Análisis sobre la inmigración en Barcelona por nacionalidad, barrio y edad (2022)

Amy Ramírez Jurado

Índice

1. Contexto y estructura del trabajo

En este trabajo vamos a analizar el número de inmigrantes en el año 2022 en la ciudad de Barcelona por nacionalidad, sexo y grupos de edad quinquenales según el Padrón Municipal de Habitantes, con datos extraídos de la página oficial del gobierno de España. Para ello, utilizaremos MongoDB a través de Jupyter Notebook.

El objetivo de este análisis es identificar las nacionalidades con mayor peso, la distribución por sexo y las diferentes franjas de edad de la población inmigrante dentro de Barcelona, para así poder reconocer tendencias y definir conclusiones sobre la migración del año 2022.

El presente trabajo se va a dividir en tres partes. En primer lugar, tras cargar los datos, aplicaremos el método CRUD, referido a las operaciones básicas que se pueden realizar sobre los datos en una base de datos para realizar ejemplos y poder comprender mejor los siguientes apartados. Una vez vistas las interactuaciones básicas con los datos, pasaremos a analizar de forma más específica los datos obtenidos y añadiremos gráficos y mapas para poder visualizarlos mejor. Por último, se expondrán unas conclusiones basadas en los datos analizados.

Con respecto a la estructura del dataset, se trata de un objeto JSON que representa un registro de datos. Cada registro contiene información sobre un inmigrante en Barcelona para el año 2022, con detalles como el distrito y barrio donde reside, su nacionalidad, edad, sexo, y otros atributos.

Para cada inmigrante se recogen los siguientes datos:

- "**Any**": Año al que pertenece el registro.
- **"Codi_Districte"**: Código numérico del distrito.
- "**Nom_Districte**": Nombre del distrito.
- "**Codi_Barri**": Código numérico del barrio.
- "**Nom_Barri**": Nombre del barrio.
- "**AEB**": Código que indica el tipo de entidad básica (por ejemplo, un municipio).
- "**Seccio_Censal**": Código de la sección censal.
- "**Valor**": Valor asociado al registro (puede contener un valor numérico que represente una cantidad o un valor especial como "..").
- "**Nacionalitat_G** ": Código que representa la nacionalidad del inmigrante (español, perteneciente a la Unión Europea o de fuera de la Unión Europea).
- "**Edat_Q**": Código que representa un rango quinquenal de edad.
- "**Sexe**": Código que representa el sexo del inmigrante (1 para masculino y 2 para femenino).

2. Carga de datos

Primero, importamos las bibliotecas necesarias para realizar el análisis.

```
import pymongo
import requests
import json
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import folium
from folium.plugins import HeatMap
```

Tras descargar el archivo JSON de la página del Gobierno, lo cargamos en una base de datos MongoDB.

CARGA DE DATOS

```
# Cargamos el archivo JSON desde la URL
url = 'https://opendata-ajuntament.barcelona.cat/resources/bcn/EstadisticaPadro/imm/2022/2022_pad_imm_mdbas_sexe_edat-q_nacionalitat-g.json'

# Realizamos una solicitud GET para obtener el contenido del archivo JSON
response = requests.get(url)

# Verificamos el estado de la solicitud antes de intentar procesar los datos
if response.status_code == 200:
    # Cargamos el contenido del archivo JSON en una variable Python
    data = response.json()
    print("El archivo JSON se ha cargado correctamente desde la URL.")
else:
    print("Error al descargar el archivo JSON.")

def main():
    if data:
        # Utilizamos pymongo para establecer la conexión con el servidor de MongoDB en localhost
        try:
            client = pymongo.MongoClient('mongodb://localhost:27017/')
            db = client['barcelona_immigrants']   # Creamos una base de datos
            collection = db['immigration_data']   # Creamos una colección para almacenar los datos del archivo JSON
            print("Conexión establecida con MongoDB.")
            collection.insert_many(data)  # Insertamos los datos en la colección
            print("Datos insertados correctamente en MongoDB.")
        except pymongo.errors.ConnectionFailure as e:
            print(f'Error al conectar con MongoDB: {e}')
    else:
        print("No se pudieron cargar los datos.")

main()
```

```
El archivo JSON se ha cargado correctamente desde la URL.
Conexión establecida con MongoDB.
Datos insertados correctamente en MongoDB.
```

En primer lugar, vamos a utilizar la librería "*request*" para hacer una solicitud GET a la URL que contiene el archivo JSON. Luego, verificamos si la solicitud fue exitosa (código de estado 200) y cargamos los datos del archivo JSON en la variable "*data*".

Para el almacenamiento de datos vamos a definir la función "*main*", encargada de establecer la conexión con MongoDB usando "*pymongo*". Conectamos al cliente de MongoDB en "*localhost*" en el puerto 27017 y creamos nuestra base de datos denominada "*barcelona_inmigrants*" y la colección "*inmigration_data*", para posteriormente insertar los datos en esta última.

3. Método CRUD

El método CRUD representa las siguientes operaciones fundamentales:

- ***Create*** (Crear): Crear nuevos documentos o registros en la base de datos.
- ***Read*** (Leer): Leer o recuperar registros específicos o conjuntos de datos existentes de la base de datos que cumplen ciertos criterios de búsqueda.
- ***Update*** (Actualizar): Actualizar documentos o registros existentes en la base de datos.
- ***Delete*** (Eliminar): Eliminar documentos o registros existentes de la base de datos.

Vamos a realizar algunos ejemplos con cada operación para después poder aplicar el método en el análisis final.

3.1 Creación

Comenzamos creando un nuevo documento en nuestra base de datos de inmigrantes en Barcelona y lo insertamos en la colección como ejemplo:

Creación

```
try:
    client = pymongo.MongoClient('mongodb://localhost:27017/')
    db = client['barcelona_immigrants']   # Base de datos
    collection = db['immigration_data']   # Colección para los datos de inmigración
    print("Conexión establecida con MongoDB.")
except pymongo.errors.ConnectionFailure as e:
    print(f'Error al conectar con MongoDB: {e}')

def insert_data(data, collection):
    if data:
        try:
            # Insertamos los datos en la colección
            collection.insert_many(data)  # Para insertar múltiples documentos en la colección de una sola vez
            print("Datos insertados correctamente.")
        except pymongo.errors.PyMongoError as e:
            print(f'Error al insertar datos en MongoDB: {e}')

# Creamos un nuevo documento
new_document = [
    {
        "Any": 2022,
        "Codi_Districte": 1,
        "Nom_Districte": "Ciutat Vella",
        "Codi_Barri": 1,
        "Nom_Barri": "el Raval",
        "AEB": 1,
        "Seccio_Censal": 1001,
        "Valor": 10,
        "NACIONALITAT_G": 1,
        "EDAT_Q": 0,   # Por ejemplo, edad entre 0-4 años
        "SEXE": 1
    },
    {
        "Any": 2022,
        "Codi_Districte": 3,
        "Nom_Districte": "Sants-Montjuïc",
        "Codi_Barri": 18,
        "Nom_Barri": "Sants",
        "AEB": 72,
        "Seccio_Censal": 3094,
        "Valor": 10,
        "NACIONALITAT_G": 3,
        "EDAT_Q": 2,  # Edad entre 5-9 años
        "SEXE": 2
    }
]

# Llamamos a la función para insertar los nuevos documentos en la colección
insert_data(new_document, collection)
```

```
Conexión establecida con MongoDB.
Datos insertados correctamente.
```

La función "*insert_data*" se encarga de insertar los datos en una colección de MongoDB, tomando dos parámetros: "*data*" (lista de documentos a insertar) y "*collection*" (colección de MongoDB donde se insertarán los datos). También, verifica si la lista de documentos no se encuentra vacía antes de insertarlos en la colección.

Se define una lista denominada "*new_document*" compuesta por dos diccionarios, cada uno representando un documento que se insertará en la base de datos. Estos documentos simulan datos de inmigrantes que se insertarán en la colección.

3.2 Lectura

En el siguiente paso, filtraremos por el nombre del barrio “Sants” para que nos muestre únicamente los datos de las personas pertenecientes a dicho barrio.

Lectura

```
def get_immigrant_data(collection, filter={}):
    try:
        # Obtenemos datos según el filtro indicado y convertirmos el cursor de resultados a una lista de Python
        result = collection.find(filter)
        return list(result)
    except pymongo.errors.PyMongoError as e:
        print(f'Error al obtener datos desde MongoDB: {e}')
        return []

# Datos de inmigrantes del barrio "Sants"
immigrant_data_barri = get_immigrant_data(collection, {"Nom_Barri": "Sants"})
df = pd.DataFrame(immigrant_data_barri)
df
```

	_id	Any	Codi_Districte	Nom_Districte	Codi_Barri	Nom_Barri	AEB	Seccio_Censal	Valor	NACIONALITAT_G	EDAT_Q	SEXE
0	6639342032bf1669344bf074	2022	3	Sants-Montjuïc	18	Sants	72	3086	..	1	0	2
1	6639342032bf1669344bf075	2022	3	Sants-Montjuïc	18	Sants	72	3086	..	1	1	2
2	6639342032bf1669344bf076	2022	3	Sants-Montjuïc	18	Sants	72	3086	..	1	2	2
3	6639342032bf1669344bf077	2022	3	Sants-Montjuïc	18	Sants	72	3086	..	1	4	1
4	6639342032bf1669344bf078	2022	3	Sants-Montjuïc	18	Sants	72	3086	..	1	4	2
...	...	...	...	...	...	...	...	...	...	...	...	...
6593	663a7ece94dd4bccae2c426c	2022	3	Sants-Montjuïc	18	Sants	76	3114	..	3	11	2
6594	663a7ece94dd4bccae2c426d	2022	3	Sants-Montjuïc	18	Sants	76	3114	..	3	13	1
6595	663a7ece94dd4bccae2c426e	2022	3	Sants-Montjuïc	18	Sants	76	3114	..	3	16	1
6596	663a7ece94dd4bccae2c426f	2022	3	Sants-Montjuïc	18	Sants	76	3114	..	3	17	2
6597	663a7ee294dd4bccae2cbdd3	2022	3	Sants-Montjuïc	18	Sants	72	3094	10	3	2	2

6598 rows × 12 columns

La función “*get_inmigrant_data*” toma dos parámetros: “*collection*” (colección de MongoDB de la cual queremos obtener los datos) y “*filter*” (diccionario que representa el filtro que se aplicará a la consulta). Al crear la función, definimos por defecto el filtro vacío para contar así con todos los documentos de la colección. De esta forma, cuando queramos aplicar un filtro para obtener datos específicos, por ejemplo, de inmigrantes que pertenecen al barrio Sants, basta con introducirlo en la función como segundo argumento en forma de un diccionario {“Nom_Barri”: “Sants”}. Esto significa que solo queremos obtener documentos de inmigrantes cuyo campo “Nom_Barri” sea igual a “Sants”.

