

Grado en Ingeniería Informática  
2024-2025

*Trabajo Fin de Grado*

“Desarrollo de un dataset para el cálculo de distancias entre núcleos urbanos y servicios de ámbito comarcal con titularidad pública en la Comunidad de Madrid”

Raúl García Romero

Tutor/es

Javier García Guzmán

Leganés, 2025



Esta obra se encuentra sujeta a la licencia Creative Commons **Reconocimiento – No Comercial – Sin Obra Derivada**

**RESUMEN**

El siguiente Trabajo de Fin de Grado está orientado a el cálculo de disponibilidad de distintos servicios públicos para los distintos municipios de la Comunidad de Madrid, a partir del cálculo de la distancia media entre cada núcleo urbano y un conjunto con cada uno de los siguientes servicios públicos: Hospitales, juzgados, centros de salud mental y parques de bomberos.

Inicialmente, los datos utilizados para llevar a cabo esta práctica son distintos datasets georreferenciados, es decir, un conjunto de instancias de datos cada una ligada a un punto geográfico por medio de coordenadas. Estos datos serán preprocesados, procesados y utilizados en programas de Python y QGIS para obtener distintos conjuntos de datos que den la información necesaria.

ÍNDICE DE CONTENIDOS

[1. INTRODUCCIÓN 1](#_Toc206867155)

[1.1. Área de Trabajo 1](#_Toc206867156)

[1.2. Problema por resolver 1](#_Toc206867157)

[1.3. Motivación del Trabajo 1](#_Toc206867158)

[1.4. Objetivos 1](#_Toc206867159)

[2. ESTADO DE LA CUESTIÓN 4](#_Toc206867160)

[3. MÉTODOS 4](#_Toc206867161)

[3.1. Fuente de los datos 4](#_Toc206867162)

[3.2. Procesamiento inicial de los datos mediante QGIS 5](#_Toc206867163)

[3.3. Procesamiento adicional mediante Python 6](#_Toc206867164)

[3.4. Cálculo de distancias utilizando la API de Google Maps 6](#_Toc206867165)

[4. METODOLOGÍA PARA LA CONSTRUCCIÓN DEL DATASET 7](#_Toc206867166)

[4.1. Obtención de los datos iniciales 7](#_Toc206867167)

[4.2. Procesado de datos en QGIS 8](#_Toc206867168)

[4.3. Selección de vecinos más cercanos a través de la API de QGIS para Python 14](#_Toc206867169)

[4.3.1. Funciones implementadas en Python 15](#_Toc206867170)

[4.4. Llamadas a la Distance Matrix API mediante script de Python 19](#_Toc206867171)

[4.5 Depurado de los resultados y análisis en QGIS 22](#_Toc206867172)

[5. DESCRIPCIÓN DEL DATASET OBTENIDO 23](#_Toc206867173)

[6. ESPECIFICACIÓN DE REQUISITOS DE LA APLICACIÓN GIS PARA VISUALIZAR LOS DATOS 24](#_Toc206867174)

[7. DISEÑO 25](#_Toc206867175)

[8. RESULTADOS 26](#_Toc206867176)

[9. CONCLUSIONES Y FUTUROS TRABAJOS 27](#_Toc206867177)

[Referencias 28](#_Toc206867178)

**ÍNDICE DE FIGURAS**

[Ilustración 1 Captura de pantalla de la página web NOMECALLES 4](#_Toc206880631)

[Ilustración 2 Vista previa de la herramienta QGIS 5](#_Toc206880632)

[Ilustración 3 Vista previa del código usando la API de QGIS 6](#_Toc206880633)

[Ilustración 4 Proceso de filtrado de núcleos poblacionales 8](#_Toc206880634)

[Ilustración 5 Ajuste de centroide a carretera 9](#_Toc206880635)

[Ilustración 6 Ajuste a parada de bus más cercana 10](#_Toc206880636)

[Ilustración 7 Tabla de hospitales de grupo 2 y 3 11](#_Toc206880637)

[Ilustración 8 Núcleos y hospitales clasificados por su DAS 12](#_Toc206880638)

[Ilustración 9 División por partido judicial 13](#_Toc206880639)

[Ilustración 11 Flowchart de funciones implementadas en Python 16](#_Toc206880640)

[Ilustración 12 Función get\_cords\_name() 17](#_Toc206880641)

[Ilustración 13 Función get\_nearest() 18](#_Toc206880642)

[Ilustración 14 Función get\_jsons() 19](#_Toc206880643)

[Ilustración 15 Ejemplo de archivo de la carpeta jsons/ 20](#_Toc206880644)

[Ilustración 16 Código de google\_request.py 21](#_Toc206880645)

[Ilustración 17 Ejemplo de output generado por una consulta a Distance Matrix API 22](#_Toc206880646)

**ÍNDICE DE CÓDIGO**

[Código 1 Método closest\_destinations\_features() 15](#_Toc206935107)

[Código 2 Método closest\_destinations\_cords() 15](#_Toc206935108)

[Código 3 Método request\_routes\_v2\_async 18](#_Toc206935109)

**LISTA DE ABREVIATURAS**

QGIS Quantum Geographic Information System

DAS Dirección Asistencial de Salud

INE Instituto Nacional de Estadística

**GLOSARIO**

|  |  |
| --- | --- |
| Término | Descripción |
| Núcleo de población | Conjunto de al menos diez edificaciones y/o cincuenta habitantes con calles, plazas u otras vías urbanas. [1] |
| Diseminado | Edificaciones o viviendas que no pueden ser incluidas en el concepto de núcleo de población. [2] |

# INTRODUCCIÓN

## 1.1. Área de Trabajo

El área de trabajo de este estudio son los 179 municipios que componen la Comunidad de Madrid, así como los núcleos poblacionales que lo forman y las distintas infraestructuras donde se brindan servicios públicos esenciales.

