cc2a8ce18601             1
585  89b2437366bffff  f1f99dba-b142-4ac4-8c17-db6e559a1866             1

[586 rows x 3 columns]
```

In [104…

```python
# Ordena o DataFrame pelo ID do dispositivo em ordem crescente
df_device_counts = df_device_counts.sort_values(by='device_aid')

# Visualiza o novo DataFrame ordenado
print(df_device_counts)
```

```
          main_hex_id                            device_aid  device_count
259  89b24370433ffff  000a4eb8-9719-6089-1b27-c2a2873f09c9             3
117  89b243700cfffff  00394180-6bbc-6d6e-bcab-0cfe7d2b0460             1
434  89b24371513ffff  006c07bd-abde-6aab-a710-06994e100a3a            41
455  89b24372157ffff  00b7f373-e891-6318-17f9-12e80b3f0c0f             2
179  89b2437042fffff  010103dd-744b-6007-1980-c71cba2a8fda             2
..                ...                                   ...           ...
27   89b24309a6bffff  fe9b0b08-7ab7-6269-bcfd-c9e13fc0ada8             7
340  89b2437045bffff  fefa31e2-32f4-67fa-1271-6437380c11a4             1
116  89b24370087ffff  ff39a042-1e90-6e12-a6ea-8c64993bddb6             2
389  89b243704afffff  ff68a4e0-fd88-6664-2309-ddd0662b49d2             2
454  89b24371513ffff  ff8b81a0-073b-60b8-a244-9483defe0f7b             1

[586 rows x 3 columns]
```

In [105…

```python
# Segunda pergunta ------------
# ¿Cuántos dispositivos han transitado por 2 o más hexágonos? ¿Cuál es la canibalizaci

# Agrupa o DataFrame por "device_aid" e conta quantos hexágonos únicos cada dispositiv
device_transits = df_device_counts.groupby('device_aid').size()

# Filtra os dispositivos que transitaram por 2 ou mais hexágonos
devices_transited_multiple_hexagons = device_transits[device_transits >= 2]

# Conta quantos dispositivos transitaram por 2 ou mais hexágonos
total_devices_transited_multiple_hexagons = len(devices_transited_multiple_hexagons)

print("Número de dispositivos que transitaram por 2 ou mais hexágonos:", total_devices
```

```
Número de dispositivos que transitaram por 2 ou mais hexágonos: 64
```

In [110…

```python
# Segunda pergunta ------------
# ¿Cuántos dispositivos han transitado por 2 o más hexágonos? ¿Cuál es la canibalizaci

# Contar el número total de dispositivos únicos que estuvieron en 2 o más hexágonos
total_devices_transited_multiple_hexagons = len(devices_transited_multiple_hexagons)
```

```python
# Contar el número total de dispositivos únicos en cada hexágono
total_devices_per_hexagon = df_device_counts.groupby('main_hex_id')['device_aid'].nuni

# Calcular la tasa de canibalización para cada hexágono
cannibalization_rate = total_devices_transited_multiple_hexagons / total_devices_per_h

# Visualizar el DataFrame con las tasas de canibalización
print(cannibalization_rate)
```

```
main_hex_id
89b24309a6bffff     2.285714
89b24309b4fffff     4.000000
89b24309b57ffff     3.555556
89b24370087ffff     1.163636
89b243700cfffff     6.400000
89b243700d7ffff     3.764706
89b2437016bffff     3.200000
89b24370203ffff     4.266667
89b2437042fffff     0.800000
89b24370433ffff     1.254902
89b2437045bffff     2.064516
89b243704afffff     1.306122
89b243705dbffff     2.782609
89b24370c87ffff     3.047619
89b24371513ffff     3.047619
89b24372157ffff    21.333333
89b2437232bffff     2.461538
89b24372867ffff     5.333333
89b24373053ffff    10.666667
89b24373417ffff     1.729730
89b243735c7ffff     3.047619
89b2437366bffff     2.461538
Name: device_aid, dtype: float64
```

In [ ]: