

Estudio sobre distribución de los servicios en Barcelona

Trabajo final de máster

Estudiante: Manuel Taberner Llorca

Tutor: Daniel Everett Rhoades

Introducción

Librerías

Importación e inicialización de las librerías que vamos a utilizar a lo largo del desarrollo y estudio de los datos

```
In [ ]: import pandas as pd
import numpy as np
import geopandas as gpd
import geojson
import json
import random
import re
import base64
import folium
import branca.colormap as cm
import h3
from tabulate import tabulate
from folium.plugins import HeatMap, DualMap, FastMarkerCluster
from matplotlib import pyplot as plt
from pandas.plotting import table
```

Carga de dataset con la geometría de los barrios

Se utiliza los datos de este dataset **Unitats AdministrativesBCN GeoJSON**([link](#)) que contiene información geoespacial sobre los barrios de Barcelona y su geometría en forma de *polygon*

Se cambia el nombre de la columna que nos interesa y se transforma el dataset manteniendo solo las columnas que son necesarias o de interés relativas a los barrios.

```
In [ ]: df_barrios = gpd.read_file('barrios_polygons.json')

# Filtramos solo los polygon que sean barrios
df_barrios = df_barrios[df_barrios.TIPUS_UA=='BARRI'].reset_index()
df_barrios = df_barrios[['NOM','Shape_Leng','Shape_Area','geometry']]
df_barrios.crs
df_barrios = df_barrios.to_crs(epsg=4326)
```

```
print(df_barrios.crs)
df_barrios.head()
```

epsg:4326

Out[]:

	NOM	Shape_Leng	Shape_Area	geometry
0	el Raval	5521.646549	1.100286e+06	POLYGON ((2.16471 41.38593, 2.16401 41.38540, ...
1	el Barri Gòtic	5197.999887	8.155939e+05	POLYGON ((2.17701 41.38525, 2.17658 41.38558, ...
2	la Barceloneta	13853.129525	1.179382e+06	POLYGON ((2.19623 41.38745, 2.19617 41.38746, ...
3	Sant Pere, Santa Caterina i la Ribera	4664.482949	1.109669e+06	POLYGON ((2.18345 41.39061, 2.18238 41.39142, ...
4	el Fort Pienc	4137.328784	9.293558e+05	POLYGON ((2.18353 41.39227, 2.18388 41.39253, ...

Lectura inicial CSV y creacion del dataframe

Para comenzar cargamos los datos de **2019_CensComercialBcn_Detall.csv** que se pueden encontrar en este [link](#) a un dataframe.

Tambien observamos la informacion de los datos y una muestra de como son los datos. Se puede observar la descripcion de los campos en la misma web donde se encuentra el dataset en el apartado "*veure descripcio complementaria y definicio de camps*".

In []:

```
df_raw = pd.read_csv("2019_censcomercialbcn_detall.csv", low_memory=False)
df_raw.info()
df_raw.head()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 80554 entries, 0 to 80553
Data columns (total 49 columns):
 #   Column           Non-Null Count Dtype  
 --- 
 0   ID_Bcn_2019      80554 non-null  int64  
 1   ID_Bcn_2016      70529 non-null  float64 
 2   Codi_Principal_Activitat 80554 non-null  int64  
 3   Nom_Principal_Activitat 80554 non-null  object  
 4   Codi_Sector_Activitat   80554 non-null  int64  
 5   Nom_Sector_Activitat   80554 non-null  object  
 6   Codi_Grup_Activitat    80554 non-null  int64  
 7   Nom_Grup_Activitat    80554 non-null  object  
 8   Codi_Activitat_2019    80554 non-null  int64  
 9   Nom_Activitat        80554 non-null  object  
 10  Codi_Activitat_2016    80554 non-null  object  
 11  Nom_Local            80554 non-null  object  
 12  SN_Oci_Nocturn       80554 non-null  int64  
 13  SN_Coworking         80554 non-null  int64  
 14  SN_Servei_Degustacio 80554 non-null  int64  
 15  SN_Obert24h          80554 non-null  int64  
 16  SN_Mixtura           80554 non-null  int64  
 17  SN_Carrer             80554 non-null  int64  
 18  SN_Mercat             80554 non-null  int64  
 19  Nom_Mercat            2187 non-null   object  
 20  SN_Galeria            80554 non-null  int64  
 21  Nom_Galeria           429 non-null   object  
 22  SN_CComercial         80554 non-null  int64  
 23  Nom_CComercial        896 non-null   object  
 24  SN_Eix                80554 non-null  int64  
 25  Nom_Eix               23722 non-null  object  
 26  X_UTM_ETRS89          80554 non-null  float64 
 27  Y_UTM_ETRS89          80554 non-null  float64 
 28  Latitud                80554 non-null  float64 
 29  Longitud               80554 non-null  float64 
 30  Direccio_Unica        80554 non-null  object  
 31  Codi_Via               80554 non-null  int64  
 32  Nom_Via                80554 non-null  object  
 33  Planta                 80554 non-null  object  
 34  Porta                  77337 non-null  float64 
 35  Num_Policia_Inicial   80554 non-null  object  
 36  Lletra_Inicial         2556 non-null   object  
 37  Num_Policia_Final      80546 non-null   object  
 38  Lletra_Final            2535 non-null   object  
 39  Solar                  80452 non-null  float64 
 40  Codi_Parcela           80460 non-null  float64 
 41  Codi_Illa              80460 non-null  float64 
 42  Seccio_Censal          80549 non-null  float64 
 43  Codi_Barri             80554 non-null  int64  
 44  Nom_Barri              80554 non-null  object  
 45  Codi_Districte         80554 non-null  int64  
 46  Nom_Districte          80554 non-null  object  
 47  Referencia_cadastral  80463 non-null   object  
 48  Data_Revisio           80485 non-null  float64 

dtypes: float64(11), int64(18), object(20)
memory usage: 30.1+ MB

```

	ID_Bcn_2019	ID_Bcn_2016	Codi_Principal_Activitat	Nom_Principal_Activitat	Codi_Sector_Activitat
0	1075454	NaN	1	Actiu	
1	1075453	NaN	1	Actiu	
2	1075451	NaN	1	Actiu	
3	1075449	NaN	1	Actiu	
4	1075448	NaN	1	Actiu	

5 rows × 49 columns

Observamos que el dataset esta formado por 80554 filas y 49 columnas.

Funciones de utilidad

Funciones que se utilizan en repetidas ocasiones a lo largo del proyecto para crear visualizaciones u de los datos.

```
In [ ]: # Funcion crear grafica de barras

# Funcion para crear Pie Chart
def pie_chart(sizes,labels):
    ig, ax = plt.subplots(figsize=(15,15))
    ax.pie(sizes, labels=labels, autopct='%0.1f%%')
    ax.axis('equal') # Equal aspect ratio ensures the pie chart is circular.
    plt.show()

# Funcion para mostrar datos de las localizaciones de un dataframe en un mapa
def map_dots(df):
    m= folium.Map(location=[df.Latitud.mean(),df.Longitud.mean()],zoom_start=12.45)
    for _, row in df.iterrows():
        folium.CircleMarker(
            location=[row.Latitud,row.Longitud],
            radius=1,
            popup=re.sub(r'[^a-zA-Z ]+', ' ', row.Nom_Local),
            color='blue',
            fill=True,
            fill_color='#1787FE'
        ).add_to(m)
    for _, r in df_barrios.iterrows():
        sim_geo = gpd.GeoSeries(r['geometry']).simplify(tolerance=0.001)
        geo_j = sim_geo.to_json()
        geo_j = folium.GeoJson(geo_j,style_function = lambda x: {
            'color': 'orange',
            'weight': 2,
            "opacity":1,
            'fillOpacity': 0,
```

```

        })
        folium.Popup(r[ 'NOM' ]).add_to(geo_j)
        geo_j.add_to(m)
    return m

def map_heatmap(df):
    m= folium.Map(location=[df.Latitud.mean(),df.Longitud.mean()],zoom_start=12.45)
    HeatMap(list(zip(df.Latitud,df.Longitud)),radius=15).add_to(m)
    for _, r in df_barrios.iterrows():
        sim_geo = gpd.GeoSeries(r['geometry']).simplify(tolerance=0.001)
        geo_j = sim_geo.to_json()
        geo_j = folium.GeoJson(geo_j,style_function = lambda x: {
            'color': 'orange',
            'weight': 2,
            "opacity":1,
            'fillOpacity': 0,
        })
        folium.Popup(r[ 'NOM' ]).add_to(geo_j)
        geo_j.add_to(m)
    return m

def heat_dots(df):
    m = folium.plugins.DualMap(location=[df.Latitud.mean(),df.Longitud.mean()], titl

    fg_1 = folium.FeatureGroup(name='Puntos').add_to(m.m1)
    fg_2 = folium.FeatureGroup(name='HeatMap').add_to(m.m2)
    fg_both = folium.FeatureGroup(name='Barrios').add_to(m)

    icon_red = folium.Icon(color='red')
    for _, row in df.iterrows():
        folium.CircleMarker(
            location=[row.Latitud,row.Longitud],
            radius=1,
            popup= re.sub(r'^[a-zA-Z ]+', ' ', row.Nom_Local),
            color='blue',
            fill=True,
            fill_color='#1787FE'
        ).add_to(fg_1)
    HeatMap(list(zip(df.Latitud,df.Longitud)),radius=15).add_to(fg_2)
    for _, r in df_barrios.iterrows():
        sim_geo = gpd.GeoSeries(r['geometry']).simplify(tolerance=0.001)
        geo_j = sim_geo.to_json()
        geo_j = folium.GeoJson(geo_j,style_function = lambda x: {
            'color': 'orange',
            'weight': 1.5,
            "opacity":0.7,
            'fillOpacity': 0,
        })
        folium.Popup(r[ 'NOM' ]).add_to(geo_j)
        geo_j.add_to(fg_both)

    folium.LayerControl(collapsed=True).add_to(m)

    return m

# Funcion para mostrar una tabla con todos los datos de un DF
def df_table(df):
    fig, ax = plt.subplots()
    ax.axis('off')
    ax.axis('tight')
    t= ax.table(cellText=df.values, colLabels=df.columns, loc='center')
    t.auto_set_font_size(False)

```

```

t.auto_set_column_width(list(range(len(df.columns))))
t.set_fontsize(8)
plt.show()

# Funcion para mostrar un histograma en dos dimensiones
def heat_hist(df):
    plt.figure(figsize = (15,12))
    plt.hist2d(df.Longitud, df.Latitud ,bins=80, cmap='hot')
    plt.colorbar().set_label('Número de servicios')
    plt.xlabel('Longitud', fontsize=14)
    plt.ylabel('Latitud', fontsize=14)
    plt.title('Distribución de servicios en Barcelona', fontsize=17)
    plt.show()

```

Transformacion de los datos

Primero de todo vamos a filtrar los datos segun la columna **Nom Principal Activitat** que indica si un servicio esta activo o no, solo nos interesan los servicios activos. Antes de filtrar el DF contiene **80554** servicios.

```
In [ ]: df = df_raw[df_raw.Nom_Principal_Activitat == 'Actiu']
print("Número de servicios del DF filtrado: ",len(df.index))
```

Número de servicios del DF filtrado: 61558

El siguiente paso es seleccionar las columnas que puede que nos resulten de interes. Las cuales son:

- **ID_Bcn_2019**: identificador unico de cada local 2019
- **Latitud**: coordenada de latitud
- **Longitud**: coordenada de longitud
- **Nom_Sector_Activitat**: nombre del sector del servicio
- **Nom_Grup_Activitat**: nombre del grupo al que pertenece del servicio
- **Nom_Activitat**: nombre del tipo de servicio
- **Nom_Barri**: nombre del barrio en la ciudad de Barcelona
- **Nom_Districte**: nombre del distrito en la ciudad de Barcelona

```
In [ ]: # Filtrado de las columnas que pueden resultar de interés
df_final = df[['ID_Bcn_2019','Latitud','Longitud','X_UTM_ETRS89','Y_UTM_ETRS89','Nom_Sector_Activitat','Nom_Grup_Activitat','Nom_Activitat','Nom_Barri','Nom_Districte']]
df_final.head()
```

	ID_Bcn_2019	Latitud	Longitud	X_UTM_ETRS89	Y_UTM_ETRS89	Nom_Sector_Activitat	Nom
0	1075454	41.346101	2.130166	427229.272	4577543.637	Serveis	
1	1075453	41.345939	2.129560	427178.393	4577526.160	Serveis	
2	1075451	41.345591	2.128543	427092.921	4577488.381	Serveis	
3	1075449	41.346262	2.130599	427265.676	4577561.147	Altres	
4	1075448	41.346514	2.131271	427322.177	4577588.560	Serveis	

A continuacion comprobamos que no existen valores nulos en nuestro DF.

```
In [ ]: df_final.isna().values.any()
```

```
Out[ ]: False
```

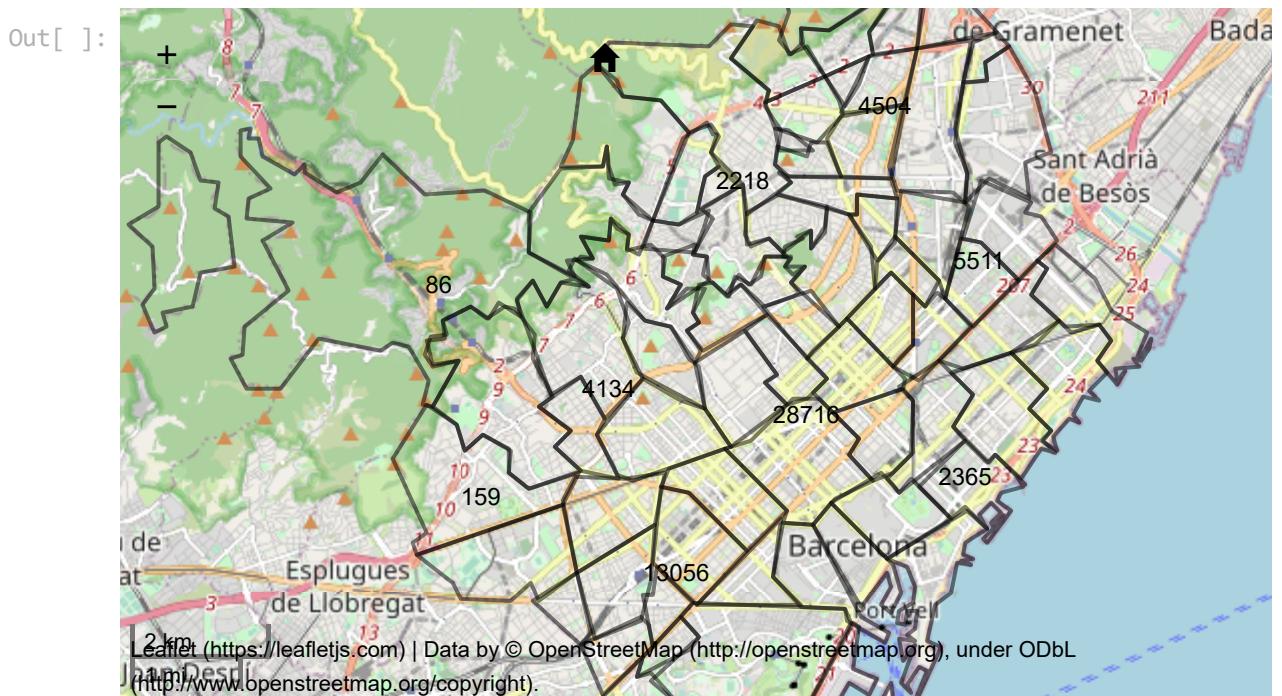
Analisis de datos

Observamos la distribución de todos los servicios en un mapa interactivo de la ciudad de Barcelona para obtener una idea simple de como estan distribuidos.

Al finalizar esta presentación de los datos se continua con un análisis de las diferentes variables que hemos seleccionado para crear nuestro dataset y eliminar alguna si es necesario.

```
In [ ]: def barrios_borders(map,border_color):
    for _, r in df_barrios.iterrows():
        sim_geo = gpd.GeoSeries(r['geometry']).simplify(tolerance=0.001)
        geo_j = sim_geo.to_json()
        geo_j = folium.GeoJson(geo_j,style_function = lambda x: {
            'color': border_color,
            'weight': 2,
            "opacity":0.5,
            'fillOpacity': 0,
        })
        folium.Popup(r['NOM']).add_to(geo_j)
        geo_j.add_to(map)
    return map

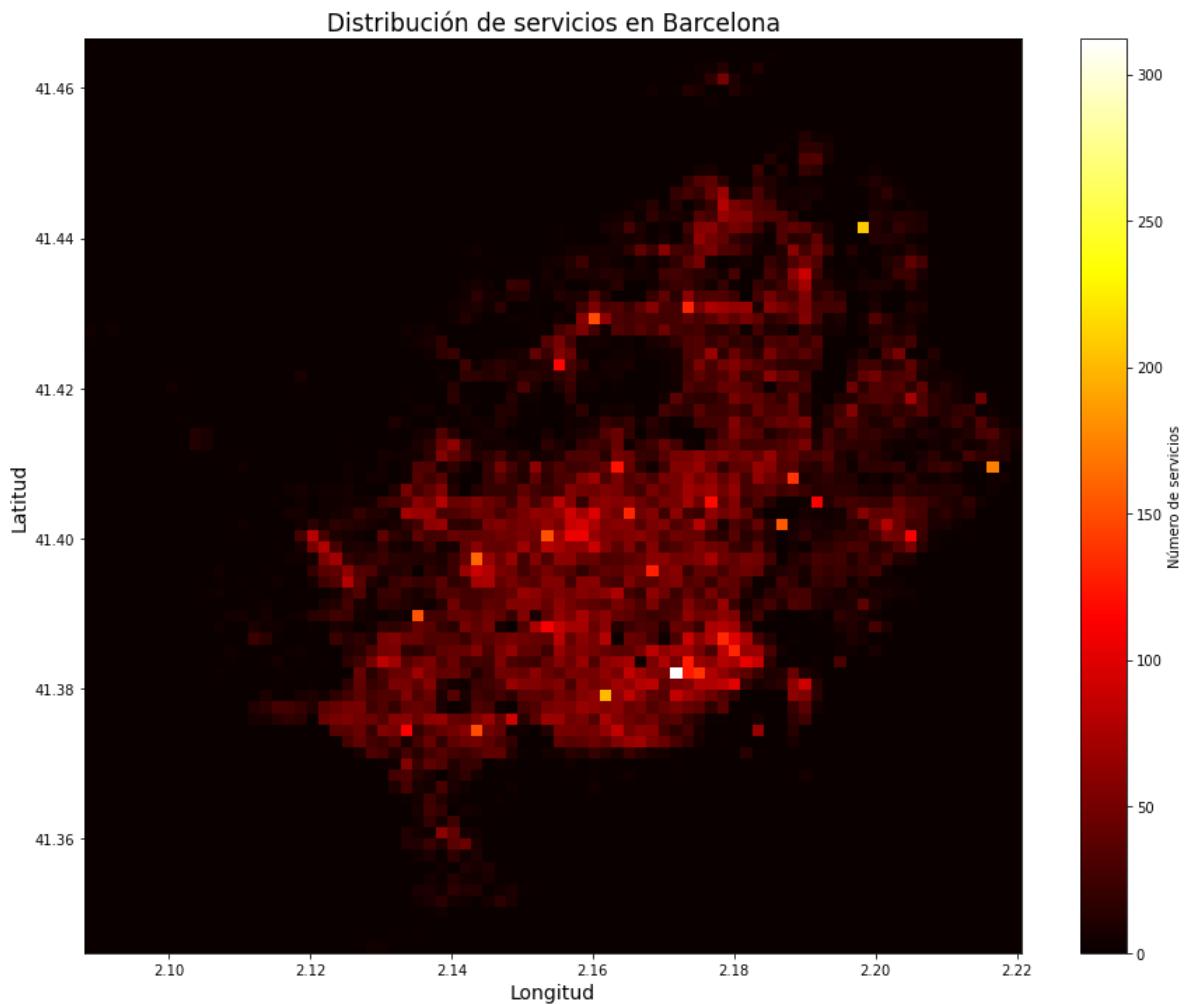
map = folium.Map(location=[df_final.Latitud.mean(), df_final.Longitud.mean()], zoom_start=12)
map.add_child(FastMarkerCluster(df_final[['Latitud','Longitud']].values.tolist()))
barrios_borders(map,'black')
map
```



Hist2d de los servicios

Permite intuir cuales son las zonas con mas servicios y donde hay una mayor acumulación.

In []: `heat_hist(df_final)`



Hexagons heatmap

La creación de este mapa permite ver por zonas hexagonales y el color de estas cual es la diferencia en cantidad de servicios entre una zonas u otras.

```
In [ ]: def counts_by_hexagon(df, resolution):
    """
        Uso de h3.geo_to_h3 para indexar cada punto en el índice espacial de la resolución
        Uso de h3.h3_to_geo_boundary para obtener las geometrías de los hexágonos
    """
    df = df[["Latitud", "Longitud"]].copy()

    df["hex_id"] = df.apply(lambda row: h3.geo_to_h3(row["Latitud"], row["Longitud"], resolution))

    df_aggreg = df.groupby(by = "hex_id").size().reset_index()
    df_aggreg.columns = ["hex_id", "value"]

    df_aggreg["geometry"] = df_aggreg.hex_id.apply(lambda x:
        {
            "type" : "Polygon",
            "coordinates": [
                h3.h3_to_geo_boundary(x, resolution)
            ]
        }
    )

    return df_aggreg

def hexagons_dataframe_to_geojson(df_hex, file_output = None):
    """
        Crea el GeoJSON desde un DataFrame que tiene la columna en formato geojson
        y además las columnas hex_id y value
    """
    list_features = []

    for i, row in df_hex.iterrows():
        feature = geojson.Feature(geometry = row["geometry"], id=row["hex_id"], properties = {"value": row["value"]})
        list_features.append(feature)

    feat_collection = geojson.FeatureCollection(list_features)

    geojson_result = json.dumps(feat_collection)

    return geojson_result

def choropleth_map(df_aggreg, border_color = 'black', fill_opacity = 1, initial_map = None, kind = "greentored"):
    """
        Crea un mapa choropleth mediante los datos agregados utilizando la función choropleth_map
    """
    #colormap
    min_value = df_aggreg["value"].min()
    max_value = df_aggreg["value"].max()
    m = round ((min_value + max_value) / 2, 0)

    #coger la resolution de la primera fila
    res = h3.h3_get_resolution(df_aggreg.loc[0, 'hex_id'])

    if initial_map is None:
        initial_map = folium.Map(location= [df_final.Latitud.mean(), df_final.Longitud.mean()],
                                attr= '@ <a href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>')
    else:
        initial_map = folium.Map(location= [df_final.Latitud.mean(), df_final.Longitud.mean()],
                                attr= '@ <a href="http://www.openstreetmap.org/copyright">OpenStreetMap</a>')

    #selección de color
    if kind == "greentored":
        colors = ["#d9ead3", "#c8e6c9", "#b2df8a", "#82e0AA", "#4DB6AC", "#31a354"]
    else:
        colors = ["#fbb4ae", "#fdd0a2", "#fdd0a2", "#fdd0a2", "#fdd0a2", "#fdd0a2"]

    for index, row in df_aggreg.iterrows():
        folium.Choropleth(
            name="Choropleth Map",
            geo_data=df_hex,
            data=df_aggreg,
            columns=["hex_id", "value"],
            key_on="hex_id",
            fill_color=colors[int((row['value'] - min_value) / (max_value - min_value) * len(colors))],
            fill_opacity=fill_opacity,
            line_color=border_color,
            line_weight=1
        ).add_to(initial_map)

    return initial_map
```

```

        custom_cm = cm.LinearColormap(['white','red','black'], vmin=min_value, vmax=max_value)
    elif kind == "bwr":
        #for outliers, values would be -11,0,1
        custom_cm = cm.LinearColormap(['blue','white','red'], vmin=min_value, vmax=max_value)
    elif kind == "sgyl":
        custom_cm = cm.LinearColormap(['sienna','green','yellow','red'],
                                      index=[0,min_value,m,max_value],vmin=min_value,vmax=max_value)

#creacion de datos geojson desde el dataframe
geojson_data = hexagons_dataframe_to_geojson(df_hex = df_aggreg)

#añade plot en el mapa
name_layer = "Choropleth " + str(res)

folium.GeoJson(
    geojson_data,
    style_function=lambda feature: {
        'fillColor': custom_cm(feature['properties']['value']),
        'color': border_color,
        'weight': 1,
        'fillOpacity': fill_opacity
    },
    name = name_layer
).add_to(initial_map)

#Leyenda
if with_legend == True:
    custom_cm.add_to(initial_map)

return initial_map

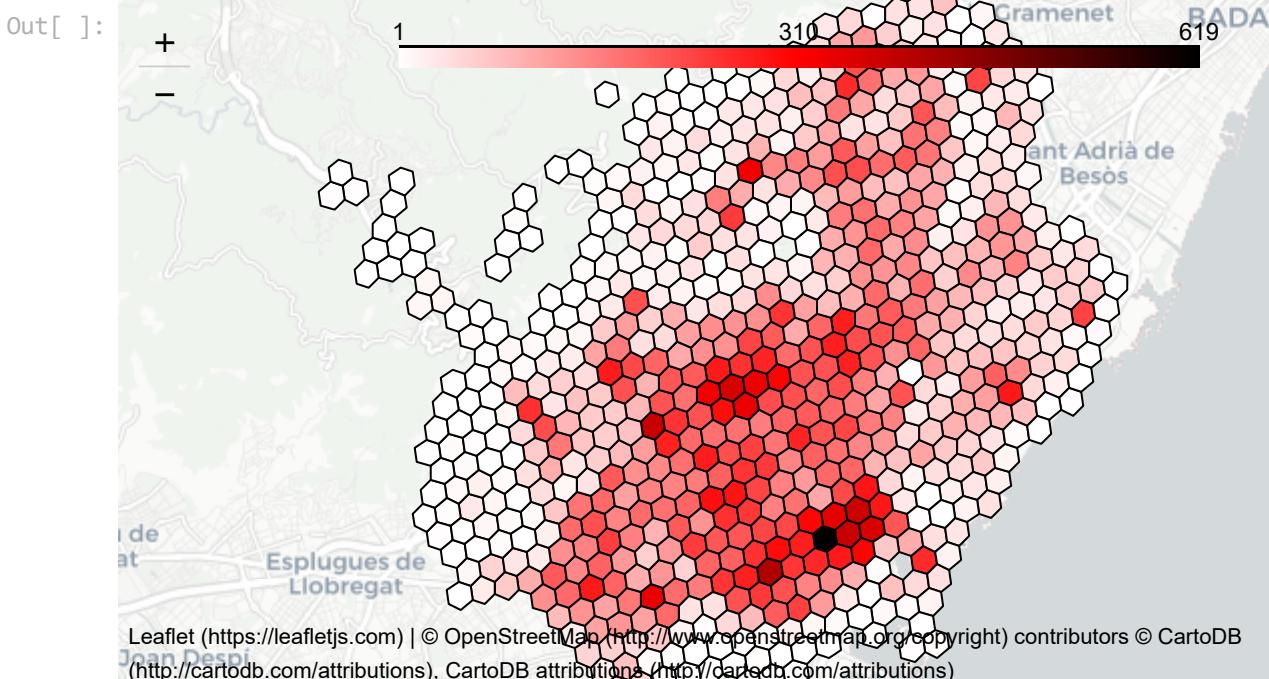
```

In []:

```

# Contador de puntos dentro del hexagono
df_aggreg = counts_by_hexagon(df_final, resolution = 9)
# Ordenar los valores de forma descendente
df_aggreg.sort_values(by = "value", ascending = False, inplace = True)
# Creacion del mapa usando folium
hexmap = choropleth_map(df_aggreg, with_legend = True)
hexmap

```



Sectores de actividad

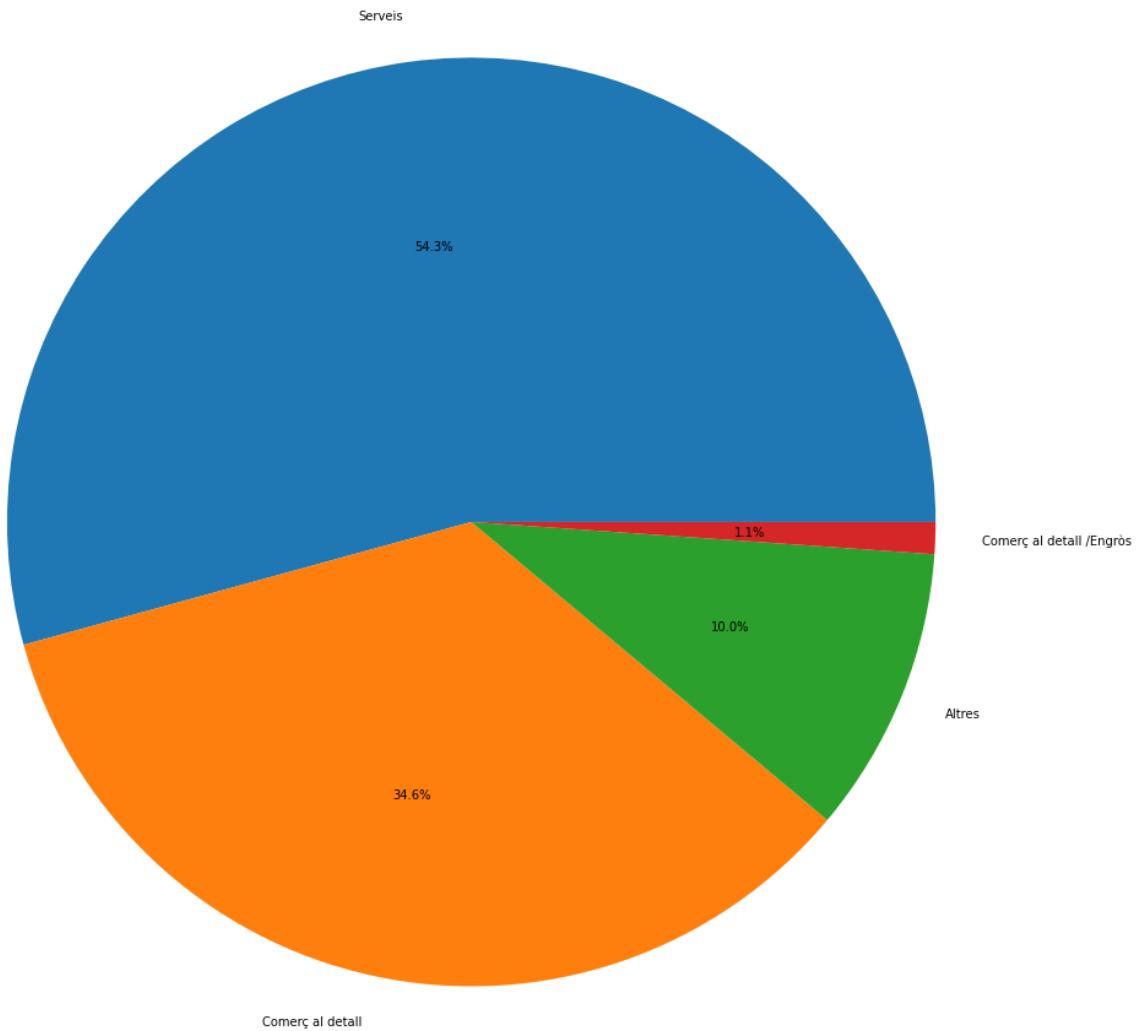
Mediante un grafico de barras podemos observar los diferentes tipos de sectores que hay y cuantos serv y de esta forma

```
In [ ]: df_sectors_counts = df_final['Nom_Sector_Activitat'].value_counts().rename_axis('Sector').reset_index(name='Cantidad')
```

```
Out[ ]:
```

	Sectores	Cantidad
0	Serveis	33402
1	Comerç al detall	21326
2	Altres	6148
3	Comerç al detall /Engròs	682

```
In [ ]: pie_chart(df_sectors_counts.Cantidad,df_sectors_counts.Sectores)
```



En la gráfica de arriba podemos observar que el sector con mayor cantidad de servicios es Serveis.

A continuación mostramos todos los valores distintos y cuantos hay de cada uno segun Sector-Grupo-Actividad. Ahora podemos estudiar si es posible eliminar alguno sector, grupo o actividad más.

Observamos que de todos los tipos de Actividad los unicos que no nos interesan soon los que pone "Altres" por lo tanto procedemos a eliminar esos servicios.

```
In [ ]: df_distinct_count = df_final.value_counts(['Nom_Sector_Activitat','Nom_Grup_Activitat'])
tabla_counts = df_table(df_distinct_count)
```

Sectores	Grupos	Actividad	Cantidad
Serveis	Restaurants, bars i hotels (Inclòs hostals, pensions i fordes)	Restaurants	5599
Serveis	Altres	Serveis a les empreses i oficines	3464
Serveis	Restaurants, bars i hotels (Inclòs hostals, pensions i fordes)	Bars / CIBERCAFE	3290
Comerç al detall	Equipament personal	Vestir	3157
Serveis	Altres	Perruqueries	2659
Comerç al detall	Quotidiana alimentari	Autoservei / Supermercat	2337
Serveis	Ensenyament	Ensenyament	2115
Serveis	Sanitat i assistència	Sanitat i assistència	2024
Altres	Altres	Activitats de la construcció	1965
Comerç al detall	Quotidiana alimentari	Pa, pastisseria i làctics	1562
Serveis	Altres	Centres d'estètica	1499
Serveis	Altres	Activitats emmagatzematge	1442
Serveis	Reparacions (Electrodomèstics i automòbils)	Reparacions (Electrodomèstics i automòbils)	1405
Serveis	Finances i assegurances	Finances i assegurances	1242
Comerç al detall	Quotidiana no alimentari	Farmàcies PARAFARMÀCIA	1086
Comerç al detall	Quotidiana alimentari	Carn i Porc	1069
Serveis	Equipaments culturals i recreatius	Equipaments culturals i recreatius	1037
Comerç al detall	Quotidiana alimentari	Fuites i verdures	1021
Comerç al detall	Parament de la llar	Material equipament llar	966
Altres	Altres	Associacions	952
Serveis	Activitats immobiliàries	Activitats immobiliàries	950
Serveis	Restaurants, bars i hotels (Inclòs hostals, pensions i fordes)	Serveis de menjar take away MENJAR RÀPID	929
Comerç al detall	Equipament personal	Calçat i pell	808
Serveis	Restaurants, bars i hotels (Inclòs hostals, pensions i fordes)	serveis d'allotjament	740
Comerç al detall	Equipament personal	joieria, rellotgeria i bijuteria	731
Altres	Altres	Activitats industrials	712
Serveis	Altres	Arranjaments	692
Comerç al detall	Oci i cultura	Informàtica	691
Comerç al detall /Engròs	Altres	Altres	682
Comerç al detall	Quotidiana alimentari	Altres	682
Comerç al detall	Quotidiana no alimentari	Drogueria i perfumeria	680
Altres	Altres	Arts gràfiques	642
Comerç al detall	Parament de la llar	Mobles i articles fusta i metall	556
Comerç al detall	Altres	Basars	554
Comerç al detall	Quotidiana alimentari	Peix i marisc	499
Serveis	Altres	Pàrquings i garatges	494
Comerç al detall	Oci i cultura	Llibres, diaris i revistes	453
Serveis	Altres	Serveis Socials	450
Serveis	Altres	Activitats de transport	448
Serveis	Altres	Tintoreries	432
Comerç al detall	Oci i cultura	Juguetes i esports	419
Comerç al detall	Quotidiana no alimentari	Tabac i articles fumadors	417
Altres	Altres	Equipaments religiosos	393
Serveis	Altres	Veterinaris / Mascotes	392
Comerç al detall	Automoció	Vehicles	386
Altres	Altres	Altres	385
Altres	Altres	Administració	361
Comerç al detall	Altres	Souvenirs	345
Comerç al detall	Quotidiana no alimentari	Herbolàrics, dietètica i NUTRICIÓ	337
Comerç al detall	Altres	Optiques	328
Serveis	Altres	Altres	324
Serveis	Altres	Agències de viatge	324
Serveis	Manteniment, neteja i producció	Manteniment, neteja i similars	321
Serveis	Restaurants, bars i hotels (Inclòs hostals, pensions i fordes)	Bars especials amb actuació / Bars musicals / Discoteques /PUB	315
Comerç al detall	Parament de la llar	Parament ferreteria	309
Altres	Altres	Gimnàs/fitness	292
Comerç al detall	Quotidiana alimentari	Ous i aus	280
Comerç al detall	Quotidiana alimentari	Begudes	254
Serveis	Altres	Serveis de telecomunicacions	233
Comerç al detall	Parament de la llar	Roristeries	229
Comerç al detall	Equipament personal	Merceria	222
Altres	Altres	Altres equipaments esportius	221
Comerç al detall	Parament de la llar	Segells, monedes i antiguitats	192
Comerç al detall	Parament de la llar	Aparells domèstics	169
Serveis	Restaurants, bars i hotels (Inclòs hostals, pensions i fordes)	Xocolateries / Geladeries / Degustació	164
Serveis	Altres	Locutoris	157
Altres	Altres	Fabricació tèxtil	150
Comerç al detall	Quotidiana alimentari	Plats preparats (no degustació)	146
Serveis	Altres	Activitats de transport i emmagatzematge	141
Comerç al detall	Altres	Fotografia	134
Serveis	Restaurants, bars i hotels (Inclòs hostals, pensions i fordes)	serveis de menjar i begudes	88
Comerç al detall	Oci i cultura	Música	81
Altres	Altres	Esports	75
Comerç al detall	Quotidiana no alimentari	Combustibles i carburants	72
Comerç al detall	Altres	Maquinària	67
Comerç al detall	Altres	Souvenirs i basars	40
Comerç al detall	Altres	Optiques i fotografia	27
Comerç al detall	Altres	Grans magatzems i hipermercats	20
Serveis	Restaurants, bars i hotels (Inclòs hostals, pensions i fordes)	Altres (per exemple VENDING)	16
Serveis	Restaurants, bars i hotels (Inclòs hostals, pensions i fordes)	altres	16

```
In [ ]: print("Número de servicios del DF: ",len(df_final.index))
df_final = df_final[df_final.Nom_Activitat != 'Altres']
df_final = df_final[df_final.Nom_Activitat != 'Serveis a les empreses i oficines']
df_final = df_final[df_final.Nom_Activitat != 'Activitats de la construcció']
df_final = df_final[df_final.Nom_Activitat != 'Activitats emmagatzematge']
df_final = df_final[df_final.Nom_Activitat != 'Associacions']
df_final = df_final[df_final.Nom_Activitat != 'Activitats industrials']
df_final = df_final[df_final.Nom_Activitat != 'altres']
```

```
df_final = df_final[df_final.Nom_Activitat != 'Altres ( per exemple VENDING)']
print("Número de servicios del DF: ", len(df_final.index))
```

Número de servicios del DF: 61558
Número de servicios del DF filtrado: 50918

Grupos de actividad

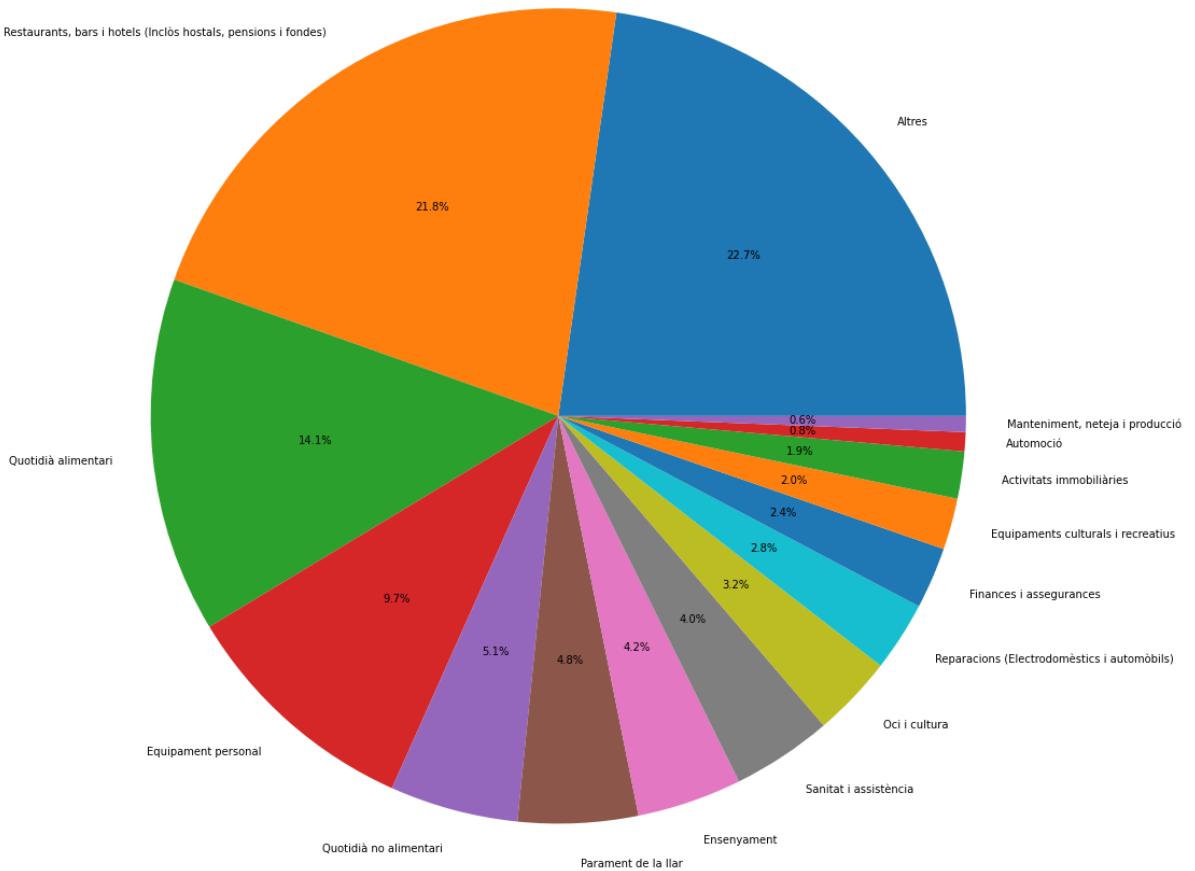
En este apartado estudiamos los diferentes grupos de actividad que existen para obtener una visión de cuales son los más grandes.

```
In [ ]: df_grups_counts = df_final['Nom_Grup_Activitat'].value_counts().rename_axis('Grupos')
df_grups_counts
```

Out[]:

	Grupos	Cantidad
0	Altres	11570
1	Restaurants, bars i hotels (Inclòs hostals, pe...	11125
2	Quotidià alimentari	7168
3	Equipament personal	4918
4	Quotidià no alimentari	2592
5	Parament de la llar	2421
6	Ensenyament	2115
7	Sanitat i assistència	2024
8	Oci i cultura	1644
9	Reparacions (Electrodomèstics i automòbils)	1405
10	Finances i assegurances	1242
11	Equipaments culturals i recreatius	1037
12	Activitats immobiliàries	950
13	Automoció	386
14	Manteniment, neteja i producció	321

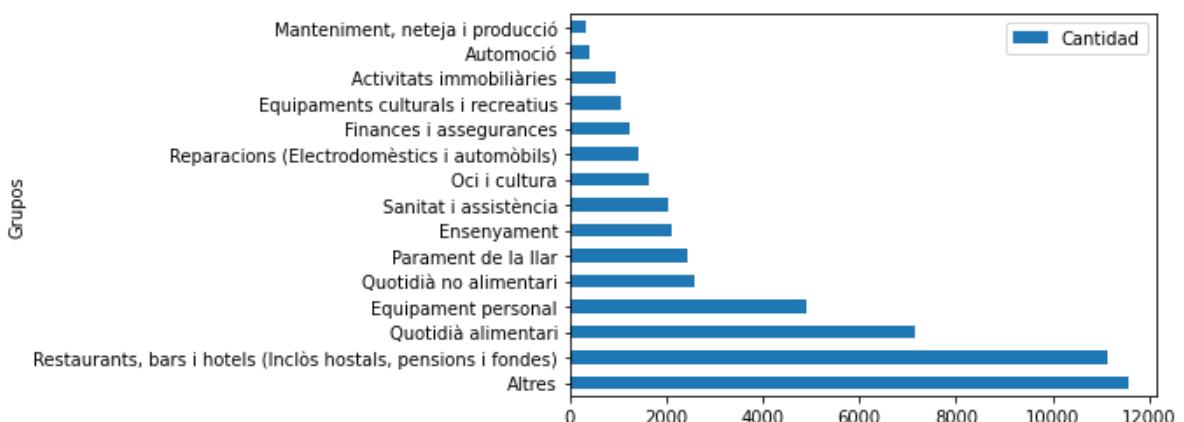
```
In [ ]: pie_chart(df_grups_counts.Cantidad,df_grups_counts.Grupos)
```



Como podemos observar en la grafica anterior y en el diagrama de bajo hay varios grupos que predominan sobre los anteriores, en el grupo mas grande *Altres* se encuentran muchos tipos de actividad.

```
In [ ]: df_grups_counts.plot(kind="barh", x='Grupos')
```

```
Out[ ]: <AxesSubplot:ylabel='Grupos'>
```



Actividad

La siguiente tabla permite observar todos los tipos de actividad que hay y cuantos servicios hay de cada uno.

```
In [ ]: df_activitat_counts = df_final['Nom_Activitat'].value_counts().rename_axis('Activitat')
total_activitat = df_activitat_counts['Cantidad'].sum()
total_activitat
```

Out[]: 50918

In []: # Crea una tabla de todos los tipos de actividad y la cantidad

```
tabla = df_table(df_activitat_counts)

# Crea un archivo .txt con el contenido de la tabla para utilizar como recurso de
with open('table_less.txt', 'w', encoding='utf-8') as f:
    dfAsString = dfAsString.encode("utf-8")
    f.write(tabulate(df_activitat_counts, ["Actividad", "Cantidad"], tablefmt="grid"))
```

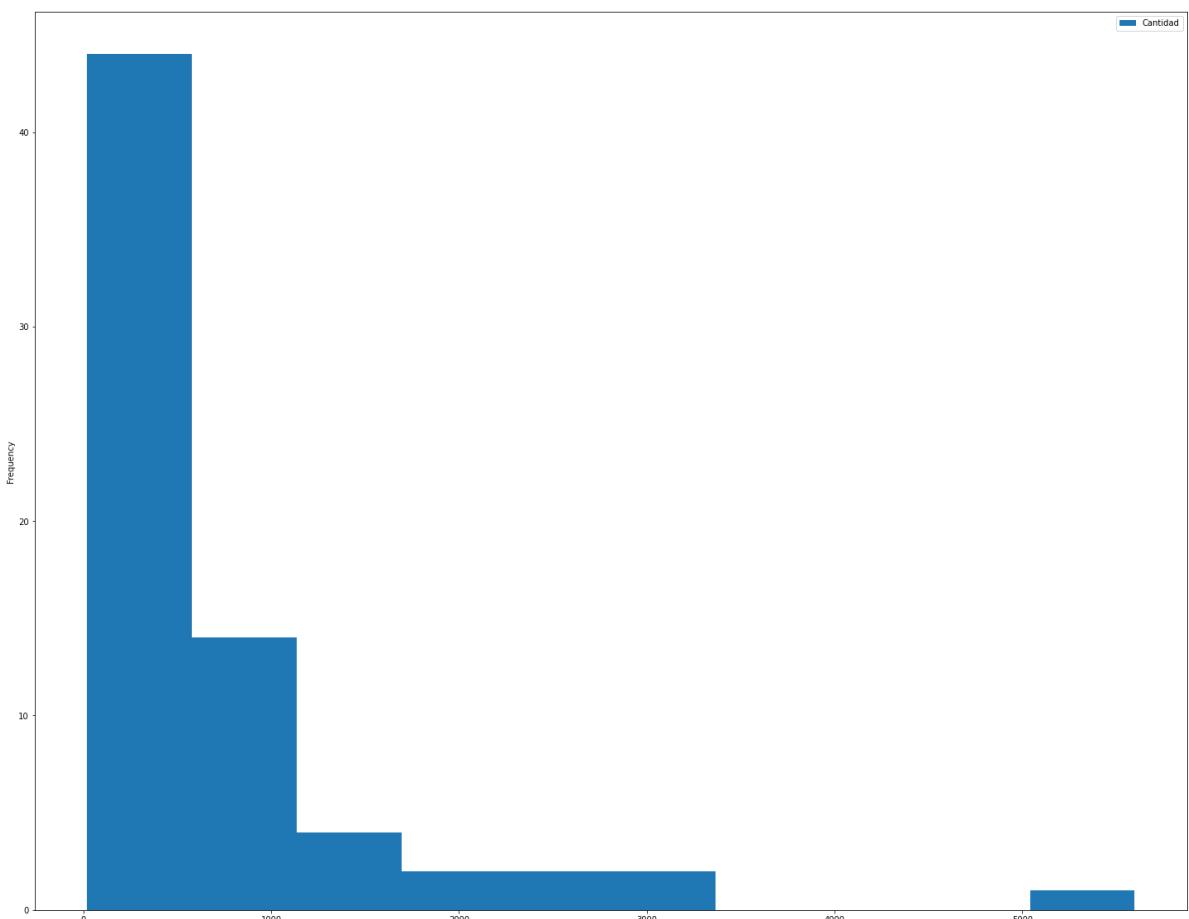
Actividad	Cantidad
Restaurants	5599
Bars / CIBERCAFE	3290
Vestir	3157
Perruqueries	2659
Autoservei / Supermercat	2337
Ensenyament	2115
Sanitat i assistència	2024
Pa, pastisseria i làctics	1562
Centres d'estètica	1499
Reparacions (Electrodomèstics i automòbils)	1405
Finances i assegurances	1242
Farmàcies PARAFARMACIA	1086
Carr i Porc	1069
Equipaments culturals i recreatius	1037
Fruites i verdures	1021
Material equipament llar	966
Activitats immobiliàries	950
Serveis de menjar take away MENJAR RAPID	929
Calçat i pell	808
serveis d'allotjament	740
Joieria, rellotgeria i bijuteria	731
Aranjaments	692
Informàtica	691
Drogueria i perfumeria	680
Arts gràfiques	642
Mobles i articles fusta i metall	556
Basars	554
Peix i marisc	499
Parquings i garatges	494
Llibres, diaris i revistes	453
Serveis Socials	450
Activitats de transport	448
Tintoreries	432
Juguetes i esports	419
Tabac i articles fumadors	417
Equipaments religiosos	393
Veterinaris / Mascotes	392
Vehicles	386
Administració	361
Souvenirs	345
Herboraries, dietètica i NUTRICIÓ	337
Optiques	328
Agències de viatge	324
Manteniment, neteja i similars	321
Bars especials amb actuació / Bars musicals / Discoteques /PUB	315
Parament ferreteria	309
Gimnàs /fitnes	292
Ous i aus	280
Begudes	254
Serveis de telecomunicacions	233
Floristeries	229
Merceria	222
Altres equipaments esportius	221
Segells, monedes i antiguitats	192
Aparells domèstics	169
Xocolateries / Geladeries / Degustació	164
Locutoris	157
Fabricació tèxtil	150
Plats preparats (no degustació)	146
Activitats de transport i emmagatzematge	141
Fotografia	134
serveis de menjar i begudes	88
Música	81
Esports	75
Combustibles i carburants	72
Maquinària	67
Souvenirs i basars	40
Optiques i fotografia	27
Grans magatzems i hipermercats	20

Observamos que el tipo de actividad predominante es la restauración con mucha diferencia de las demás ya que encontramos *Restaurants* como el tipo de actividad mas repetida y *Bar* y *Cafe* como el tercer tipo.

En el histograma que podemos ver a continuación, se aprecia como los tipos de actividades hay muchos pero con cantidades mas pequeñas. además observamos lo mismo que en la gráfica de más arriba la mayoría de los tipos de actividad tienen menos de 1000 servicios.

```
In [ ]: df_activitat_counts.plot(kind="hist", x='Actividad', figsize= (25,20))
```

```
Out[ ]: <AxesSubplot:ylabel='Frequency'>
```



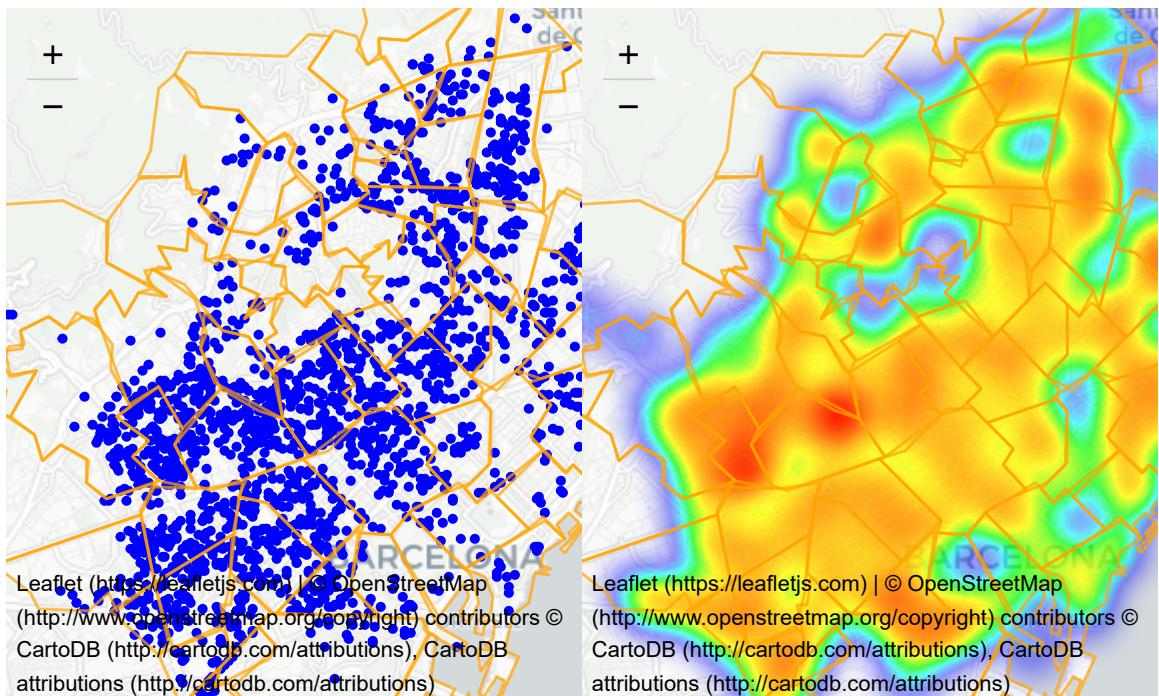
Actividades de interés

A continuación se visualizan gráficas de puntos y heatmaps de tipos de acitividades que se han visto.

- **Sanidad y asistencia**

```
In [ ]: df_sanitat_assitencia = df_final[df_final.Nom_Activitat == 'Sanitat i assistència']
heat_dots(df_sanitat_assitencia)
```

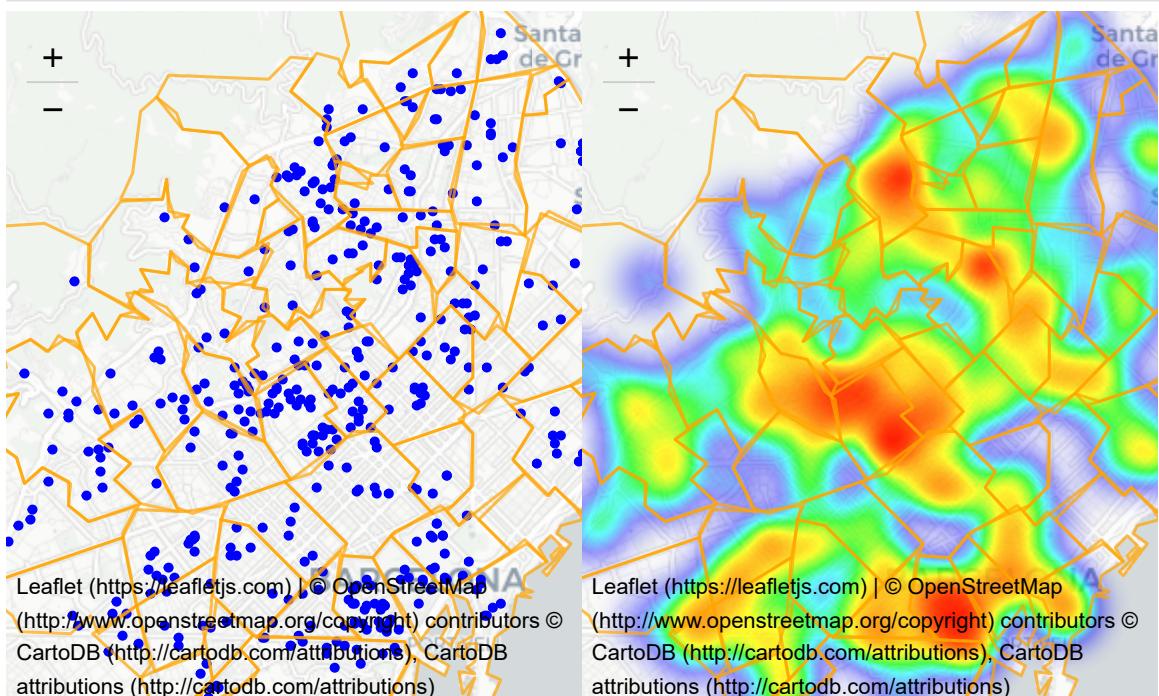
Out[]:



- **Servicios sociales**

In []: `df_serveis_socials = df_final[df_final.Nom_Activitat == 'Serveis Socials']
heat_dots(df_serveis_socials)`

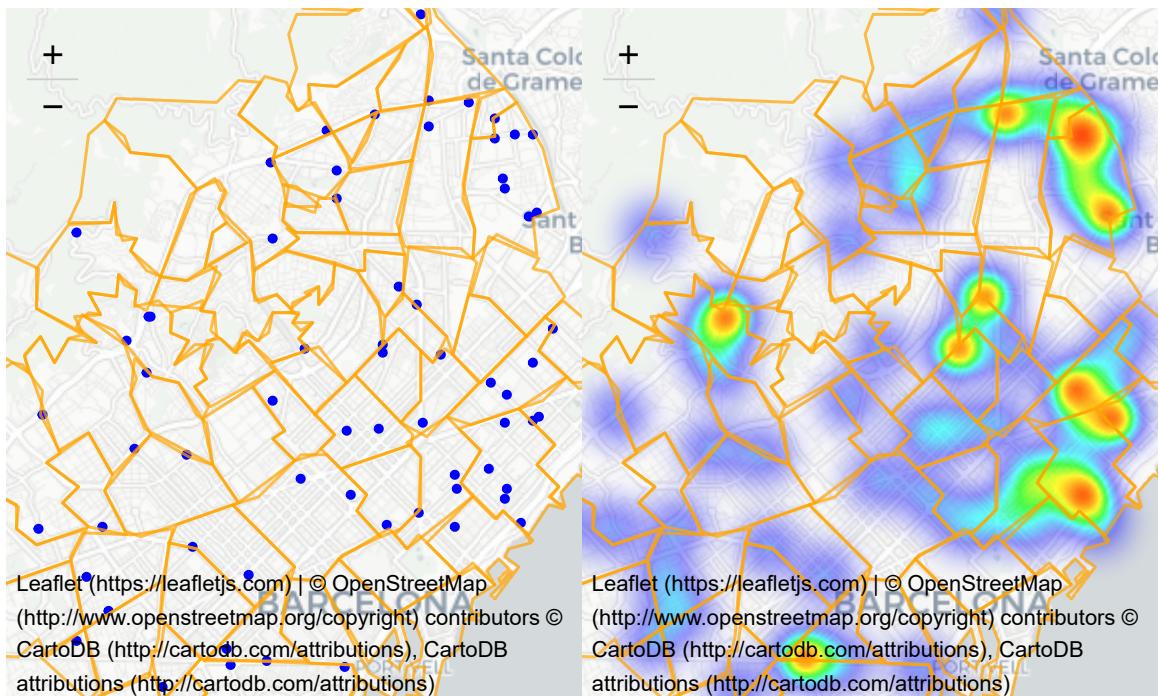
Out[]:



- **Combustibles y carburantes**

In []: `df_gasolina = df_final[df_final.Nom_Activitat == 'Combustibles i carburants']
heat_dots(df_gasolina)`

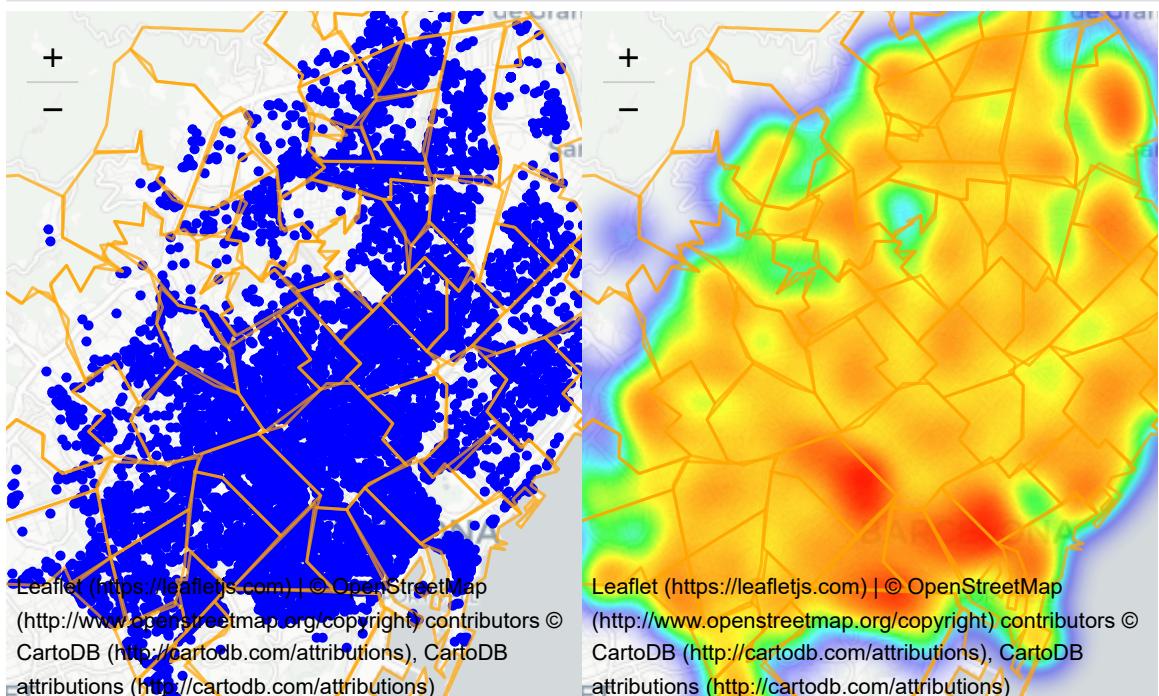
Out[]:



- Restauració

In []: `df_rest = df_final[(df_final.Nom_Activitat == 'Restaurants') | (df_final.Nom_Activitat == 'Cafeteria')]`
`heat_dots(df_rest)`

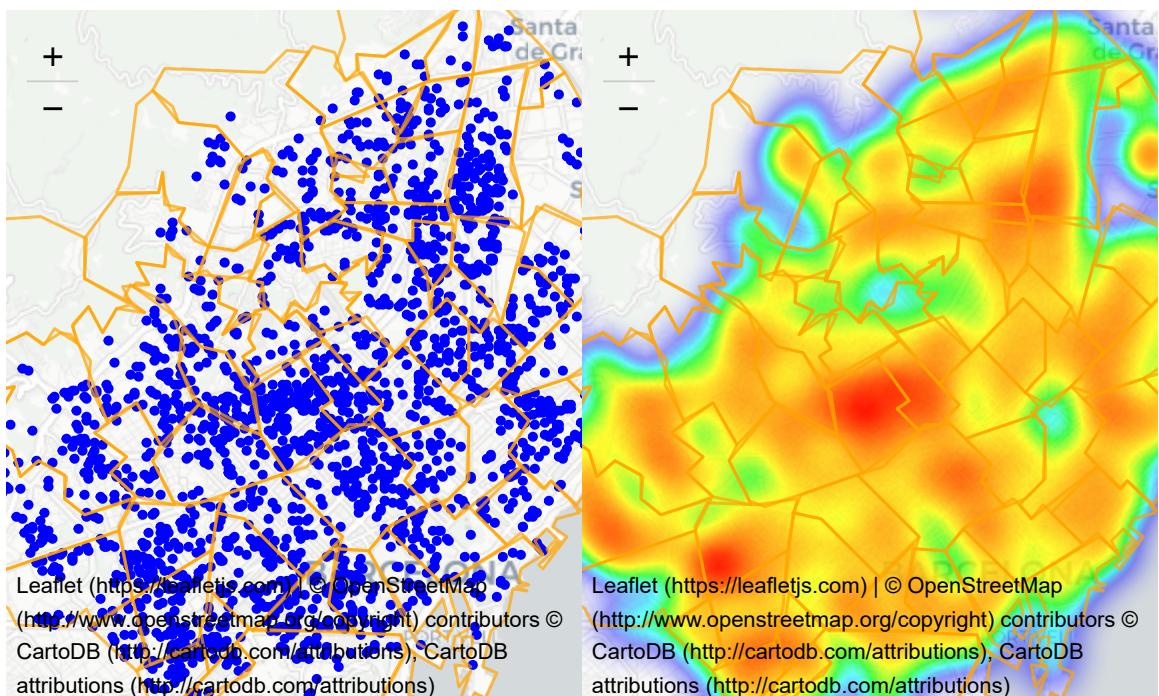
Out[]:



- Enseñanza

In []: `df_ensena = df_final[df_final.Nom_Activitat == 'Ensenyament']`
`heat_dots(df_ensena)`

Out[]:

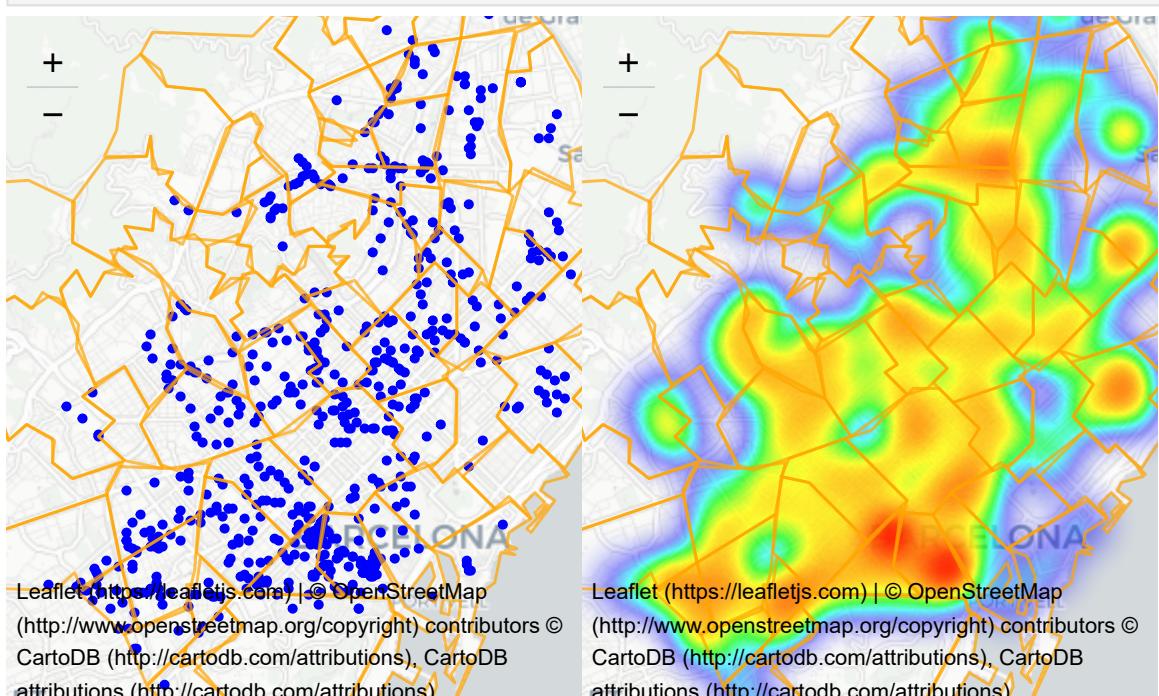


- **Informàtica**

In []:

```
df_infor = df_final[df_final.Nom_Activitat == 'Informàtica']
heat_dots(df_infor)
```