La información sobre este dominio de trabajo la tratamos en forma de datasets georreferenciados que nos permiten operar sobre ellos, obtener información adicional y permite la visualización de dichos datos en el mapa mediante aplicaciones como QGIS, ArcGIS

## 1.2. Problema por resolver

Un problema que genera bastante preocupación desde hace muchos años es la falta de servicios públicos en zonas rurales o alejadas de centros de provincia y ciudades importantes. A esto se le suma la falta de datos empíricos y demostrables que validen y cuantifiquen el grado de accesibilidad de los servicios públicos y cuáles son los servicios públicos más necesarios para reforzar en cada zona.

Por lo tanto, el problema que pretendemos abordar es el del desarrollo de una base de datos en la que se pueda ver la disponibilidad de servicios en todas las áreas de la Comunidad de Madrid

## 1.3. Motivación del Trabajo

El fácil acceso y cercanía a servicios públicos es una de las principales necesidades para el bienestar de los habitantes de una ciudad. En un mundo industrializado y con la infraestructura necesaria para conectar todos los núcleos urbanos de un país, es cada vez más fácil y necesario determinar qué servicios públicos brindar y dónde se establece la construcción de la infraestructura destinada a brindar dicho servicio.

Por lo tanto, se tiene como principal objetivo determinar el nivel de necesidad de nuevos centros públicos para cada núcleo urbano y, por lo tanto, de cada municipio.

## 1.4. Objetivos

Este trabajo tiene como meta la obtención de información y la subsecuente creación de una base de datos sobre la disponibilidad de distintos servicios públicos en los diferentes municipios de la Comunidad de Madrid

Los servicios públicos sobre los cuales se hace el trabajo serán:

* Hospitales
* Juzgados
* Centros de salud mental
* Parques de bomberos

Software utilizado: QGIS, software de gestión de bases de datos con representación geográfica de la información.

Fuente de los datos: nomecalles [3]. Base de datos con puntos de interés y distintas delimitaciones de la comunidad de Madrid

# 2. ESTADO DE LA CUESTIÓN

# 3. MÉTODOS

La metodología utilizada en este trabajo se divide en distintas fases, empezando por la recopilación de datos hasta la obtención de las distancias finales y el análisis de resultados.

A grandes rasgos, los pasos realizados son los siguientes:

* Recopilación de los datasets georreferenciados con información sobre núcleos urbanos, delimitación de municipios, zonas de Dirección Asistencial de Salud o la posición de los servicios públicos sobre los que trata el trabajo.
* Procesado de los datos, añadiendo información adicional, estableciendo relaciones entre los distintos datasets, transformándolos o creando nuevos datos para poder ser utilizados a posteriori. Realizado mediante QGIS y Python
* Análisis de los datos, seleccionando para cada núcleo urbano, los servicios públicos más cercanos. Realizado a través de scripts de Python
* Cálculo de distancias de viaje, utilizando scripts de Python que realizan consultas a la API de Google Maps.
* Unión de los resultados obtenidos al dataset original: Los nuevos datos obtenidos son procesados y añadidos como atributos a los datasets, aportando información nueva a los datasets de centros urbanos y de municipios.

## 3.1. Fuente de los datos

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.La fuente principal de los datasets es la página web nomecalles del Instituto de Estadística de la Comunidad de Madrid (IECM). La página contiene el nomenclátor oficial y distintas capas de información territorial con puntos de interés de titularidad pública.

Ilustración 1 Captura de pantalla de la página web NOMECALLES

A través de dicha página se obtuvieron los datasets de delimitaciones de los núcleos urbanos, las ubicaciones de los cuatro servicios públicos a analizar, junto con otros datasets que puedan ser útiles durante alguna otra fase del trabajo.

Otros datos adicionales, pero también imprescindibles en este trabajo se obtienen de otros sitios web como:

* Base de datos del Instituto Nacional de Estadística [4]
* Ministerio de Justicia de España [5]
* Consorcio Regional de Transportes de Madrid [6]

## 3.2. Procesamiento inicial de los datos mediante QGIS

Mapa

El contenido generado por IA puede ser incorrecto.Los datos obtenidos están en un formato legible por la herramienta de software QGIS[6]. Con esta herramienta de software podemos realizar distintas operaciones sobre los datos, añadir atributos con información extra o crear nuevas capas. Un ejemplo de esto es la adición de un atributo que indica, para cada núcleo urbano, la Dirección Asistencial de Salud en la que se encuentra, lo cual será necesario para determinar a qué hospital será referenciado.

Ilustración 2 Vista previa de la herramienta QGIS

## 3.3. Procesamiento adicional mediante Python

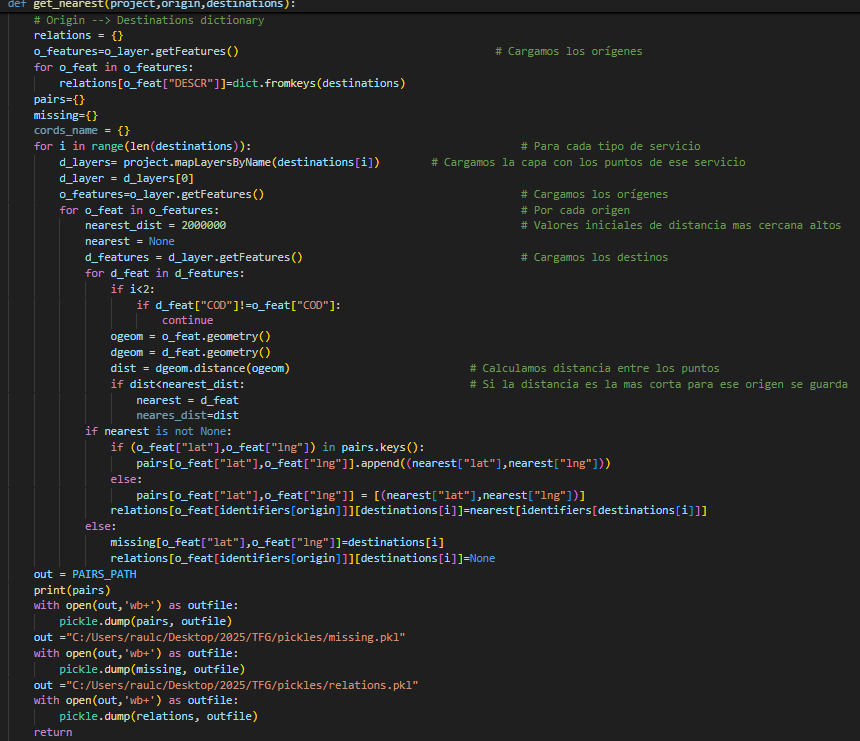
El siguiente paso es determinar, para cada núcleo urbano, el hospital, parque de bomberos, juzgado y centro de salud mental más cercano. En un principio parecía posible hacerlo desde QGIS, pero este proceso en particular es muy tedioso y requiere una enorme cantidad de backtracking en caso de error. Por lo tanto, se decide utilizar la librería de QGIS para Python, la cual nos permite leer y operar sobre capas de

Ilustración 3 Vista previa del código usando la API de QGIS

De esta forma tenemos un método más intuitivo, automatizado y fácil de editar en caso de necesitar un cambio a posteriori, además de permitirnos generar nuevos archivos o realizar consultar en pasos posteriores.

## 3.4. Cálculo de distancias utilizando la API de Google Maps

La Google Maps Platform ofrece acceso a la Routes API [7] a usuarios registrados.

Haciendo uso de esta API se realizan consultas con un formato específico y obtenemos como resultado la distancia y tiempo exactos de un trayecto en coche a los conjuntos [origen, destino] que nosotros especifiquemos.

# 4. METODOLOGÍA PARA LA CONSTRUCCIÓN DEL DATASET

En este punto se explicará en más detalle los pasos realizados, así como los resultados obtenidos durante el proceso.

Cabe aclarar que aunque estas decisiones se presentan de forma linear, algunas se toman en partes más tardías del desarrollo por motivo de backtracking. En estos casos, se aclarará el motivo de las decisiones o se dejará claro que se deben a etapas posteriores del desarrollo

## 4.1. Obtención de los datos iniciales

Nuestro punto de inicio es el sitio web NOMECALLES. Como se describe en el punto 2.1., esta página nos permite visualizar y descargar distintos datasets georreferenciados de la Comunidad de Madrid, estos se dividen en Puntos y Delimitaciones, el primero ofrece datasets de puntos exactos y el segundo ofrece datasets de distintas divisiones territoriales.

De todos los conjuntos de datos que ofrece la página extraemos los siguientes:

* Delimitación de municipios: Delimitaciones 🡪 Municipios
* Núcleos urbanos: Delimitaciones 🡪 Nomenclátor 🡪 Núcleos
* Dirección Asistencial de Salud (DAS): Delimitaciones🡪Zonificación de Salud de Área Única🡪Dirección Asistencial de Salud
* Juzgados: Puntos🡪Admón. Pública y Seguridad Ciudadana🡪Instituciones Públicas🡪Admón. Justicia
* Centros de Salud Mental: Puntos🡪 Salud y Servicios Sociales🡪 Centros Sanitarios🡪 Centros de Salud Mental
* Hospitales: Puntos🡪Salud y Servicios Sociales🡪 Centros Sanitarios🡪Hospital
* Parques de bomberos: Puntos🡪Admón. Pública y Seguridad Ciudadana🡪 Protección y Seguridad Ciudadana🡪Bomberos

Algunos datasets georreferenciados fueron obtenidos de otras fuentes al no aparecer en el nomenclátor de NOMECALLES:

* Paradas de autobús EMT/Urbano/Interurbano: Obtenidas de la página del Consorcio de Transportes de Madrid [8]. Este dataset es necesario para obtener distancias utilizando transporte público
* Red de carreteras de Madrid: Versión modificada de dataset obtenido en el Centro de Descargas del Instituto Geográfico Nacional [9]. Este dataset es necesario para poder aproximar centroides a la carretera más cercana

Datos adicionales en formato tabla/CSV/XLSX

* Población por unidad poblacional: Obtenidas en el buscador del Insituto Nacional de Estadística [4]

## 4.2. Procesado de datos en QGIS

Una vez tenemos todos los datasets necesarios cargados en QGIS, podemos comenzar el proceso de transformación de los datos obtenidos.

### Filtrado de diseminados y núcleos con menos de 10 habitantes

El primer paso a realizar es el filtrado del dataset de núcleos urbanos. En dicho conjunto de datos aparecen tanto núcleos de población como diseminados. El propio dataset contiene información sobre qué delimitación entra en cada una de estas dos categorías, por lo tanto, creamos una nueva capa que excluya todos los diseminados del conjunto.

A continuación, utilizando datos del INE sobre población por unidad poblacional [4], pasamos el dataset por otro filtrado, dejando fuera todos los núcleos poblacionales con menos de 10 habitantes.

Esta operación hace que pasemos de 1208 a 468 núcleos poblacionales sobre los que llevaremos a cabo nuestras operaciones

Imagen que contiene Mapa

El contenido generado por IA puede ser incorrecto.

Ilustración 4 Proceso de filtrado de núcleos poblacionales

### Obtención de los centroides

Como se puede observar, este dataset contiene datos georreferenciados de cada núcleo poblacional en forma de delimitaciones geográficas. Esta representación es importante pero no nos sirve para las operaciones que llevar a cabo. Lo que sí podemos utilizar es el centroide de cada uno de estos núcleos poblacionales, los cuales conservan además toda su información original: Nombre del núcleo, municipio al que pertenecen, población, etc.

Originalmente se planteó el uso de la herramienta Centroides, pero rápidamente se puede llegar a la conclusión de que muchos núcleos poblacionales tienen formas no convexas, haciendo que el centroide caiga fuera del núcleo.

Para evitar futuras complicaciones elegimos utilizar la herramienta Punto en superficie, la cual devuelve una dataset de puntos que, aunque no sean centroides de los núcleos poblacionales, está garantizado que se encuentra dentro de la delimitación de este.

En pasos posteriores se tuvo que hacer backtracking a este punto para añadir varios cambios al dataset que se necesitan para pasos posteriores:

* Mover cada centroide a la carretera más cercana, siempre y cuando esa carretera se encuentre dentro del mismo núcleo poblacional.
  + Para esta tarea se hace uso de un plugin de QGIS llamado ProcessX, el cual permite ajustar puntos a una línea añadiendo condiciones, en este caso, el de que pertenezca al mismo núcleo poblacional, evitando que se ajuste a carreteras de otro núcleo.

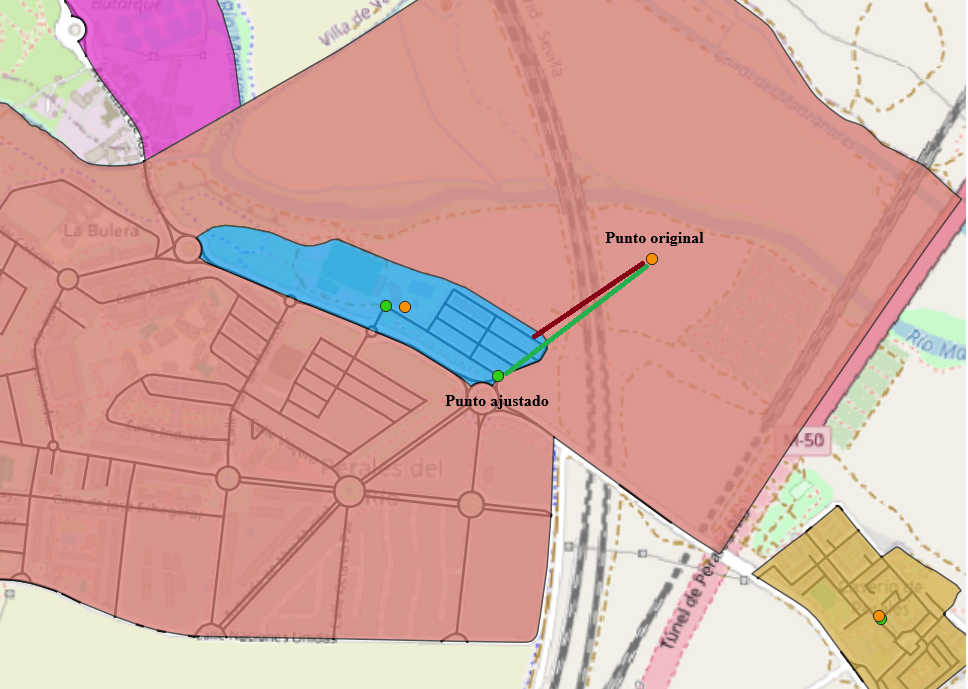


Ilustración 5 Ajuste de centroide a carretera

* Mover a mano ciertos núcleos poblacionales: Alrededor de 10 núcleos poblacionales se encuentran prácticamente incomunicados en lo que respecta a transporte público. En pasos posteriores, utilizaremos una API que devuelve distancias por transporte público, esta API tiene un grado de tolerancia limitado y puede calcular el tiempo a pie a la parada más cercana, pero si la distancia es excesiva deja de devolver resultados. Por lo tanto, movemos a mano estos centroides a la parada de bus más cercana, llevando un registro para tener en cuento este ajuste en pasos posteriores.

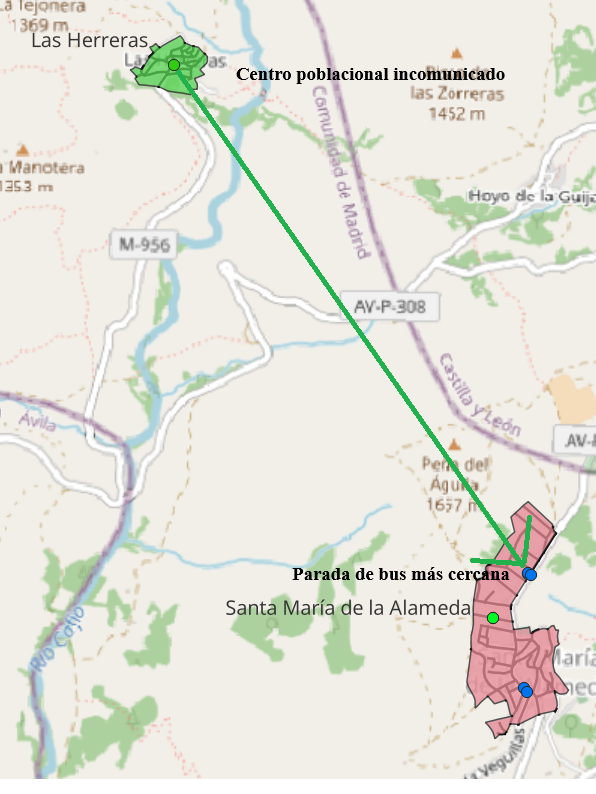


Ilustración 6 Ajuste a parada de bus más cercana

**Filtrado de hospitales**

El dataset de hospitales proporcionado por NOMECALLES contiene centros hospitalarios con enorme disparidad de tamaño y complejidad. Por lo tanto, no podemos considerar de la misma manera un núcleo de población cuyo hospital más cercano es un hospital de la complejidad más baja y un núcleo de población a la misma distancia del hospital Gregorio Marañón.

Por esto mismo, dividimos el dataset en otros dos nuevos conjuntos de datos, esta división la hacemos acorde a la complejidad de los hospitales y, como aparece reflejada en el dataset de NOMECALLES, se hace de forma manual guiándonos por la información sobre centros hospitalarios de la Red del Servicio Madrileño de Salud de la página web de la Comunidad de Madrid [10].

Los dos datasets resultantes son:

* Hospitales de grupo 3: Hospitales de gran complejidad.
* Hospitales de grupo 2: Hospitales de complejidad intermedia

En la siguiente tabla se puede ver la lista de hospitales que pasaron el filtro y que, por lo tanto, serán utilizados en los siguientes pasos:

|  |  |  |
| --- | --- | --- |
| Nombre | Grupo | Zona |
| Hospital Universitario 12 de Octubre | 3 | Centro |
| Hospital Universitario de la Princesa | 3 | Centro |
| Hospital Universitario Fundación Jiménez Díaz | 3 | Noroeste |
| Hospital Clínico San Carlos | 3 | Noroeste |
| Hospital Universitario Puerta de Hierro Majadahonda | 3 | Noroeste |
| Hospital Universitario La Paz | 3 | Norte |
| Hospital Universitario Ramón y Cajal | 3 | Norte |
| Hospital General Universitario Gregorio Marañón | 3 | Sureste |
| Hospital Centro Sanitario de Vida y Esperanza | 2 | Centro |
| Hospital Universitario Príncipe de Asturias | 2 | Este |
| Hospital Universitario de Torrejón | 2 | Este |
| Hospital General de Villalba | 2 | Noroeste |
| Hospital Universitario Infanta Sofía | 2 | Norte |
| Hospital Universitario de Móstoles | 2 | Oeste |
| Hospital Universitario Fundación Alcorcón | 2 | Oeste |
| Hospital Universitario Fuenlabrada | 2 | Oeste |
| Hospital Rey Juan Carlos | 2 | Oeste |
| Hospital Universitario Severo Ochoa | 2 | Sur |
| Hospital Universitario de Getafe | 2 | Sur |
| Hospital Universitario Infanta Leonor | 2 | Sureste |

Ilustración 7 Tabla de hospitales de grupo 2 y 3

### Adición de atributos

El dataset en este momento contiene todos los destinos (hospitales, juzgados, parques de bomberos y centros de salud mental), los municipios y los núcleos poblacionales con sus respectivos centroides ajustados. Pero aún se necesitan varios atributos que faciliten los pasos posteriores, además de brindar información adicional que puede ser relevante.

El primer atributo adicional es la **Dirección Asistencial de Salud (DAS)**. En Madrid existen 7 DAS y estas delimitaciones determinan cuál es el hospital de referencia de cada núcleo poblacional, esto evita que se calcule la distancia de un núcleo poblacional a un hospital de una DAS distinta.

Cada DAS tiene, por lo menos, un hospital de grupo 2, por lo que el hecho de la existencia de un hospital de grupo 3 dentro de la misma DAS se considera como un punto a favor al calcular la disponibilidad de un núcleo poblacional.

Cada DAS se representa por un atributo **COD**, con un número del 1 al 7. La herramienta Unir atributos por localización permite que un punto/delimitación de una capa herede el atributo de la delimitación de otra capa siempre que lo interseque o lo contenga. En la Ilustración 8 se puede ver la delimitación de cada DAS y cómo los núcleos y hospitales dentro de cada uno conserva el atributo COD de dicho DAS.

Mapa

El contenido generado por IA puede ser incorrecto.

Ilustración 8 Núcleos y hospitales clasificados por su DAS

El segundo atributo adicional es el **partido judicial**. **S**egún el Artículo 32 de la Ley Orgánica del Poder judicial [11], el partido judicial es una división de territorios uno o más municipios limítrofes, pertenecientes a una misma provincia. Esta división la utilizaremos para que un núcleo urbano no sea dirigido a un juzgado de un partido judicial distinto.

Por lo tanto, se añade un atributo que determine a cuál de los 21 partidos judiciales pertenece cada núcleo de población y cada juzgado, acorde a los listados proporcionados por el Ministerio de Justicia [5] [12]. En la Ilustración 9 se representa una clasificación de núcleos poblacionales (círculos) y juzgados (cuadrados) por el partido judicial al que pertenecen.

Mapa

El contenido generado por IA puede ser incorrecto.

Ilustración 9 División por partido judicial

Atributos **lat** y **lng**. Contienen la latitud y longitud de cada punto. Estos atributos se pueden obtener en este punto del trabajo utilizando la herramienta Añadir campos X/Y a capa, pero se puede omitir completamente ya que utilizando la librería de QGIS para Python se puede obtener de forma rápida.

## 4.3. Selección de vecinos más cercanos a través de la librería de QGIS para Python

QGIS ofrece su propia librería de Python para realizar operaciones sobre capas de archivos .qgz, que contienen el proyecto QGIS completo.

El primer archivo Python utilizado se llama file\_creator.py y contiene todos los métodos que permiten operar con archivos o leer el prioyecto QGIS, permitiéndonos obtener los datos de todas las capas y operar sobre ellas.

### Métodos closest\_destinations\_cords() y closest\_destinations\_features()

Esta función toma solo dos parámetros:

* origin: String con el nombre de la capa de origen, es decir, de los centroides de núcleos poblacionales.
* destinations: Lista con los nombres de las capas de hospitales, parques de bomberos, juzgados y centros de salud mental.

Antes de la ejecución de la función se definen varias variables.

project es, de forma resumida el proyecto QGIS donde se encuentran todas las capas.

El funcionamiento del algoritmo es, traducida a lenguaje natural:

1. Crea una lista con todos los orígenes y una lista para cada tipo de destino
2. Por cada núcleo itera por todos los destinos, y dependiendo del tipo de destino:
   * + - Hospitales: Guarda las coordenadas de todos los hospitales de grupo 2 y 3 por separado que pertenezcan al mismo DAS.
       - Juzgados: Filtra todos los juzgados que no estén en el mismo partido judicial y luego calcula la distancia euclídea a cada uno de ellos, guardando las coordenadas del juzgado más cercano.
       - Parques de bombero: No hace filtrado, por lo que calcula todas la distancia desde el núcleo a todos los parques de bomberos y guarda las coordenadas del más cercano.
       - Centros de salud mental: Filtra para cada núcleo los CSM del mismo DAS y calcula la distancia para quedarse con las coordenadas del más cercano
3. Devuelve un diccionario de Python que, para cada núcleo poblacional, contiene las coordenadas del propio núcleo y las coordenadas de los destinos.