3.3 Actualización

A continuación, realizamos la actualización de los datos. Ahora, el valor asociado al barrio de Sants pasará a ser 15.

Actualización

```
def update_immigrant_data(collection, filter, update):
    try:
        # Definimos un filtro para encontrar el documento que queremos actualizar
        collection.update_many(filter, {"$set": update})
        print("Datos actualizados correctamente.")
    except pymongo.errors.PyMongoError as e:
        print(f'Error al actualizar datos en MongoDB: {e}')

# Actualizamos el documento encontrado con las modificaciones definidas
update_filter = {"Nom_Barri": "Sants"}
update_data = {"Valor": 15}  # Actualiza el valor a 15
update_immigrant_data(collection, update_filter, update_data)
```

```
Datos actualizados correctamente.
```

La función "*update_immigrant_data*" toma tres parámetros: "collection" (colección de MongoDB en la que queremos realizar la actualización), "*filter*" (diccionario que representa el filtro para encontrar el documento que queremos actualizar) y "*update*" (diccionario que contiene las modificaciones que queremos aplicar al documento encontrado). En este caso, estamos buscando documentos donde el campo "Nom_Barri" sea igual a "Sants" para que, mediante el diccionario "*update_data*" se actualice el campo "Valor" a 15.

3.4 Eliminación

Por último, eliminaremos documentos de la base de datos. A partir de ahora, se habrán eliminado de nuestra base de datos los documentos pertenecientes al barrio de Sants.

Eliminación

```
def delete_immigrant_data(collection, filter):
    try:
       # Definimos un filtro para encontrar el documento que queremos eliminar
        collection.delete_many(filter)
        print("Datos eliminados correctamente.")
    except pymongo.errors.PyMongoError as e:
        print(f'Error al eliminar datos en MongoDB: {e}')

# Eliminamos el documento encontrado que cumple con el filtro
delete_filter = {"Nom_Barri": "Sants"}
delete_immigrant_data(collection, delete_filter)
```

```
Datos eliminados correctamente.
```

La función "*delete_immigrant_data*" toma dos parámetros: "*collection*" (colección de MongoDB de la cual queremos eliminar documentos) y "*filter*" (diccionario que representa el filtro para encontrar el documento que queremos eliminar). Dentro de la función, utilizamos "*collection.delete_many(filter)*" para eliminar, mediante "*delete_many*", todos los documentos que cumplan con el filtro que especificaremos a continuación. Igual que en el punto 3.3, estamos buscando documentos donde el campo "Nom_Barri" sea igual a "Sants", por lo que usaremos el diccionario "*delete_filter*" para que nos ayude a encontrar el documento a eliminar.

4. Análisis

Para realizar un análisis más enfocado a nuestra base de datos, cargaremos de nuevo los datos desde la URL y los reinsertaremos en la colección para no tener en cuenta las modificaciones que hemos realizado anteriormente.

```
# URL del archivo JSON original
url = 'https://opendata-ajuntament.barcelona.cat/resources/bcn/EstadisticaPadro/imm/2022/2022_pad_imm_mdbas_sexe_edat-q_nacionalitat-g.json'

# Solicitud GET para obtener el contenido del archivo JSON
response = requests.get(url)

# Verificamos el estado de la solicitud y cargamos el contenido del archivo en una variable Python
if response.status_code == 200:
    original_data = response.json()
    print("El archivo JSON original se ha cargado correctamente desde la URL.")
else:
    print("Error al descargar el archivo JSON.")

# Establecemos la conexión con el servidor de MongoDB
client = pymongo.MongoClient('mongodb://localhost:27017/')

# Accedemos a la base de datos 'barcelona_immigrants'
db = client['barcelona_immigrants']

# Accedemos a la colección 'immigration_data'
collection = db['immigration_data']

# Eliminamos todos los documentos existentes en la colección
collection.delete_many({})

# Insertamos los datos originales en la colección
collection.insert_many(original_data)

print("\nSe han restaurado los datos originales en la base de datos MongoDB.")
```

```
El archivo JSON original se ha cargado correctamente desde la URL.

Se han restaurado los datos originales en la base de datos MongoDB.
```

4.1 Inmigrantes según nacionalidad y sexo

Primero, calcularemos el número total de inmigrantes según nacionalidad (española, europea o de fuera de la Unión Europea) y sexo.

Para ello, utilizaremos un pipeline de agregación compuesto por una primera etapa ($*group*), que agrupa los documentos de la colección por nacionalidad y sexo y calcula el total de documentos ($*sum*: 1) en cada grupo; y una segunda etapa ($*project*) para mostrar los resultados de la agrupación, seleccionando específicamente los campos deseados (Nacionalidad, Sexo, y Total) y renombrando los campos para simplificar la estructura del documento resultante ("_id.NACIONALITAT_G" como "NACIONALITAT_G" y "_id.SEXE" como "SEXE").

Inmigrantes según nacionalidad y sexo

```
# Pipeline de agregación para contar inmigrantes por nacionalidad y sexo
pipeline = [
    {"$group": {"_id": {"NACIONALITAT_G": "$NACIONALITAT_G", "SEXE": "$SEXE"},"Total": {"$sum": 1}}},
    { "$project": {"_id": 0, "NACIONALITAT_G": "$_id.NACIONALITAT_G", "SEXE": "$_id.SEXE", "Total": 1}}
]

# Realizamos una consulta para obtener los datos procesados mediante el pipeline de agregación
cursor = collection.find()

# Creamos un DataFrame a partir de los documentos obtenidos de la consulta
df = pd.DataFrame(list(cursor))

# Convertimos las columnas de interés a tipos numéricos
df['NACIONALITAT_G'] = df['NACIONALITAT_G'].astype(int)
df['SEXE'] = df['SEXE'].astype(int)
```

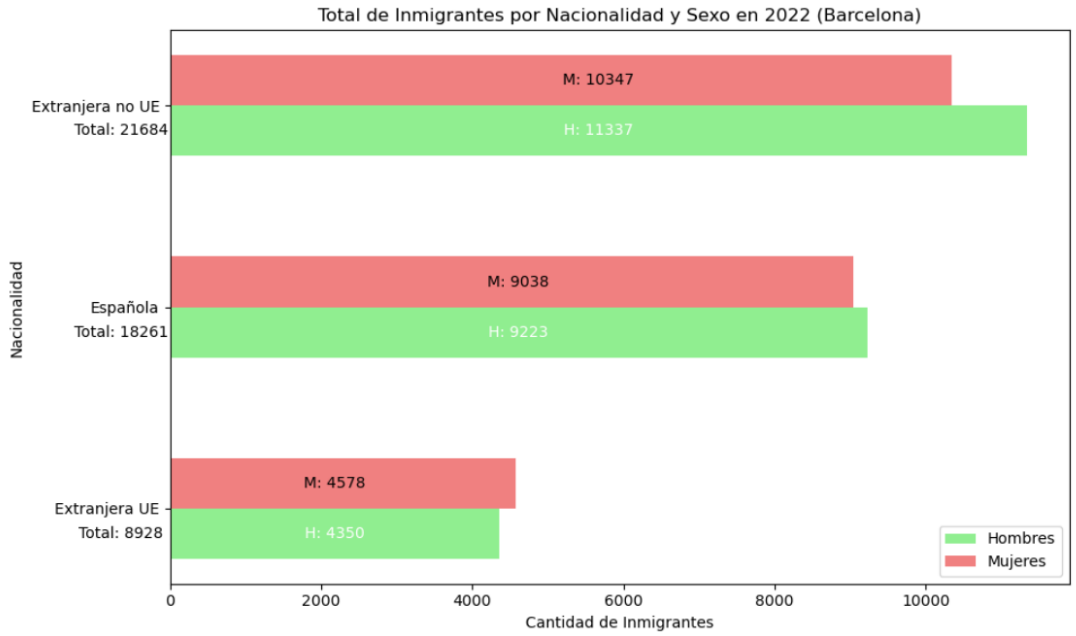
```python
# Agrupamos por nacionalidad y sexo, contando el número de inmigrantes en cada grupo
grouped_data = df.groupby(['NACIONALITAT_G', 'SEXE']).size().reset_index(name='Total')

# Mapeamos los códigos de nacionalidad para agruparlos más fácilmente
nacionalidad_map = {
    1: 'Española',
    2: 'Extranjera UE',
    3: 'Extranjera no UE',
    None: 'Desconocida'
}

# Reemplazamos los códigos de nacionalidad con nombres descriptivos mediante 'map'
grouped_data['Nacionalidad'] = grouped_data['NACIONALITAT_G'].map(nacionalidad_map)
grouped_data['Sexo'] = grouped_data['SEXE']

# Mostramos los datos agrupados antes de graficar
print("Datos agrupados por nacionalidad y sexo:\n")
print(f"{'Nacionalidad':<20} {'Sexo':<10} {'Total':<5}")
print("-" * 40)
for index, row in grouped_data.iterrows():
    print(f"{row['Nacionalidad']:<20} {row['Sexo']:<10} {row['Total']:<5}")
```

```
Datos agrupados por nacionalidad y sexo:

Nacionalidad         Sexo       Total
----------------------------------------
Española             1          9223
Española             2          9038
Extranjera UE        1          4350
Extranjera UE        2          4578
Extranjera no UE     1          11337
Extranjera no UE     2          10347
nan                  1          60
nan                  2          39
```

Pasamos a graficar los resultados para una mejor visualización de los datos. A continuación, se encuentra el código utilizado para crear un gráfico de barras horizontales ordenadas de mayor a menor según la cantidad total de inmigrantes que compone cada categoría de nacionalidad (distinguiéndose entre hombres y mujeres en cada caso).

```python
# Sumamos el total de inmigrantes (hombres + mujeres) por nacionalidad
total_inmigrantes = grouped_data.groupby('Nacionalidad')['Total'].sum().reset_index()

# Ordenamos las nacionalidades de mayor a menor según el total de inmigrantes
total_inmigrantes = total_inmigrantes.sort_values(by='Total', ascending=True)

# Configuramos el gráfico de barras horizontal
plt.figure(figsize=(10, 6))

# Obtenemos las posiciones en el eje Y para las barras agrupadas
y_pos = np.arange(len(total_inmigrantes))

# Ajustamos el ancho de las barras
bar_width = 0.25

# Obtenemos los datos para hombres y mujeres de cada nacionalidad
male_data = grouped_data[grouped_data['Sexo'] == 1]
female_data = grouped_data[grouped_data['Sexo'] == 2]

# Graficamos las barras para hombres y mujeres por cada nacionalidad
for i, (nacionalidad, total) in enumerate(zip(total_inmigrantes['Nacionalidad'], total_inmigrantes['Total'])):
    # Filtramos los datos por nacionalidad y lo aplicamos en hombres y mujeres
    national_data = grouped_data[grouped_data['Nacionalidad'] == nacionalidad]
    males = national_data[national_data['Sexo'] == 1]['Total'].sum()
    females = national_data[national_data['Sexo'] == 2]['Total'].sum()

    # Graficamos la barra para hombres (verde) y mujeres (coral) y ponemos dentro de cada uno la cantidad de inmigrantes por sexo
    plt.barh(y_pos[i], males, color='lightgreen', height=bar_width, label='Hombres')
    plt.text(males / 2, y_pos[i], f'H: {males}', ha='center', va='center', color='white', fontsize=10)
    plt.barh(y_pos[i] + bar_width, females, color='lightcoral', height=bar_width, label='Mujeres')
    plt.text(males / 2, y_pos[i] + bar_width, f'M: {females}', ha='center', va='center', color='black', fontsize=10)

    # Mostramos el total debajo de cada nacionalidad en el eje Y
    plt.text(-650, y_pos[i], f'Total: {males + females}', ha='center', va='center', color='black', fontsize=10)

# Configuramos las etiquetas y el título del gráfico
plt.xlabel('Cantidad de Inmigrantes')
plt.ylabel('Nacionalidad')
plt.yticks(y_pos + bar_width / 2, total_inmigrantes['Nacionalidad'])     # Mostramos las nacionalidades en el eje Y
plt.title('Total de Inmigrantes por Nacionalidad y Sexo en 2022 (Barcelona)')
```

```
# Mostramos el gráfico y su leyenda
plt.legend(['Hombres', 'Mujeres'])
plt.tight_layout()
plt.show()
```



Los resultados obtenidos nos arrojan los siguientes datos:

- La nacionalidad más común entre los inmigrantes registrados en el año 2022 es la "Extranjera no UE" (personas provenientes de fuera de la Unión Europea), seguida por la "Española" (españoles de otras comunidades autónomas) y la "Extranjera UE" (provenientes de otros países europeos). La nacionalidad "Extranjera no UE" presenta un total combinado de 21.684 inmigrantes (11.337 hombres y 10.347 mujeres); mientras que la nacionalidad "Española" cuenta con un total combinado de 18.261 inmigrantes (9.223 hombres y 9.038 mujeres); y la "Extranjera UE" total combinado de 8.928 inmigrantes (4.350 hombres y 4.578 mujeres).
  La inmigración de fuera de la UE podría estar relacionada con oportunidades laborales, crecimiento económico o la búsqueda de mejores condiciones de vida en Barcelona, pues la mayoría provienen de países con economías en desarrollo o en crisis, donde las oportunidades laborales y los estándares de vida pueden ser menos favorables que en Europa. También, las políticas de inmigración en España y en la UE pueden haber influido en este flujo migratorio.
- En términos de sexo, parece haber una distribución bastante equitativa entre hombres (Sexo 1) y mujeres (Sexo 2) en cada grupo de nacionalidad. En la categoría "Extranjera no UE" se puede observar que hay más mujeres que hombres (al contrario que en las otras categorías), pero la diferencia no es significativa. La distribución equitativa por género podría estar relacionada con la igualdad de oportunidades laborales en el país de destino.
- Hay un pequeño número de registros (60 para hombres y 39 para mujeres) donde la nacionalidad no está especificada (NaN). Esto podría deberse a la falta de documentación adecuada en el momento del registro.

4.2 Top 5 barrios con mayor número de inmigrantes

Para calcular los cinco barrios con mayor número de inmigrantes en Barcelona crearemos un índice en el campo “Nom_Barri” de la colección para mejorar el rendimiento de las consultas que involucran dicho campo, haciendo que las búsquedas sean más eficientes.

Igual que en el caso anterior, vamos a definir un pipeline de agregación utilizando una lista de etapas. La primera etapa ($*group*) agrupa los documentos por el campo “Nom_Barri” y calcula el conteo de documentos ($*sum*: 1) en cada grupo. La segunda etapa ($*sort*) ordena los resultados en orden descendente según el conteo. Por último, la tercera etapa ($*limit*) limita los resultados a los primeros cinco barrios, es decir, obtiene el Top 5 de barrios con más inmigrantes en el año 2022.

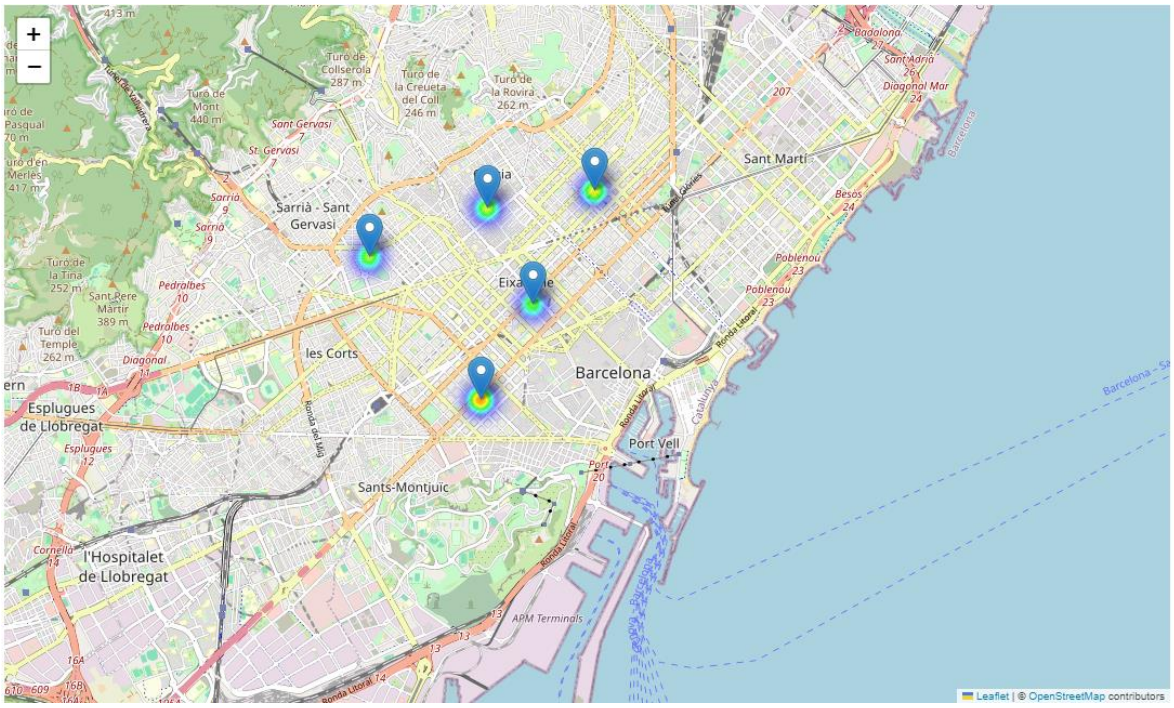
Top 5 barrios con mayor número de inmigrantes

```python
def barrio_index(collection):
    try:
        # Creamos un índice en el campo Nom_Barri para mejorar el rendimiento de la consulta
        collection.create_index([("Nom_Barri", pymongo.ASCENDING)])
        print("Índice en 'Nom_Barri' creado correctamente.\n")
    except pymongo.errors.PyMongoError as e:
        print(f'Error al crear índice en MongoDB: {e}')

def top_barrios(collection):
    try:
        pipeline = [
            {"$group": {"_id": "$Nom_Barri", "count": {"$sum": 1}}},
            {"$sort": {"count": -1}},
            {"$limit": 5}  # Para hacer el Top 5
        ]
        top_barrios = list(collection.aggregate(pipeline))
        return top_barrios
    except pymongo.errors.PyMongoError as e:
        print(f'Error al obtener top 5 de nacionalidades en MongoDB: {e}')
        return []

# Crearmos el índice en Nom_Barri
barrio_index(collection)

# Obtenermos los 5 barrios con mayor población de inmigrantes
top_5_barrios = top_barrios(collection)

# Mostramos los resultados en forma de tabla
print("Top 5 Barrios con mayor número de inmigrantes en Barcelona:\n")
print(f"{'Barrio':<40} {'Total':<10}")
print("-" * 50)
for barrio in top_5_barrios:
    print(f"{barrio['_id']:<40} {barrio['count']:<10}")
```

```
Índice en 'Nom_Barri' creado correctamente.

Top 5 Barrios con mayor número de inmigrantes en Barcelona:

Barrio                                   Total
--------------------------------------------------
la Nova Esquerra de l'Eixample           1877
la Sagrada Família                       1718
la Vila de Gràcia                        1647
Sant Gervasi - Galvany                   1612
la Dreta de l'Eixample                   1538
```

Para poder visualizar mejor los resultados obtenidos crearemos un mapa de Barcelona que muestre con puntos rojos los barrios más poblados por inmigrantes en 2022. A continuación, presentamos el código que lo genera y posteriormente el mapa obtenido:

```python
# Definimos las coordenadas de Barcelona para crear el mapa
map_barcelona = folium.Map(location=[41.3851, 2.1734], zoom_start=13)

# Datos de los barrios con mayor número de inmigrantes obtenidos del top 5
top_5_barrios = [
    {"_id": "La Nova Esquerra de l'Eixample", "count": 7508},
    {"_id": "La Sagrada Família", "count": 6872},
    {"_id": "La Vila de Gràcia", "count": 6588},
    {"_id": "Sant Gervasi - Galvany", "count": 6448},
    {"_id": "La Dreta de l'Eixample", "count": 6152}
]

# Preparamos los datos para el mapa de calor y los marcadores
data_heatmap = []

for barrio in top_5_barrios:
    # Obtenemos el nombre del barrio y su población
    nombre_barrio = barrio['_id']
    inmigrantes = barrio['count']

    # Tras buscar las coordenadas de cada barrio, las incluimos en el mapa.
    if nombre_barrio == "La Nova Esquerra de l'Eixample":
        lat, lon = 41.3797, 2.1564
    elif nombre_barrio == "La Sagrada Família":
        lat, lon = 41.4045, 2.1744
    elif nombre_barrio == "La Vila de Gràcia":
        lat, lon = 41.4024, 2.1575
    elif nombre_barrio == "Sant Gervasi - Galvany":
        lat, lon = 41.3966, 2.1388
    elif nombre_barrio == "La Dreta de l'Eixample":
        lat, lon = 41.3910, 2.1647
    else:
        continue

    # Agregamos las coordenadas y el peso (número de inmigrantes) para el mapa de calor
    if not np.isnan(lat) and not np.isnan(lon) and not np.isnan(inmigrantes):
        data_heatmap.append([lat, lon, inmigrantes])

        # Creamos marcadores emergentes para que al hacer clic en los puntos del mapa podamos ver la información de cada barrio
        folium.Marker(
            location=[lat, lon],
            popup=f"{nombre_barrio}: {inmigrantes} inmigrantes",
            icon=None  # Usamos el ícono predeterminado para los marcadores
        ).add_to(map_barcelona)

# Creamos el mapa de calor con los datos
heatmap_layer = HeatMap(data=data_heatmap, radius=15)
map_barcelona.add_child(heatmap_layer)

# Mostrar finalmente el mapa interactivo con el mapa de calor y marcadores
map_barcelona
```



Podemos concluir lo siguiente:

- Los cinco barrios más poblados por inmigrantes en Barcelona en el año 2022 ordenados de mayor a menor son: La Nova Esquerra de l'Eixample con 1.877 inmigrantes, La Sagrada Familia con 1.718, La Vila de Gràcia con 1.647, Sant Gervasi-Galvany con 1.612 y La Dreta de l'Eixample con 1.538. La concentración de población inmigrante en esta zona de la ciudad puede deberse a factores como la accesibilidad a servicios públicos, como educación, transporte o atención médica.

4.3 Inmigrantes según nacionalidad y barrio

Además de saber los cinco barrios con mayor número de inmigrantes en Barcelona, también vamos a analizar en qué barrios se encuentra la mayor cantidad de españoles procedentes de otra ciudad, europeos y personas provenientes de fuera de la Unión Europea.

El pipeline de agregación definido para este apartado se divide en: primera etapa ($*match*) para filtrar los documentos que tienen el campo de Nacionalidad igual a 1 (española), 2 (UE) o 3 (fuera de la UE), y segunda etapa ($*group*) para agrupar los documentos por el nombre del barrio y la nacionalidad calculando el conteo total de documentos en cada grupo.

Inmigrantes según nacionalidad y barrio

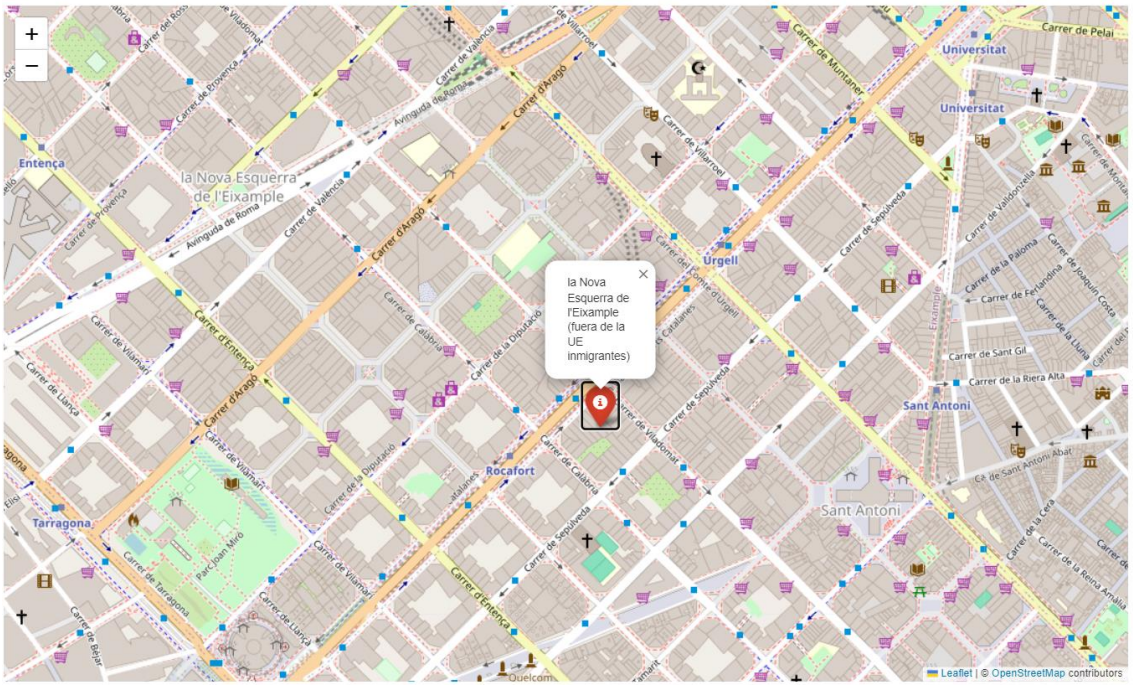
```
# Filtramos los datos por las nacionalidades de interés (1: Española, 2: UE, 3: Fuera de la UE)
pipeline = [
    {"$match": {"NACIONALITAT_G": {"$in": [1, 2, 3]}}},
    {"$group": {"_id": {"Nom_Barri": "$Nom_Barri", "NACIONALITAT_G": "$NACIONALITAT_G"}, "Total": {"$sum": 1}}}
]

# Ejecutamos la consulta en MongoDB
results = list(collection.aggregate(pipeline))

# Creamos un DataFrame a partir de los resultados
df = pd.DataFrame(results)

# Agregamos una columna llamada 'Nacionalidad' basada en el mapeo de códigos del ejercicio anterior
df['Nacionalidad'] = df['_id'].apply(lambda x: nacionalidad_map.get(x['NACIONALITAT_G']))
# Extraemos los datos de 'Nom_Barri'
df['Nom_Barri'] = df['_id'].apply(lambda x: x.get('Nom_Barri'))

# Filtramos por las nacionalidades específicas y agrupamos por barrio para obtener los totales
spanish_immigrants = df[df['Nacionalidad'] == 'Española'].groupby('Nom_Barri')['Total'].sum().sort_values(ascending=False)
eu_immigrants = df[df['Nacionalidad'] == 'Extranjera UE'].groupby('Nom_Barri')['Total'].sum().sort_values(ascending=False)
non_eu_immigrants = df[df['Nacionalidad'] == 'Extranjera no UE'].groupby('Nom_Barri')['Total'].sum().sort_values(ascending=False)

# Obtenemos el barrio con más inmigrantes españoles, de la UE y fuera de la UE y su cantidad
barrio_most_spanish = spanish_immigrants.idxmax()
num_spanish_immigrants = spanish_immigrants.max()

barrio_most_eu = eu_immigrants.idxmax()
num_eu_immigrants = eu_immigrants.max()

barrio_most_non_eu = non_eu_immigrants.idxmax()
num_non_eu_immigrants = non_eu_immigrants.max()

# Mostramos los resultados
print(f"Barrio con más inmigrantes españoles -----> {barrio_most_spanish} ({num_spanish_immigrants} inmigrantes)\n")
print(f"Barrio con más inmigrantes de la UE -----> {barrio_most_eu} ({num_eu_immigrants} inmigrantes)\n")
print(f"Barrio con más inmigrantes fuera de la UE -----> {barrio_most_non_eu} ({num_non_eu_immigrants} inmigrantes)\n")
```

```
Barrio con más inmigrantes españoles -----> la Nova Esquerra de l'Eixample (699 inmigrantes)

Barrio con más inmigrantes de la UE -----> la Vila de Gràcia (417 inmigrantes)

Barrio con más inmigrantes fuera de la UE -----> la Nova Esquerra de l'Eixample (817 inmigrantes)
```

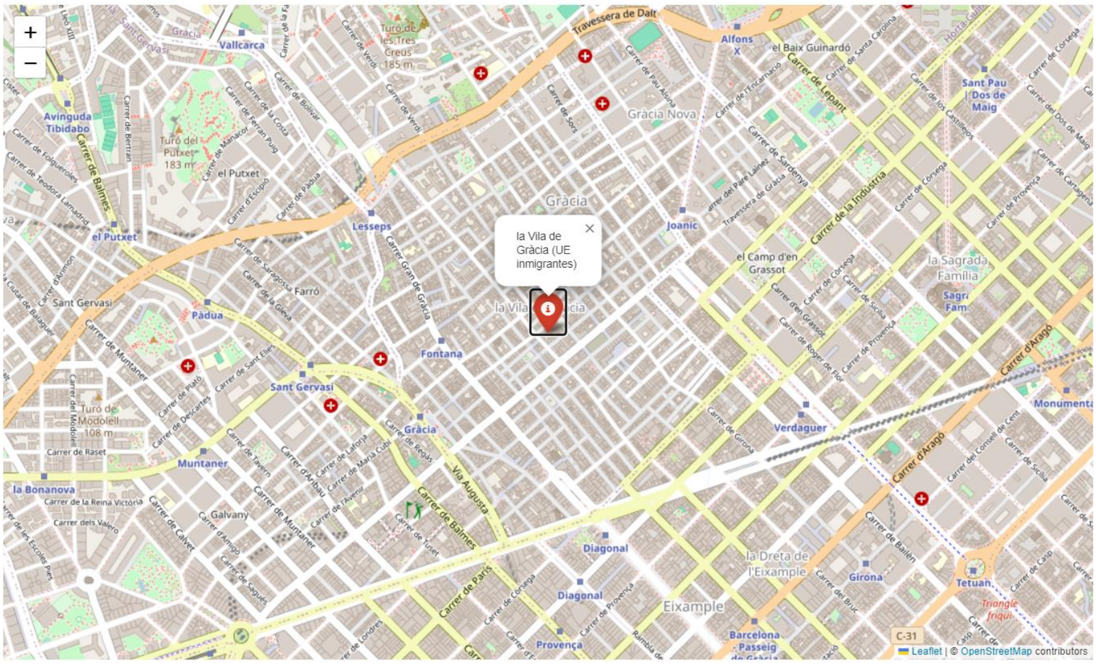
Para visualizar mejor los datos, crearemos de nuevo un mapa centrado en cada barrio. En el caso de la Nova Esquerra de l'Eixample, los datos de inmigrantes de fuera de la UE se solapan con los datos de inmigrantes españoles por lo que, al ser los últimos obtenidos, solo se mostrará su mapa.

```python
# Definimos una función para crear un mapa centrado en los barrios obtenidos en los resultados
def crear_mapa_barrio(lat, lon, barrio, tipo_inmigrantes):
    # Creamos un mapa centrado en las coordenadas que añadiremos más tarde
    mapa_barrio = folium.Map(location=[lat, lon], zoom_start=15)

    # Agregamos un marcador al mapa para mostrar el barrio y el tipo de inmigrantes
    folium.Marker(
        location=[lat, lon],
        popup=f"{barrio} ({tipo_inmigrantes} inmigrantes)",
        icon=folium.Icon(color='red')
    ).add_to(mapa_barrio)

    # Mostramos el mapa interactivo
    return mapa_barrio

# Datos de los barrios con mayor número de inmigrantes
barrios_inmigrantes = {
    "la Nova Esquerra de l'Eixample": {"lat": 41.3797, "lon": 2.1564, "tipo": "españoles", "inmigrantes": 699},
    "la Vila de Gràcia": {"lat": 41.4024, "lon": 2.1575, "tipo": "UE", "inmigrantes": 417},
    "la Nova Esquerra de l'Eixample": {"lat": 41.3797, "lon": 2.1564, "tipo": "fuera de la UE", "inmigrantes": 817}
}

# Creamos y mostramos cada mapa para los barrios especificados
for barrio, info in barrios_inmigrantes.items():
    lat = info['lat']
    lon = info['lon']
    tipo_inmigrantes = info['tipo']
    inmigrantes = info['inmigrantes']

    mapa = crear_mapa_barrio(lat, lon, barrio, tipo_inmigrantes)

    # Mostramos el mapa con el título correspondiente
    print(f"\nMapa de '{barrio}' con inmigrantes '{tipo_inmigrantes}'\n")
    display(mapa)
```

```
Mapa de 'la Nova Esquerra de l'Eixample' con inmigrantes 'fuera de la UE'
```



Mapa de 'la Vila de Gràcia' con inmigrantes 'UE'



Como conclusiones podemos destacar lo siguiente:

- La Nova Esquerra de l'Eixample es el barrio más poblado por inmigrantes españoles y de fuera de la UE. Uno de los motivos puede ser la vivienda relativamente accesible en esta zona en comparación con otras áreas céntricas de la ciudad.
- La Vila de Gràcia es el barrio más poblado por inmigrantes europeos. Se trata de un área comercial y turística que podría atraer a inmigrantes de la UE en busca de oportunidades laborales en sectores como el turismo, la hostelería o el comercio.

4.4 Inmigrantes por edad quinquenal

En este apartado calcularemos el número total de inmigrantes según su edad quinquenal, agrupándolos según:

- Niños: quinquenal de 1-3.
- Adolescentes: quinquenal 4.
- Jóvenes: quinquenal de 5-9.
- Adultos: quinquenal de 10-19.
- Personas mayores: a partir del quinquenal 19.

El pipeline de agregación para este análisis resulta más complejo que los anteriores. Su objetivo es contar inmigrantes por categoría de edad quinquenal utilizando una serie de condiciones definidas con el operador ($*switch*). Dentro de este operador, se evalúan varias condiciones ("*branches*") utilizando el campo de categoría de edad quinquenal ("EDAT_Q"):

- ✓ Si EDAT_Q es menor o igual a 3, la categoría de edad asignada es "Niños" (de 0 a 14 años).
- ✓ Si EDAT_Q es menor o igual a 4, la categoría de edad es "Adolescentes" (de 15 a 19 años).
- ✓ Si EDAT_Q es menor o igual a 9, la categoría de edad es "Jóvenes" (de 20 a 39 años).
- ✓ Si EDAT_Q es menor o igual a 19, la categoría de edad es "Adultos" (de 40 a 59 años).
- ✓ Si EDAT_Q es mayor que 19, la categoría de edad es "Personas mayores" (de 60 años en adelante).

- ✓ La condición "*default*": se utiliza para casos que no cumplen ninguna de las condiciones anteriores.

El operador ($*project*) se utiliza para cambiar la estructura de los documentos resultantes. Es decir, los usamos para renombrar el campo _id (categoría de edad) como "Edad" y mantener el campo "Total" para contar el número de inmigrantes en cada categoría.

Por último, en lugar de añadir un operador ($*sort*) para ordenar los resultados, creamos una variable fuera del pipeline para poder personalizar el orden que queremos obtener.