Out[]:

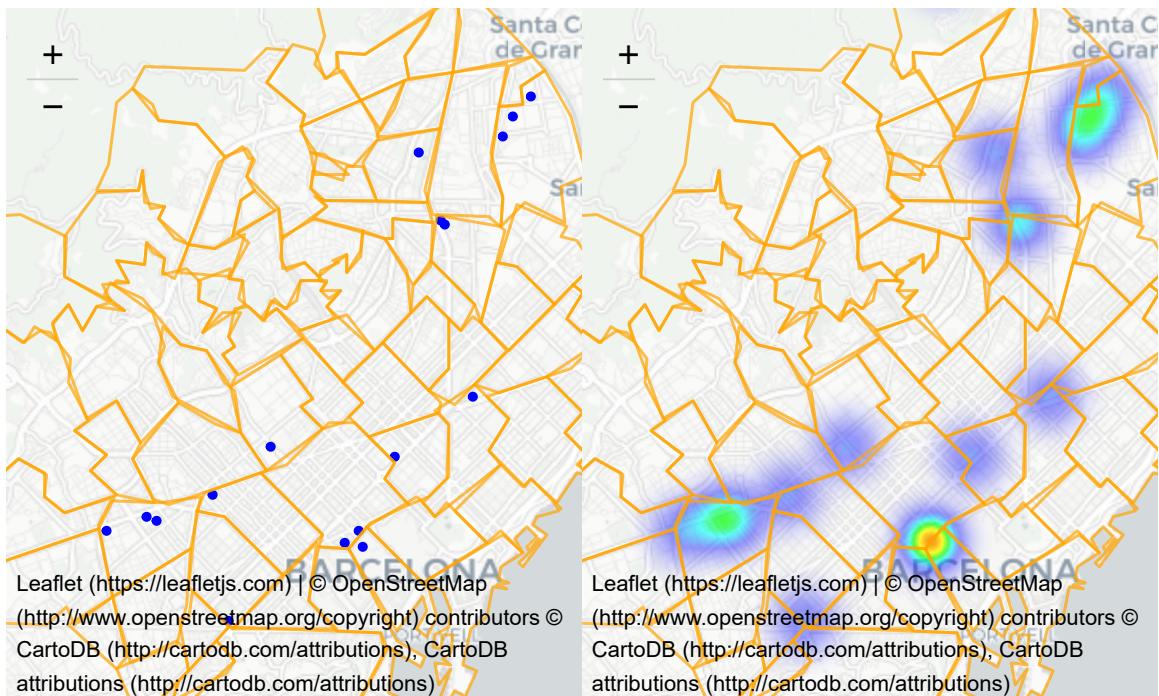


- **Grandes almacenes**

In []:

```
df_grans = df_final[df_final.Nom_Activitat == 'Grans magatzems i hipermercats']
heat_dots(df_grans)
```

Out[]:



Algoritmo DBSCAN

In []:

```
'''  
    Implementación del algoritmo DBSCAN  
    Se utiliza las coordenadas de Longitud y Latitud del sistema ETRS89  
'''  
  
# Comprueba si el punto es CENTRO  
def check_core_point(eps,minPts, df, index):  
    # coge los puntos del indice  
    x, y = df.iloc[index]['X_UTM_ETRS89'] , df.iloc[index]['Y_UTM_ETRS89']  
  
    #chequea los puntos disponibles dentro del radio  
    temp = df[((np.abs(x - df['X_UTM_ETRS89']) <= eps) & (np.abs(y - df['Y_UTM_ETRS89']) <= eps))]  
  
    #chequea cuantos puntos hay disponibles dentro del radio  
    if len(temp) >= minPts:  
        #devuelve cuando el punto es core  
        return (temp.index , True, False, False)  
  
    elif (len(temp) < minPts) and len(temp) > 0:  
        #devuelve cuando el punto es frontera  
        return (temp.index , False, True, False)  
  
    elif len(temp) == 0:  
        #devuelve cuando el punto es ruido  
        return (temp.index , False, False, True)  
  
# Realiza la clusterizacion  
def cluster_with_stack(eps, minPts, df):  
  
    #iniciando el numero de cluster  
    C = 1  
    #inicializando la stacks a mantener  
    current_stack = set()  
    unvisited = list(df.index)  
    clusters = []  
  
    while (len(unvisited) != 0): #continua hasta que todos los puntos se visiten
```

```

#identifica el primer punto de un cluster
first_point = True

#elige un punto aleatorio no visitado
current_stack.add(random.choice(unvisited))

while len(current_stack) != 0: #continua hasta que el cluster este completo

    #pop el punto actual del stack
    curr_idx = current_stack.pop()

    #chequea si el punto es core, vecino o frontera
    neigh_indexes, iscore, isborder, isnoise = check_core_point(eps, minPts)

    #lidiando con un caso extremo
    if (isborder & first_point):
        #para el primero punto de frontera se marca como ruido y tambien su
        clusters.append((curr_idx, 0))
        clusters.extend(list(zip(neigh_indexes,[0 for _ in range(len(neigh_
        #se marca como visitado
        unvisited.remove(curr_idx)
        unvisited = [e for e in unvisited if e not in neigh_indexes]

        continue

unvisited.remove(curr_idx) #quitar punto de lista no visitada

neigh_indexes = set(neigh_indexes) & set(unvisited) #mirar solo puntos

if iscore: #si el punto actual es un core
    first_point = False

    clusters.append((curr_idx,C)) #asignar a un cluster
    current_stack.update(neigh_indexes) #añadir vecinos a la stack

elif isborder: #si es un punto frontera
    clusters.append((curr_idx,C))

    continue

elif isnoise: #si es ruido pertenece al cluster 0
    clusters.append((curr_idx, 0))

    continue

if not first_point:
    #aumentar el numero de clusters
    C+=1

return clusters

```

In []:

```

...
Funcion para realizar los clusters partiendo de un DF que contenga las columnas
Resultado: df_merge que contiene una columna indicando a que cluster pertenece

df:      dataframe a clusterizar
eps:     radio de vecindad de un punto en METROS
minPts: numero minimo de vecinos que tiene que haber dentro del radio eps para
...
def DBSCANwithDF(df,eps,minPts):

```

```

df.reset_index(drop=True, inplace=True)
data = pd.DataFrame(df, columns = ["X_UTM_ETRS89", "Y_UTM_ETRS89"] )

clustered = cluster_with_stack(eps, minPts, data)

idx , cluster = list(zip(*clustered))

cluster_df = pd.DataFrame(clustered, columns = ["idx", "cluster"])
cluster_df = cluster_df.set_index('idx')
cluster_df = cluster_df.sort_index()

df_merge = pd.merge(df, cluster_df, left_index=True, right_index=True)
df_merge = df_merge.drop_duplicates()
df_merge = df_merge[df_merge.cluster != 0]
df_merge

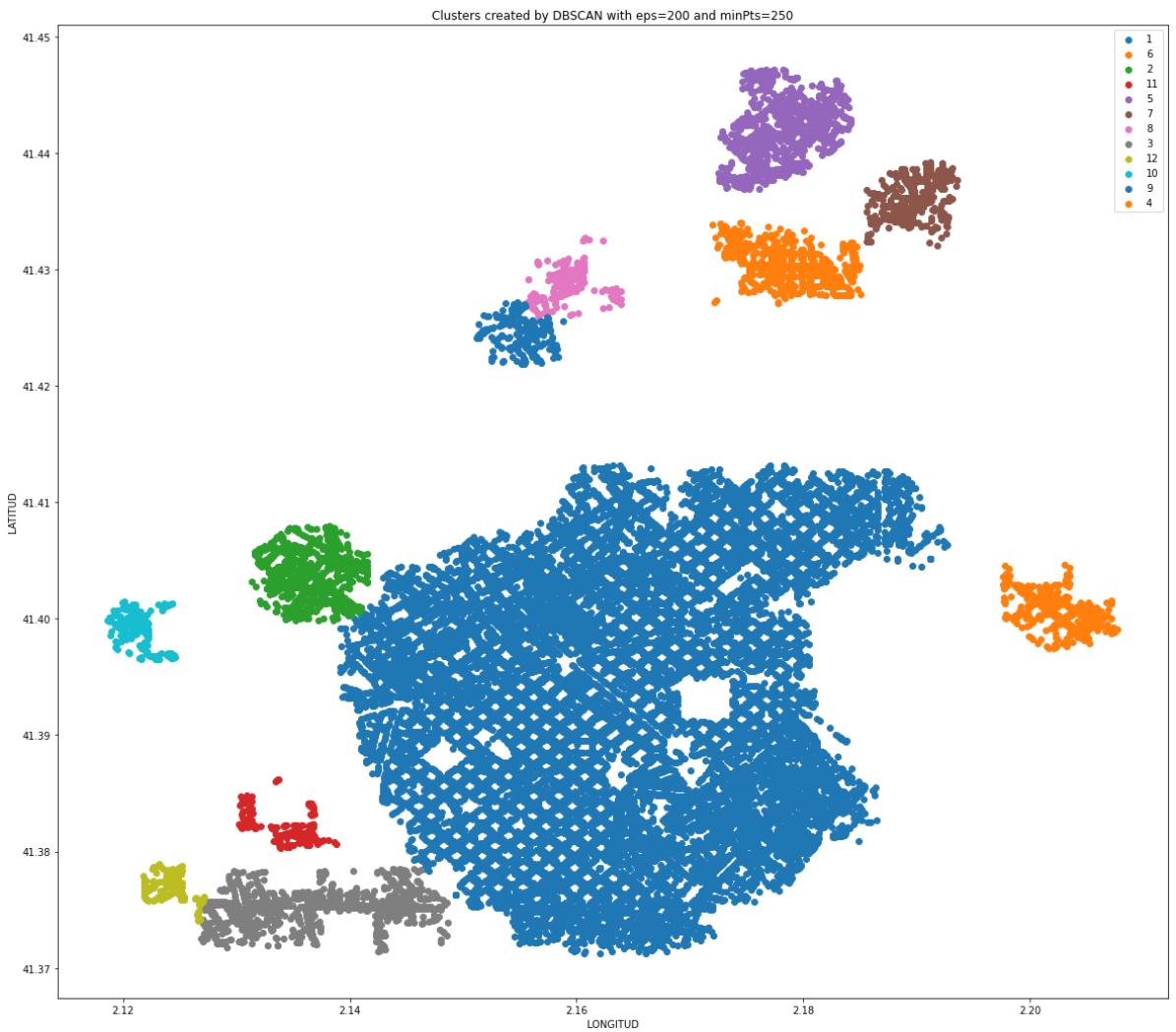
return df_merge

...
    Scatter PLOT con los clusters que ha generado el algoritmo DBSCAN
...
def scatterDBSCAN(df,eps,minPts):
    plt.figure(figsize=(20,18))
    for clu in df.cluster.unique():
        plt.scatter(df[df.cluster == clu].Longitud, df[df.cluster == clu].Latitud,
                    color='red' if clu == 0 else 'blue')

    plt.title('Clusters created by DBSCAN with eps='+str(eps)+ ' and minPts=' +str(m))
    plt.xlabel('LONGITUD')
    plt.ylabel('LATITUD')
    plt.legend()
    return plt.show()

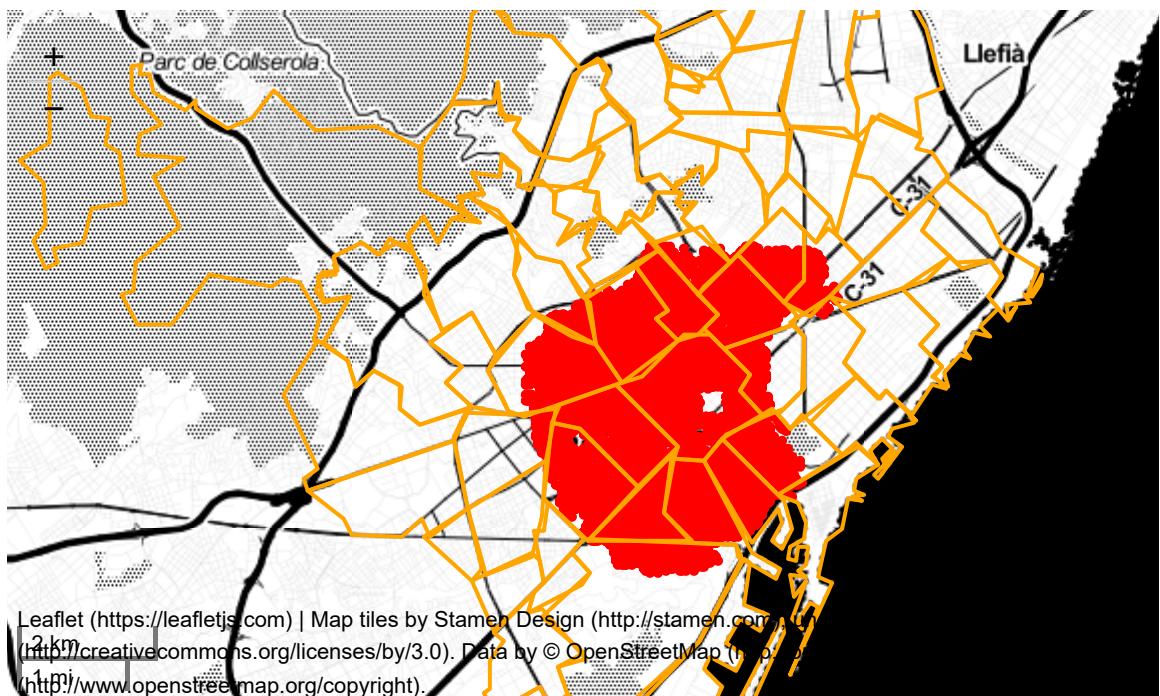
```