def closest\_destinations\_cords(origin: str, destinations: list[str]) -> dict[str,dict[str,list[str]]]:

"""

Lee el proyecto QGIS y devuelve un diccionario

Para cada origen guarda sus coordenadas y las coordenadas de los destinos más cercanos

"""

closest = {}

origin\_layer = project.*mapLayersByName*(origin)[0]

origin\_features = list(origin\_layer.*getFeatures*())

destination\_layers = [project.*mapLayersByName*(dest)[0] for dest in destinations]

destination\_features = [list(layer.*getFeatures*()) for layer in destination\_layers]

crs\_src = origin\_layer.*crs*()

crs\_dest = qgis.*QgsCoordinateReferenceSystem*("EPSG:4326")

transform = qgis.*QgsCoordinateTransform*(crs\_src, crs\_dest, qgis.*QgsProject*.*instance*())

for origin\_feat in origin\_features:

origin\_id = origin\_feat[identifiers[origin\_layer.*name*()]]

origin\_point = origin\_feat.*geometry*().*asPoint*()

origin\_ll = transform.*transform*(origin\_point)

origin\_cords = [origin\_ll.*y*(),origin\_ll.*x*()]

closest[origin\_id]={"cords":origin\_cords, "destinations":[]}

for i, dest\_layer\_feats in enumerate(destination\_features):

dest\_name = destinations[i]

filtered\_dest\_feats = []

# Filtro

if dest\_name in ["Hospitales grupo 2", "Hospitales grupo 3", "Salud mental"]:

if 'COD' in origin\_feat.*fields*().*names*():

origin\_cod = origin\_feat['COD']

filtered\_dest\_feats = [f for f in dest\_layer\_feats if f['COD'] == origin\_cod]

elif dest\_name == "Juzgados":

if 'PART\_JUD' in origin\_feat.*fields*().*names*():

origin\_part\_jud = origin\_feat['PART\_JUD']

filtered\_dest\_feats = [f for f in dest\_layer\_feats if f['PART\_JUD'] == origin\_part\_jud]

else:

filtered\_dest\_feats = dest\_layer\_feats

if dest\_name in ["Hospitales grupo 2", "Hospitales grupo 3"]:

added = [[transform.*transform*(f.*geometry*().*asPoint*()).*y*(),

transform.*transform*(f.*geometry*().*asPoint*()).*x*()]

for f in filtered\_dest\_feats]

closest[origin\_id]["destinations"].*append*(added)

continue

# Busca el destino más cercano

min\_dist = float('inf')

min\_feat = None

for dest\_feat in filtered\_dest\_feats:

dist = qgis.*QgsDistanceArea*().*measureLine*(

origin\_feat.*geometry*().*centroid*().*asPoint*(),

dest\_feat.*geometry*().*centroid*().*asPoint*()

)

if dist < min\_dist:

min\_dist = dist

min\_feat = dest\_feat

# Guardamos sus identificadores

pt = transform.*transform*(min\_feat.*geometry*().*asPoint*())

closest[origin\_id]["destinations"].*append*([pt.*y*(), pt.*x*()])

return closest

El método closest\_destinations\_features() hace exactamente lo mismo pero el diccionario guarda los nombres de los destinos y orígenes. Es útil principalmente a la hora de comprobar que funciona correctamente

Código 1 Método closest\_destinations\_features()

# Guardamos sus identificadores

closest\_feats**.***append***(**min\_feat**[**identifiers**[**dest\_name**]]** **if** min\_feat **else** **None)**

closest**[**origin\_id**]** **=** closest\_feats

**return** closest

Código 2 Método closest\_destinations\_cords()

Texto

El contenido generado por IA puede ser incorrecto.

Ilustración 10 Fragmento del output de closest\_dest\_cords()

Texto

El contenido generado por IA puede ser incorrecto.

Ilustración 11 Fragmento del output de closest\_dest\_features()

## 4.4. Consulta a la Routes API de Google

Una vez tenemos la lista completa con cada uno de los núcleos poblacionales relevantes para el estudio, planteamos un plan de consultas a la Routes API de Google Maps Platform [8], esta API devuelve, entre otras cosas, la distancia real entre dos puntos.

Para obtener un resultado más preciso, para cada par origen-destino, pedimos que devuelva la distancia y tiempo de recorrido con estos parámetros:

* En hora punta (8:30) en coche
* En hora punta en transporte público
* En hora con menos tráfico (11:00) en coche
* En hora con menos tráfico en transporte público

Con estos cuatro resultados nos aseguramos de que se tienen en cuenta distintas situaciones que también representan la disponibilidad de un servicio público.

### Métodos para la realización de consultas

En el archivo google\_request.py se encuentra cualquier método relacionado con consultas a la API. Después de varias iteraciones de código llegamos a la conclusión de que ejecutar las consultas de forma asíncrona, dado que el número total de consultas a realizar se acerca a 12.000 y en algunos casos la API fallaba o había problemas de comunicación.

Los dos métodos principales del programa son request\_routes\_v2\_async() y fetch\_route().

request\_routes\_v2\_async() toma un solo parámetro booleano “preserve”, el cual, si es True, indica si sólo se realizan las consultas de las cuales no se tienen resultados todavía, permitiendo que un fallo de conexión puntual no nos obligue a realizar todas las consultas de nuevo. El resto del método hace llamadas asíncronas a la función fetch\_route() y almacena y ordena los resultados de dicha función en forma de diccionario para ser utilizado a posteriori.

fetch\_route(), en cambio, es el método que lleva a cabo las consultas, prepara la cabecera y cuerpo de cada consulta, notifica en caso de recibir una respuesta correcta o un mensaje de error y devuelve dicho resultado, así como el ID del origen, el ID del destino, el modo (en coche o transporte público) y la hora de la consulta, todo necesario para ensamblar los resultados en la función request\_routes\_v2\_async().