Inmigrantes por edad quinquenal

```
# Pipeline de agregación para contar inmigrantes por categoría de edad quinquenal
pipeline = [
    {
        "$group": {
            "_id": {
                "$switch": {
                    "branches": [
                        {"case": {"$lte": ["$EDAT_Q", 3]}, "then": "Niños"},  # De 0 a 14 años
                        {"case": {"$lte": ["$EDAT_Q", 4]}, "then": "Adolescentes"}, # De 15 al 19 años
                        {"case": {"$lte": ["$EDAT_Q", 9]}, "then": "Jóvenes"}, # de 20 a 39 años
                        {"case": {"$lte": ["$EDAT_Q", 19]}, "then": "Adultos"}, # De 40 a 59 años
                        {"case": {"$gt": ["$EDAT_Q", 19]}, "then": "Personas mayores"}  # De 60 años en adelante
                    ],
                    "default": "Otro"
                }
            },
            "Total": {"$sum": 1}
        }
    },
    {
        "$project": {
            "_id": 0,
            "Edad": "$_id",
            "Total": 1
        }
    },
]

# Realizamos una consulta para obtener los datos agrupados por edad quinquenal
cursor = collection.aggregate(pipeline)

# Creamos un DataFrame a partir de los documentos obtenidos de la consulta
df = pd.DataFrame(list(cursor))

# Definimos el orden de las categorías para una mejor visibilidad
orden_categorias = ["Niños", "Adolescentes", "Jóvenes", "Adultos", "Personas mayores"]

# Creamos una nueva columna en el DataFrame para asignar el orden específico
df['Orden'] = pd.Categorical(df['Edad'], categories=orden_categorias, ordered=True)

# Ordenamos el DataFrame según el nuevo orden específico y eliminamos la columna de orden temporal
df_sorted = df.sort_values(by='Orden')
df_sorted.drop(columns=['Orden'], inplace=True)

# Mostramos los datos agrupados por edad quinquenal y total de inmigrantes en el orden deseado
print("Edad".ljust(20), "Total")
print("-" * 30)

for index, row in df_sorted.iterrows():
    print(row['Edad'].ljust(20), row['Total'])
```

```
Edad                 Total
------------------------------
Niños                9862
Adolescentes         4763
Jóvenes              22443
Adultos              11898
Personas mayores     6
```

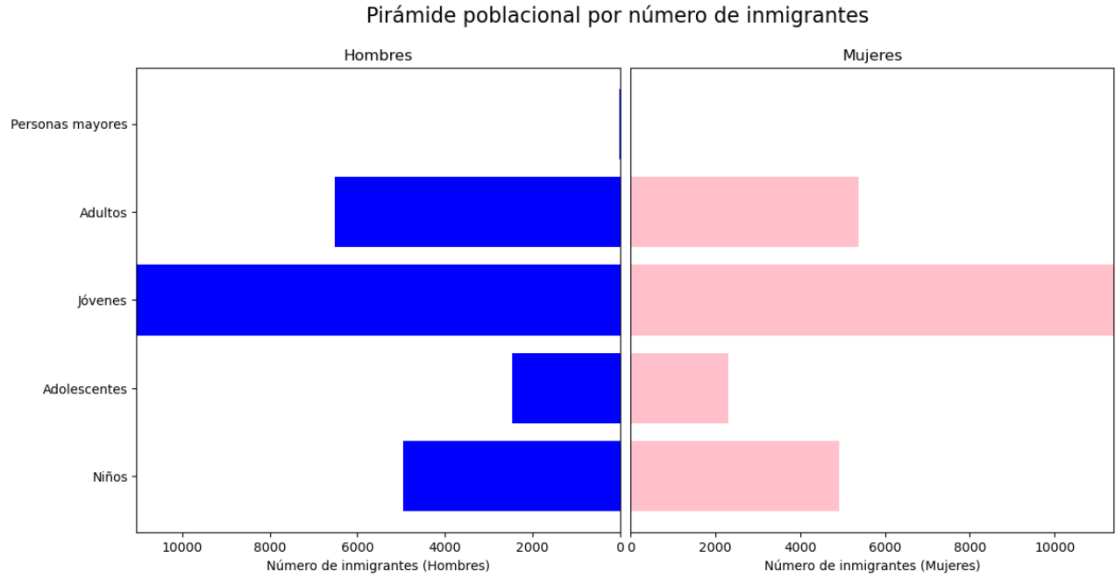
A continuación, crearemos una pirámide poblacional para visualizar mejor los resultados. Para ello, primero calcularemos cuántos hombres y mujeres constituyen cada categoría de edad.

Copiamos el pipeline anterior y en el operador (*$project*) añadimos el campo “Sexo” para obtener un conjunto de documentos que representen diferentes combinaciones de categorías de edad y sexo, junto con el número total de inmigrantes en cada combinación.

```
# Pipeline de agregación para contar inmigrantes por categoría de edad quinquenal y género
pipeline = [
    {
        "$group": {
            "_id": {
                "Edad": {
                    "$switch": {
                        "branches": [
                            {"case": {"$lte": ["$EDAT_Q", 3]}, "then": "Niños"},  # De 0 a 14 años
                            {"case": {"$lte": ["$EDAT_Q", 4]}, "then": "Adolescentes"}, # De 15 al 19 años
                            {"case": {"$lte": ["$EDAT_Q", 9]}, "then": "Jóvenes"}, # de 20 a 39 años
                            {"case": {"$lte": ["$EDAT_Q", 19]}, "then": "Adultos"}, # De 40 a 59 años
                            {"case": {"$gt": ["$EDAT_Q", 19]}, "then": "Personas mayores"}  # De 60 años en adelante
                        ],
                        "default": "Otro"
                    }
                },
                "Sexo": "$SEXE"
            },
            "Total": {"$sum": 1}
        }
    },
    {
        "$project": {
            "_id": 0,
            "Edad": "$_id.Edad",
            "Sexo": "$_id.Sexo",
            "Total": 1
        }
    }
]

# Realizamos una consulta para obtener los datos de la colección
cursor = collection.aggregate(pipeline)

# Creamos un DataFrame a partir de los documentos obtenidos de la consulta
df = pd.DataFrame(list(cursor))

# Creamos un pivot table para contar hombres y mujeres por categoría de edad
pivot_table = pd.pivot_table(df, values='Total', index='Edad', columns='Sexo', aggfunc='sum', fill_value=0)

# Cambiamos los encabezados de las columnas a 'Hombre' y 'Mujer'
pivot_table.columns = ['Hombre', 'Mujer']

# Definimos el orden personalizado de las categorías de edad
orden_categorias = ['Niños', 'Adolescentes', 'Jóvenes', 'Adultos', 'Personas mayores']

# Ordenamos el pivot table según el orden personalizado de las categorías de edad
pivot_table = pivot_table.reindex(orden_categorias)
```

```
# Mostramos el pivot table con los encabezados actualizados
print("Edad".ljust(20), "Hombre", "Mujer")
print("-" * 35)

for index, row in pivot_table.iterrows():
    print(index.ljust(20), str(row['Hombre']).ljust(6), str(row['Mujer']))

# Graficamos la pirámide poblacional combinada
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Gráfico de barras invertidas para hombres (parte izquierda)
ax1.barh(pivot_table.index, pivot_table['Hombre'], color='blue')
ax1.set_xlim(pivot_table['Hombre'].max(), 0)  # Invertimos el eje x para que no salgan cantidades negativas
ax1.set_xlabel('Número de inmigrantes (Hombres)')
ax1.set_title('Hombres')

# Gráfico de barras invertidas para mujeres (parte derecha)
ax2.barh(pivot_table.index, pivot_table['Mujer'], color='pink')
ax2.set_xlim(0, pivot_table['Mujer'].max())  # Dejamos igual el eje x
ax2.set_xlabel('Número de inmigrantes (Mujeres)')
ax2.set_yticks([])  # Eliminamos las etiquetas del eje Y para mujeres para poder mostrar los dos gráficos juntos
ax2.set_title('Mujeres')

# Ajustar el espacio entre los subplots
fig.suptitle('Pirámide poblacional por número de inmigrantes', fontsize=16, x= 0.55, y=1.03) # Insertamos un título general y lo ajustamos
plt.tight_layout(pad = 0.2)

# Mostramos la pirámide poblacional combinada
plt.show()
```

Edad	Hombre	Mujer
Niños	4945	4917
Adolescentes	2456	2307
Jóvenes	11044	11399
Adultos	6520	5378
Personas mayores	5	1



Conclusiones:

- Como se ha visto en el apartado 4.1, existe una distribución equitativa por género en los diferentes rangos de edad. Los niños suman un total de 9.862 inmigrantes (4.945 hombres y 4.917 mujeres), los adolescentes un total de 4.763 (2.456 hombres y 2.307 mujeres), los jóvenes un total de 22.443 (11.044 hombres y 11.399 mujeres), adultos un total de 11.898 (6.520 hombres y 5.378 mujeres) y personas mayores un total de 6(5 hombres y 1 mujer según el registro).
- La alta cifra de adultos y sobre todo de jóvenes inmigrantes, puede estar motivada principalmente por motivos laborales y económicos. También, tanto jóvenes como adolescentes pueden migrar en busca de educación superior o capacitación profesional en la ciudad. La presencia de niños y de personas mayores puede deberse a la reunificación familiar después de que otros miembros de la familia migraran por razones económicas.

4.5 Inmigrantes según edad y barrio

En el siguiente apartado estudiaremos la distribución de inmigrantes por los barrios de Barcelona según el rango de edad.

El pipeline utilizado es similar al que hemos usado en el apartado anterior para crear la pirámide poblacional, pero en vez de agrupar los datos por edad y sexo los agrupamos por edad y barrio. También utilizamos el operador ($*sort*) para ordenar los resultados en función del total de inmigrantes en orden descendente para que nos salgan primero los más poblados. Por último, el operador ($*project*) utiliza un ($*switch*) para asignar valores numéricos a las categorías de edad para su posterior procesamiento en Python.

Guardamos los resultados de la consulta en un DataFrame de pandas y hacemos un mapeo de los valores numéricos asignados a los rangos de edad para convertirlos en sus correspondientes categorías.