In []: df_clusterized = DBSCANwithDF(df_final,200,250)
scatterDBSCAN(df_clusterized,200,250)



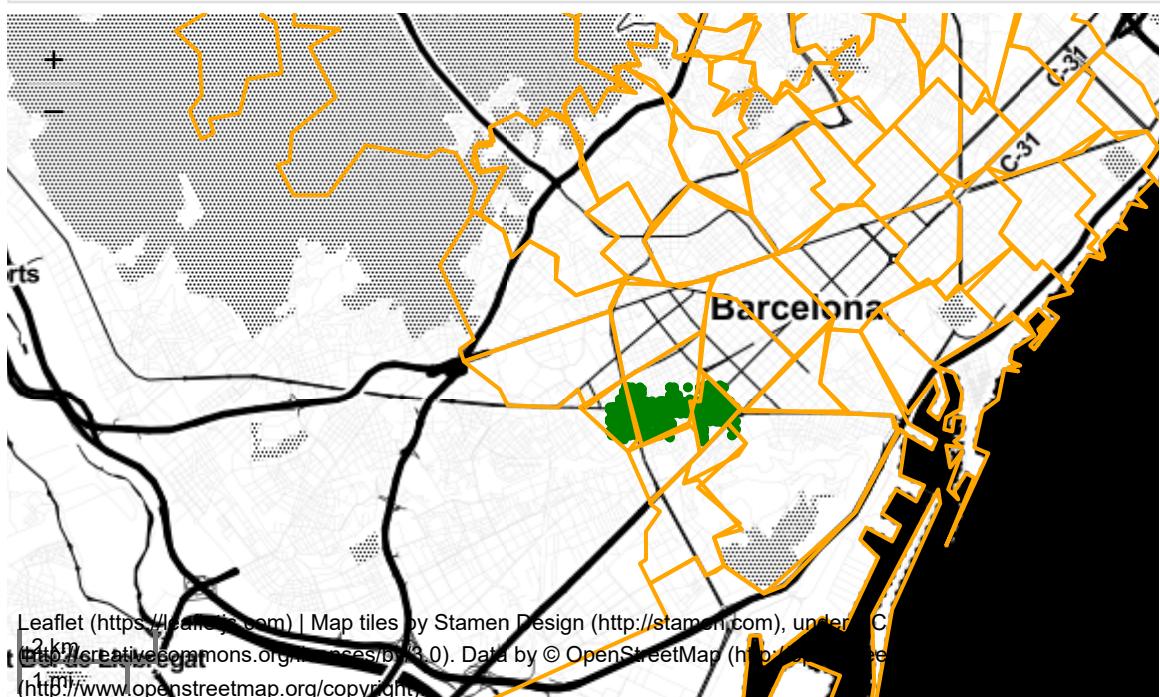
```
In [ ]: def map_clusters(df):
    m= folium.Map(location=[df.Latitud.mean(),df.Longitud.mean()],zoom_start=12.45,
    colors=['red','blue','green','yellow','orange','purple','cyan','darkblue','pink',
    for _, row in df.iterrows():
        folium.CircleMarker(
            location=[row.Latitud,row.Longitud],
            radius=1,
            popup= row.Nom_Local,
            color=colors[row.cluster-1],
            fill=True,
            fill_color=colors[row.cluster-1]
        ).add_to(m)
    for _, r in df_barrios.iterrows():
        sim_geo = gpd.GeoSeries(r['geometry']).simplify(tolerance=0.001)
        geo_j = sim_geo.to_json()
        geo_j = folium.GeoJson(geo_j,style_function = lambda x: {
            'color': 'orange',
            'weight': 2,
            "opacity":1,
            'fillOpacity': 0,
        })
        folium.Popup(r['NOM']).add_to(geo_j)
        geo_j.add_to(m)
    return m
map_clusters(df_clusterized[df_clusterized.cluster ==1])
```

```
Out[ ]:
```



```
In [ ]: map_clusters(df_clusterized[df_clusterized.cluster ==3])
```

```
Out[ ]:
```



```
In [ ]: """
```

```
SE HA USADO PARA CREAR DIFERENTES MAPAS Y VISUALIZARLOS PARA OBSERVAR CUAL PUEDE SER EL MEJOR.
```

```
eps_sweep = [75,100,150,200,250,300,350,400,450]  
minPts_sweep = [75,100,150,200,250,300,350,400,450,500]
```

```
for n in eps_sweep:  
    for x in minPts_sweep:  
        # Change to meters  
        #radius of the circle defined as 0.6  
        eps = n  
        #minimum neighbouring points set to 3  
        minPts = x  
        df_final.reset_index(drop=True, inplace=True)  
        data = pd.DataFrame(df_final, columns = ["X_UTM_ETRS89", "Y_UTM_ETRS89"] )
```

```

        clustered = cluster_with_stack(eps, minPts, data)

        idx , cluster = list(zip(*clustered))

        cluster_df = pd.DataFrame(clustered, columns = ["idx", "cluster"])
        cluster_df = cluster_df.set_index('idx')
        cluster_df = cluster_df.sort_index()

        df_merge = pd.merge(df_final, cluster_df, left_index=True, right_index=True)
        df_merge = df_merge.drop_duplicates()

        plt.figure(figsize=(20,18))
        for clu in df_merge.cluster.unique():
            if clu == 0:
                plt.scatter(df_merge[df_merge.cluster == clu].Longitud, df_merge[df_merge.cluster == clu].Latitud, c='grey', alpha=0.5, label=clu)
            else:
                plt.scatter(df_merge[df_merge.cluster == clu].Longitud, df_merge[df_merge.cluster == clu].Latitud, c='red', alpha=0.5, label=clu)
        plt.title('Clusters created by DBSCAN with eps='+str(eps) + ' and minPts=' + str(minPts))
        plt.xlabel('LONGITUD')
        plt.ylabel('LATITUD')
        plt.legend()
        plt.savefig('eps'+str(eps)+minPts'+str(minPts)+'.png')
        plt.close()
    """

```

Out[]:

```

'\nSE HA USADO PARA CREAR DIFERNETES MAPAS Y VISUALIZARLOS PARA OBSERVER CUAL PUED
E SER ADECUADO\n\neps_sweep = [75,100,150,200,250,300,350,400,450]\nminPts_sweep =
[75,100,150,200,250,300,350,400,450,500]\n\nfor n in eps_sweep:\n    for x in mi
nPts_sweep:\n        # Change to meters\n        #radius of the circle defined as
0.6\n        eps = n\n        #minimum neighbouring points set to 3\n        minPt
s = x\n        df_final.reset_index(drop=True, inplace=True)\n        data = pd.Da
taFrame(df_final, columns = ["X_UTM_ETRS89", "Y_UTM_ETRS89"] )\n\n        clustere
d = cluster_with_stack(eps, minPts, data)\n\n        idx , cluster = list(zip(*clu
stered))\n\n        cluster_df = pd.DataFrame(clustered, columns = ["idx", "cluste
r"])\n        cluster_df = cluster_df.set_index('idx')\n        cluster_df = clu
ster_df.sort_index()\n\n        df_merge = pd.merge(df_final, cluster_df, left_inde
x=True, right_index=True)\n        df_merge = df_merge.drop_duplicates()\n\n        plt.figure(figsize=(20,18))\n        for clu in df_merge.cluster.unique():\n            if clu == 0:\n                plt.scatter(df_merge[df_merge.cluster == clu].Longit
ud, df_merge[df_merge.cluster == clu].Latitud, c='grey', alpha=0.5, label=clu)
            else:\n                plt.scatter(df_merge[df_merge.cluster == clu].Longitud,
df_merge[df_merge.cluster == clu].Latitud, label=clu)\n        plt.ti
tle('Clusters created by DBSCAN with eps=' + str(eps) + ' and minPts=' + str(minPt
s))\n        plt.xlabel('LONGITUD')\n        plt.ylabel('LATITUD')\n        pl
t.legend()\n        plt.savefig('eps' + str(eps) + 'minPts' + str(minPts) + '.png')
        plt.close()'

```