**async** **def** request\_routes\_v2\_async**(**preserve**=False):**

path **=** Path**(**"output/closest\_destinations\_cords.json"**)**

**with** **open(**path**,** 'r'**)** **as** file**:**

json\_file **=** json**.***load***(**file**)**

**if** preserve**:**

**with** **open(**"output/routes\_API\_results\_dump.json"**,** 'r'**)** **as** file**:**

full\_result **=** json**.***load***(**file**)**

**else:**

full\_result **=** **{}**

tasks **=** **[]**

**for** origin **in** json\_file**:**

**if** **not** preserve**:**

full\_result**[**origin**]** **=** **[]**

origin\_cords **=** json\_file**[**origin**][**"cords"**]**

destinations **=** **[]**

**for** item **in** json\_file**[**origin**][**"destinations"**]:**

**if** item**:**

**if** **isinstance(**item**[**0**],** **list):**

destinations**.***extend***(**item**)**

**else:**

destinations**.***append***(**item**)**

**for** dest\_index**,** destination **in** **enumerate(**destinations**):**

**for** mode **in** modes**:**

**for** hour\_index**,**hour **in** **enumerate(**hours**):**

**if** preserve**:**

**if** **not** full\_result**[**origin**][**dest\_index**][**f"{mode}\_{hours\_true**[**hour\_index**].***strftime***(**"%H:%M"**)**}"**]:**

tasks**.***append***(**

fetch\_route**(**origin\_cords**,** destination**,** mode**,** hour**,** origin**,** dest\_index**)**

**)**

**else:**

tasks**.***append***(**

fetch\_route**(**origin\_cords**,** destination**,** mode**,** hour**,** origin**,** dest\_index**)**

**)**

results **=** **await** asyncio**.***gather***(\***tasks**)**

# Reconstruir el resultado en estructura original

**for** origin\_id**,** dest\_index**,** mode**,** hour**,** data **in** results**:**

hour\_key **=** hours\_true**[**hour\_index**].***strftime***(**'%H:%M'**)**

key **=** f"{mode}\_{hour\_key}"

**if** **len(**full\_result**[**origin\_id**])** **<=** dest\_index**:**

full\_result**[**origin\_id**].***extend***([{}]** **\*** **(**dest\_index **-** **len(**full\_result**[**origin\_id**])** **+** 1**))**

full\_result**[**origin\_id**][**dest\_index**][**key**]** **=** data

**return** full\_result

Código 3 Método request\_routes\_v2\_async

**async** **def** fetch\_route**(**origin\_cords**,** destination**,** mode**,** hour**,** origin\_id**,** dest\_index**):**

**async** **with** semaphore**:**

body **=** **{**

"origin"**:** **{**"location"**:** **{**"latLng"**:** **{**"latitude"**:** origin\_cords**[**0**],** "longitude"**:** origin\_cords**[**1**]}}},**

"destination"**:** **{**"location"**:** **{**"latLng"**:** **{**"latitude"**:** destination**[**0**],** "longitude"**:** destination**[**1**]}}},**

"travelMode"**:** mode**.***upper***(),**

"departureTime"**:** hour**.***strftime***(**"%Y-%m-%dT%H:%M:%SZ"**)**

**}**

**if** mode **==** "drive"**:**

body**[**"routingPreference"**]** **=** "TRAFFIC\_AWARE"

**for** attempt **in** **range(**3**):** # Reintenta hasta 3 veces

**try:**

**async** **with** httpx**.***AsyncClient***(**timeout**=**30**)** **as** client**:**

**print(**f"Enviando {origin\_id}, Destino {dest\_index} [{mode} {hour**.***strftime***(**'%H:%M'**)**}]"**)**

response **=** **await** client**.***post***(**URL**,** headers**=**HEADERS**,** json**=**body**)**

response**.***raise\_for\_status***()**

**print(**f"Recibido: {origin\_id}, Destino {dest\_index} [{mode} {hour**.***strftime***(**'%H:%M'**)**}]"**)**

**return** **(**origin\_id**,** dest\_index**,** mode**,** hour**,** response**.***json***())**

**except** httpx**.***HTTPStatusError* **as** e**:**

**print(**f"HTTP {e**.***response***.***status\_code*} en {origin\_id}-{dest\_index}-{mode}, intento {attempt**+**1}"**)**

**await** asyncio**.***sleep***(**1**)**

**except** **Exception** **as** e**:**

**print(**f"Error en {origin\_id}-{dest\_index}-{mode}: {e}"**)**

**await** asyncio**.***sleep***(**2**)**

**return** **(**origin\_id**,** dest\_index**,** mode**,** hour**,** **{**"error"**:** "intentos agotados"**})**

Código 4 Método fetch\_route()

El resultado obtenido es un diccionario que fácilmente se puede convertir en archivo JSON con la siguiente estructura:

{

"0790101": [

{

"drive\_08:30": {

"routes": [

{

"distanceMeters": 1929,

"duration": "464s"

}

]

}, 🡨 Aquí termina el resultado en coche a las 8:30

…

}, 🡨 Aquí termina el primer destino

…

}

## 4.5 Depurado de los resultados y análisis en QGIS

Una vez se tienen los resultados en formato .json hace falta aplicarle varias capas de modificaciones para obtener lo que consideraríamos el dataset completo.

En el punto 4.2 mencionamos que ciertos núcleos tienen que ser movidos a una parada de bus cercana para poder obtener resultados en las consultas a Routes API. Pues ahora es el momento en el que es necesario tener en cuenta qué núcleos fueron desplazados y cuánto desplazamiento se le aplicó.

El desplazamiento se aplicó en una copia de la capa original, así que, teniendo ambas capas podemos calcular la distancia entre el mismo punto en estas dos capas obteniendo una distancia euclídea. Esto lo obtenemos fácilmente con las funciones get\_penalization() y apply\_penalization()

El funcionamiento general de estas funciones es: get\_penalization() guarda todos los núcleos desplazados y su desplazamiento en distancia euclídea, apply\_penalizations() toma esas penalizaciones y las aplica a todos los núcleos afectados.