Inmigrantes según edad y barrio

```
# Pipeline de agregación para contar inmigrantes por categoría de edad quinquenal y barrio
pipeline = [
    {
        "$group": {
            "_id": {
                "Edad": {
                    "$switch": {
                        "branches": [
                            {"case": {"$lte": ["$EDAT_Q", 3]}, "then": "Niños"},  # De 0 a 14 años
                            {"case": {"$lte": ["$EDAT_Q", 4]}, "then": "Adolescentes"}, # De 15 al 19 años
                            {"case": {"$lte": ["$EDAT_Q", 9]}, "then": "Jóvenes"}, # de 20 a 39 años
                            {"case": {"$lte": ["$EDAT_Q", 19]}, "then": "Adultos"}, # De 40 a 59 años
                            {"case": {"$gt": ["$EDAT_Q", 19]}, "then": "Personas mayores"}  # De 60 años en adelante
                        ],
                        "default": "Otro"
                    }
                },
                "Nom_Barri": "$Nom_Barri"
            },
            "Total": {"$sum": 1}
        }
    },
    {
        "$sort": {
            "Total": -1  # Orden descendente por el total de inmigrantes
        }
    },
    {
        "$group": {
            "_id": "$_id.Edad",
            "Barrio_mas_poblado": {"$first": "$_id.Nom_Barri"},
            "Total_inmigrantes": {"$first": "$Total"}
        }
    },
    {
        "$project": {
            "_id": 0,
            "Edad": {
                "$switch": {
                    "branches": [
                        {"case": {"$eq": ["$_id", "Niños"]}, "then": 0},
                        {"case": {"$eq": ["$_id", "Adolescentes"]}, "then": 1},
                        {"case": {"$eq": ["$_id", "Jóvenes"]}, "then": 2},
                        {"case": {"$eq": ["$_id", "Adultos"]}, "then": 3},
                        {"case": {"$eq": ["$_id", "Personas mayores"]}, "then": 4}
                    ],
                    "default": 5
                }
            },
            "Barrio_mas_poblado": 1,
            "Total_inmigrantes": 1
        }
    },
]

# Realizamos una consulta para obtener los barrios más poblados por edad
cursor = collection.aggregate(pipeline)

# Creamos un DataFrame a partir de los documentos obtenidos de la consulta
df_result = pd.DataFrame(list(cursor))

# Definimos el orden específico de las categorías de edad
orden_edad = ["Niños", "Adolescentes", "Jóvenes", "Adultos", "Personas mayores"]

# Mapeamos los valores numéricos a las categorías de edad
df_result['Edad'] = df_result['Edad'].map({0: "Niños", 1: "Adolescentes", 2: "Jóvenes", 3: "Adultos", 4: "Personas mayores"})

# Ordenamos el DataFrame según el nuevo orden específico
df_result['Edad'] = pd.Categorical(df_result['Edad'], categories=orden_edad, ordered=True)
df_result = df_result.sort_values(by='Edad')

# Mostramos los resultados formateados
print("Edad".ljust(20), "Barrio más poblado".ljust(40), "Total de inmigrantes")
print("-" * 80)

for index, row in df_result.iterrows():
    edad_str = str(row['Edad'])  # Convertir la categoría de edad a cadena
    print(edad_str.ljust(20), row['Barrio_mas_poblado'].ljust(40), row['Total_inmigrantes'])
```

```
Edad                 Barrio más poblado                       Total de inmigrantes
--------------------------------------------------------------------------------
Niños                Sant Gervasi - Galvany                   399
Adolescentes         la Nova Esquerra de l'Eixample           200
Jóvenes              la Nova Esquerra de l'Eixample           891
Adultos              la Nova Esquerra de l'Eixample           461
Personas mayores     Sant Gervasi - la Bonanova               2
```

Los resultados nos arrojan la siguiente información:

- Sant Gervasi-Galvany es el barrio más poblado por niños inmigrantes en el año 2022 (399 inmigrantes), lo cual puede indicar que se trata de un área con características familiares deseables, como acceso a parques, escuelas y servicios adecuados para las familias.
- La Nova Esquerra de l'Eixample es el barrio más poblado por adolescentes (200), jóvenes (891) y adultos (461). Esto puede deberse a varios motivos: buena oferta educativa, oportunidades laborales, vida nocturna activa, actividades recreativas y acceso a servicios de salud.
- Sant Gervasi- La Bonanova es el más poblado por personas mayores, una opción de barrio más tranquila en comparación con los anteriores resultados y con servicios especializados para personas de este rango de edad.

4.6 Inmigrantes según nacionalidad, edad y barrio

Para terminar el análisis, analizaremos los barrios más poblados por inmigrantes españoles, de la UE y de fuera de la UE según su rango de edad.

En este pipeline utilizaremos un ($*match*) para filtrar los datos y evitar los "NaN" de aquellas nacionalidades que no se encuentran especificadas en la base de datos. Agrupamos mediante ($*group*) los datos por combinación de nacionalidad, categoría de edad y barrio calculando el total de inmigrantes en cada grupo. Utilizamos el operador ($*sort*) para ordenar los resultados por total de inmigrantes de manera descendente para que nos muestre, igual que en el apartado anterior, los barrios con mayor número de inmigrantes en cada categoría. Por último, agrupamos nuevamente los datos por nacionalidad y edad, conservando el barrio más poblado y el total de inmigrantes en cada grupo.

Se ejecuta el pipeline en la colección de MongoDB para obtener un cursor con los resultados y de esta forma convertirlos primero en una lista y luego en un DataFrame de pandas para facilitar su manipulación y visualización en Python.

Definimos un orden específico para las nacionalidades y rangos de edad y mostramos los resultados. En caso de que no haya datos para una combinación específica de nacionalidad y edad, se imprime una línea en blanco.

Inmigrantes según nacionalidad, edad y barrio

```
# Pipeline de agregación para contar inmigrantes por nacionalidad, categoría de edad quinquenal y barrio
pipeline = [
    {
        "$match": {
            # Filtramos por el mapeo anterior de nacionalidades (1: Española, 2: UE, 3: Fuera de la UE) para evitar los NaN
            "NACIONALITAT_G": {"$in": [1, 2, 3]}
        }
    },
    {
        "$group": {
            "_id": {
                "Nacionalidad": {
                    "$switch": {
                        "branches": [
                            {"case": {"$eq": ["$NACIONALITAT_G", 1]}, "then": "Español"},
                            {"case": {"$eq": ["$NACIONALITAT_G", 2]}, "then": "UE"},
                            {"case": {"$eq": ["$NACIONALITAT_G", 3]}, "then": "Fuera de la UE"}
                        ],
                        "default": "Otro"
                    }
                },
                "Edad": {
                    "$switch": {
                        "branches": [
                            {"case": {"$lte": ["$EDAT_Q", 3]}, "then": "Niños"},  # De 0 a 14 años
                            {"case": {"$lte": ["$EDAT_Q", 4]}, "then": "Adolescentes"}, # De 15 al 19 años
                            {"case": {"$lte": ["$EDAT_Q", 9]}, "then": "Jóvenes"}, # de 20 a 39 años
                            {"case": {"$lte": ["$EDAT_Q", 19]}, "then": "Adultos"}, # De 40 a 59 años
                            {"case": {"$gt": ["$EDAT_Q", 19]}, "then": "Personas mayores"}  # De 60 años en adelante
                        ],
                        "default": "Otro"
                    }
```

```
            },
            "Nom_Barri": "$Nom_Barri"
        },
        "Total": {"$sum": 1}  # Contar el número de inmigrantes
    }
},
{
    "$sort": {
        "Total": -1
    }
},
{
    "$group": {
        "_id": {
            "Nacionalidad": "$_id.Nacionalidad",
            "Edad": "$_id.Edad"
        },
        "Barrio_mas_poblado": {"$first": "$_id.Nom_Barri"},
        "Total_inmigrantes": {"$first": "$Total"}
    }
}
]

# Ejecutamos el pipeline en MongoDB
cursor = collection.aggregate(pipeline)

# Convertimos los resultados en un DataFrame de pandas
results = list(cursor)
df_result = pd.DataFrame(results)

# Orden de nacionalidades y grupos de edad
orden_nacionalidad = ["Español", "UE", "Fuera de la UE"]
orden_edad = ["Niños", "Adolescentes", "Jóvenes", "Adultos", "Personas mayores"]

# Mostramos los resultados ordenados por nacionalidad y luego por edad
print("Nacionalidad".ljust(20), "Edad".ljust(20), "Barrio más poblado".ljust(40), "Total de inmigrantes")
print("-" * 100)

# Iteramos sobre el orden de nacionalidades y luego sobre el orden de edad
for nacionalidad in orden_nacionalidad:
    for edad in orden_edad:
        # Filtramos los resultados por nacionalidad y edad
        filtro = (df_result["_id"].apply(lambda x: x['Nacionalidad']) == nacionalidad) & (df_result["_id"].apply(lambda x: x['Edad']) == edad)
        df_filtrado = df_result[filtro]

        # Verificamos si hay datos para esta combinación de nacionalidad y edad
        if not df_filtrado.empty:
            # Obtenemos el primer (y único) resultado para esta combinación
            row = df_filtrado.iloc[0]

            # Imprimimos los resultados formateados
            print(
                str(nacionalidad).ljust(20),
                str(edad).ljust(20),
                row['Barrio_mas_poblado'].ljust(40),
                row['Total_inmigrantes']
            )
        else:
            # Si no hay datos para esta combinación, imprimimos una línea en blanco
            print("".ljust(20), "".ljust(20), "".ljust(40), "")
```

```
Nacionalidad         Edad                 Barrio más poblado                       Total de inmigrantes
----------------------------------------------------------------------------------------------------
Español              Niños                Sant Gervasi - Galvany                   139
Español              Adolescentes         la Nova Esquerra de l'Eixample           68
Español              Jóvenes              la Nova Esquerra de l'Eixample           322
Español              Adultos              la Nova Esquerra de l'Eixample           202
Español              Personas mayores     Sant Gervasi - la Bonanova               2
UE                   Niños                Sant Gervasi - Galvany                   101
UE                   Adolescentes         la Vila de Gràcia                        65
UE                   Jóvenes              la Vila de Gràcia                        248
UE                   Adultos              Sant Gervasi - Galvany                   65

Fuera de la UE       Niños                la Sagrada Família                       188
Fuera de la UE       Adolescentes         la Nova Esquerra de l'Eixample           73
Fuera de la UE       Jóvenes              la Nova Esquerra de l'Eixample           340
Fuera de la UE       Adultos              la Nova Esquerra de l'Eixample           218
```

En base a los resultados obtenidos podemos concluir lo siguiente:

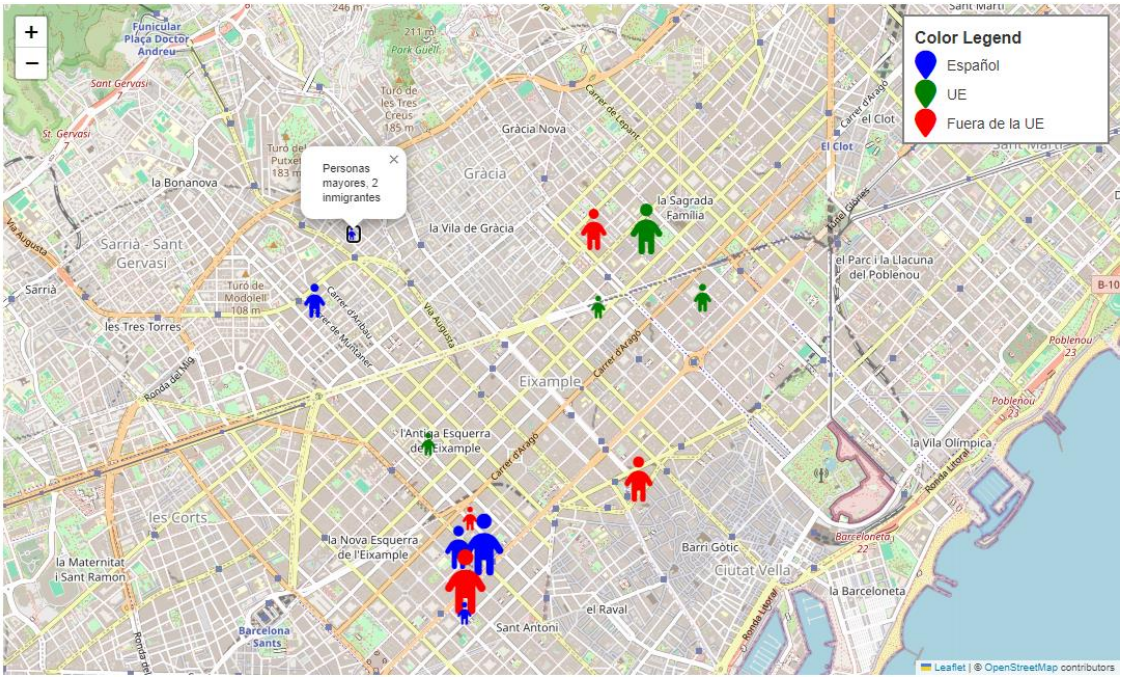
- Sant Gervasi – Galvany, como vimos en el apartado anterior, es el barrio más poblado por niños inmigrantes tanto españoles como de la Unión Europea. Sin embargo, la mayoría de niños inmigrantes de fuera de la UE se concentran en el barrio de La Sagrada Familia. Ambos barrios son conocidos por ser residenciales y con buena ubicación y servicios.

- La Nova Esquerra de l'Eixample parece ser el barrio más elegido tanto por españoles como por inmigrantes de fuera de la UE que se encuentran en los rangos de edad de adolescentes, jóvenes y adultos. Como hemos visto anteriormente, este barrio es llamativo por sus instituciones educativas, áreas de ocio activas y diversas y por sus oportunidades laborales. En cambio, jóvenes y adolescentes inmigrantes de la UE están mayormente ubicados en La Vila de Gràcia, un barrio que también presenta vida estudiantil y cultural, además de tener un mayor ambiente familiar. Asimismo, los adultos europeos muestran preferencia por Sant Gervasi - Galvany, posiblemente por su ambiente residencial y comodidades.
- Con respecto al grupo de edad de personas mayores, solo contamos con datos de inmigrantes españoles, ubicados en el barrio de Sant Gervasi – La Bonanova, un área más tranquila y residencial.

Para visualizar mejor los datos, hemos incluido un mapa interactivo de la ciudad de Barcelona que muestra iconos de personas para representar cada nacionalidad (icono azul: inmigrantes españoles, verde: de la UE y rojo: fuera de la UE). Cada icono representa, a su vez, cada grupo de edad y el número total de personas que lo conforma. Cuanto más grande sea su tamaño, significa que más densidad de población inmigrante hay en ese barrio.

```
# Creamos un mapa centrado en Barcelona
m = folium.Map(location=[41.3851, 2.1734], zoom_start=12)

# Definimos los colores para cada nacionalidad
colors = {
    'Español': 'blue',
    'UE': 'green',
    'Fuera de la UE': 'red'
}

# Agregamos la leyenda de colores y nacionalidades
legend_html = '''
<div style="position: fixed;
     top: 10px; right: 10px; width: 200px; height: 125px;
     border:2px solid grey; z-index:9999; font-size:14px; background-color:white; padding: 10px; text-align: left;
     ">
     <strong style="font-size: 16px;">Color Legend</strong><br>
     <div style="display: flex; align-items: center;">
         <i class="fa fa-map-marker fa-2x" style="color:blue; margin-right: 10px;"></i>
         <span>Español</span>
     </div>
     <div style="display: flex; align-items: center;">
         <i class="fa fa-map-marker fa-2x" style="color:green; margin-right: 10px;"></i>
         <span>UE</span>
     </div>
     <div style="display: flex; align-items: center;">
         <i class="fa fa-map-marker fa-2x" style="color:red; margin-right: 10px;"></i>
         <span>Fuera de la UE</span>
     </div>
</div>
'''

m.get_root().html.add_child(folium.Element(legend_html))

# Agregamos marcadores para cada nacionalidad y edad en los barrios correspondientes
data = {
    ('Español', 'NIÑOS'): (41.3994, 2.1438, 139),  # Coordenadas para niños españoles
    ('Español', 'Adolescentes'): (41.3797, 2.1565, 68),  # Coordenadas para adolescentes españoles
    ('Español', 'Jóvenes'): (41.3853, 2.1571, 322),  # Coordenadas para jóvenes españoles
    ('Español', 'Adultos'): (41.3845, 2.1555, 202),  # Coordenadas para adultos españoles
    ('Español', 'Personas mayores'): (41.4029, 2.1474, 2),  # Coordenadas para personas mayores españoles
    ('UE', 'NIÑOS'): (41.3994, 2.1761, 101),  # Coordenadas para niños de la UE
    ('UE', 'Adolescentes'): (41.3987, 2.1676, 65),  # Coordenadas para adolescentes de la UE
    ('UE', 'Jóvenes'): (41.4045, 2.1708, 248),  # Coordenadas para jóvenes de la UE
    ('UE', 'Adultos'): (41.3902, 2.1535, 65),  # Coordenadas para adultos de la UE
    ('Fuera de la UE', 'NIÑOS'): (41.4042, 2.1667, 188),  # Coordenadas para niños fuera de la UE
    ('Fuera de la UE', 'Adolescentes'): (41.3856, 2.1569, 73),  # Coordenadas para adolescentes fuera de la UE
    ('Fuera de la UE', 'Jóvenes'): (41.3831, 2.1555, 340),  # Coordenadas para jóvenes fuera de la UE
    ('Fuera de la UE', 'Adultos'): (41.3888, 2.1703, 218)  # Coordenadas para adultos fuera de la UE
}
```

```
# Rango para el tamaño del ícono (ajustar según tus necesidades)
min_size = 10
max_size = 50

for (nacionalidad, edad), (lat, lon, total_inmigrantes) in data.items():
    # Calculamos el tamaño del icono en función de la cantidad total de inmigrantes
    size_factor = (total_inmigrantes - min(data.values(), key=lambda x: x[2])[2]) / (max(data.values(),
                key=lambda x: x[2])[2] - min(data.values(), key=lambda x: x[2])[2])
    icon_size = min_size + (max_size - min_size) * size_factor

    folium.Marker(
        location=(lat, lon),
        popup=f"{edad}, {total_inmigrantes} inmigrantes",
        icon=folium.DivIcon(html=f'<div style="font-size: {icon_size}px; color: {colors[nacionalidad]};"><i class="fa fa-child fa-lg"></i></div>')
    ).add_to(m)

# Mostramos el mapa
m
```



5. Conclusiones

La ciudad de Barcelona presenta una diversidad demográfica importante, destacando por su población inmigrante distribuida en diversos barrios.

Tras el análisis realizado en cada apartado, hemos podido observar que la mayoría de los inmigrantes en Barcelona en el año 2022 son de nacionalidad extranjera, con una cantidad significativa (y bastante equitativa) tanto de hombres como de mujeres. La mayor parte son jóvenes y adultos, lo que indica una población migrante activa y laboralmente en edad productiva, aunque la presencia de niños y adolescentes también es notable, lo que muestra la importancia de servicios educativos y familiares en los barrios. Por ello, las políticas públicas que promuevan la integración social y educativa para todas las edades son esenciales para poder seguir manteniendo la diversidad espacial y social en Barcelona.

Por otro lado, La Nova Esquerra de l'Eixample es el barrio con más inmigrantes en general, seguido por otros barrios céntricos como la Sagrada Familia y la Vila de Gràcia. Esto se debe a factores como la ubicación, servicios disponibles y ambiente. Las políticas urbanas inclusivas también son imprescindibles para fomentar la convivencia intercultural y el desarrollo sostenible de esta ciudad.