Esta penalización es una aproximación y simula el camino andando entre la parada de bus y el destino, dado que la distancia euclídea no es una distancia realista, aproximamos que el camino real es aproximadamente un 150% de la distancia euclídea, ya que no es un camino recto, por lo que en el caso de la distancia, añadimos la penalización\*1.5, mientras que para la duración del trayecto, ya que la velocidad media de una persona andando es de 1.4 m/s y asumimos que el trayecto es 1.5 veces la distancia euclídea, añadimos la penalización\*1.5/1.4, o sea penalización\*~1.07.

**def** get\_penalization**(**old**,** new**):**

old\_layer **=** project**.***mapLayersByName***(**old**)[**0**]**

old\_features **=** **list(**old\_layer**.***getFeatures***())**

new\_layer **=** project**.***mapLayersByName***(**new**)[**0**]**

new\_features **=** **list(**new\_layer**.***getFeatures***())**

penalizations **=** **{}**

**for** old\_feature **in** old\_features**:**

**for** new\_feature **in** new\_features**:**

**if** old\_feature**[**"CDTNUCLEO"**]==**new\_feature**[**"CDTNUCLEO"**]:**

dist **=** qgis**.***QgsDistanceArea***().***measureLine***(**

old\_feature**.***geometry***().***centroid***().***asPoint***(),**

new\_feature**.***geometry***().***centroid***().***asPoint***()**

**)**

**if** dist**>**0**:**

penalizations**[**old\_feature**[**"CDTNUCLEO"**]]** **=** dist

**continue**

**return** penalizations

**def** apply\_penalization**(**objective**,**pen\_json**,):**

penalizations **=** json\_to\_dict**(**pen\_json**)**

results **=** json\_to\_dict**(**objective**)**

**for** centro\_key **,** penalization **in** penalizations**.***items***():**

**for** result\_key**,** result **in** results**[**centro\_key**].***items***():**

**if** "transit" **in** result\_key**:**

**if** "dis" **in** result\_key**:**

results**[**centro\_key**][**result\_key**]=** result **+int(**penalization**\***1.5**)**

**if** "dur" **in** result\_key**:**

results**[**centro\_key**][**result\_key**]=** result **+int(**penalization**\***1.07**)**

**return** results

Código 5 Métodos get\_penalization() y apply\_penalization()

# 5. DESCRIPCIÓN DEL DATASET OBTENIDO

Metadatos

Indicadores Obtenidos

Formatos de publicación

# 6. ESPECIFICACIÓN DE REQUISITOS DE LA APLICACIÓN GIS PARA VISUALIZAR LOS DATOS

6.1 Presentación de los formalismos para registrar las Requisitos.

6.2 Especificación de Requisitos

6.3 Plan de Pruebas

# 7. DISEÑO

7.1 Diseño de la IU (Aplicación de Prueba)

7.1.1. Paper prototype o mock up inicial de la aplicación

7.1.2. Versión final de la UI

7.1.3 Diagramas de clases (si aplica)

7.2 Diseño de la arquitectura de servidor (Back end).

7.2.1 Alternativas (definir varias posibles alternativas y seleccionar la que más te convengan).

7.2.2. Diagrama de paquetes o componentes y clases de los elementos de la arquitectura de servidor.

# 8. RESULTADOS

Especificación de preguntas de análisis de la accesibilidad de los servicios públicos en la Comunidad de Madrid y su discusión con los datos estadísticos proporcionados por el dataset generado.

# 9. CONCLUSIONES Y FUTUROS TRABAJOS

# Referencias

|  |  |
| --- | --- |
| [1] | Instituto Nacional de Estadística, «Núcleo de población,» [En línea]. Available: https://www.ine.es/DEFIne/es/concepto.htm?c=4930. [Último acceso: 2025]. |
| [2] | Instituto Nacional de Estadística, «Diseminado,» [En línea]. Available: https://www.ine.es/DEFIne/concepto.htm?c=5188. |
| [3] | Instituto de Estadística de la Comunidad de Madrid, «Nomecalles,» [En línea]. Available: https://gestiona.comunidad.madrid/nomecalles\_web/#/inicio. |
| [4] | Instituto Nacional de Estadística, «Nomenclátor. Población por unidad poblacional,» [En línea]. Available: https://www.ine.es/nomen2/. |
| [5] | Ministerio de justicia de España, «Nomenclátor de Partidos Judiciales de España,» [En línea]. Available: https://www.mjusticia.gob.es/es/JusticiaEspana/OrganizacionJusticia/Documents/011223\_Nomencl%C3%A1tor%20de%20Partidos%20Judiciales.pdf. |
| [6] | QGIS.org, «Spatial wihtout Compromise · QGIS Website,» QGIS Association, 2025. [En línea]. Available: http://www.qgis.org. |
| [7] | Google Maps Platform, [En línea]. Available: https://developers.google.com/maps/documentation/distance-matrix/overview?hl=es-419&authuser=1. [Último acceso: 2025]. |
| [8] | «Datos Abiertos del Consorcio Regional de Transportes de Madrid,» 2025. [En línea]. Available: https://data-crtm.opendata.arcgis.com/search. |
| [9] | «Centro de Descargas del CNIG (IGN),» [En línea]. Available: https://centrodedescargas.cnig.es/CentroDescargas/redes-transporte. [Último acceso: 2025]. |
| [10] | Comunidad de Madrid, «Hospitales de la red del Servicio Madrileño de Salud,» [En línea]. Available: https://www.comunidad.madrid/servicios/salud/hospitales-red-servicio-madrileno-salud. |
| [11] | Boletín Oficial del Estado, «Articulo 32 Ley Orgánica del Poder judicial,» [En línea]. Available: https://www.iberley.es/legislacion/articulo-32-ley-organica-poder-judicial. |
| [12] | Procuradora Asociados, «Buscador de PARTIDOS JUDICIALES EN Madrid,» [En línea]. Available: https://procuradoresasociadosmadrid.es/partidos-judiciales-madrid/. |