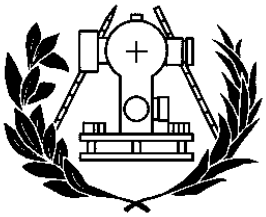
GENERACIÓN DE RUTAS PEATONALES PERSONALIZADAS CON CRITERIOS ADICIONALES A LA DISTANCIA

Autor: Rocío Jiménez Díaz

Directora: Laura Sebastiá Tarín, Dpto. Sistemas Informáticos y Computación de la Universitat Politècnica de València.

Trabajo de Fin de Máster, 2º Curso del Máster en Ingeniería Geomática y Geoinformación



ESCUELA TÉCNICA SUPERIOR
DE INGENIERÍA GEODÉSICA
CARTOGRÁFICA Y TOPOGRÁFICA
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Agradecimientos

Agradezco a mi familia y amigos, que, aunque estén lejos, han sido un apoyo fundamental en estos momentos tan complicados y rocambolescos que nos está tocando vivir. Especialmente a Alejandra y Rober, por haberme aguantado incluso cuando ni yo lo hacía y haberme encendido la bombilla cuando no veía la luz. A Adina, por sacarme de casa cuando no quería salir y apoyarme a través de una pantalla. A Héctor, por haber tolerado mi histeria con maestría. Y a mis compañeros y amigos Walter, Julián y Sonia, porque las redes de apoyo son fundamentales en los momentos importantes; a Alex, que se marchó antes pero siempre ha estado ahí.

A mi tutora, Laura, por su infinita paciencia y toda su ayuda.

Compromiso

"El presente trabajo ha sido realizado completamente por el firmante; no ha sido entregado como otro trabajo académico previo y todo el material tomado de otras fuentes ha sido convenientemente entrecomillado y citado su origen en el texto, así como referenciado en la bibliografía".

Resumen

La generación de rutas resulta fundamental en el mundo que hoy en día nos rodea. Dependemos directamente de las nuevas tecnologías, y pretendemos que estas se adapten a todas nuestras necesidades. Pese a que hoy en día la distancia y el tiempo son factores muy importantes a tener en cuenta con respecto a nuestros desplazamientos, no siempre serán los más importantes; no es lo mismo recorrer una ciudad para ir a trabajar a la oficina que hacerlo para descubrir los rincones que más nos interesen.

Como los intereses son variados y dependen intrínsecamente de la personalidad de cada individuo, se han determinado cuatro variables esenciales en este trabajo para poder realizar el trazado de una ruta que se adapte al momento en que la solicitamos, a saber, el ruido que existe en cada tramo a recorrer, en búsqueda de la ruta más tranquila, a sabiendas de que el estrés es un factor muy importante a evitar; las zonas verdes, que estarán relacionadas con aquellas calles y zonas de menor tráfico y por tanto de más bajo nivel de ruido, en busca de hallar la zona más verde por la que dar un paseo; zonas culturales, si nos interesa ver los edificios artísticos, las obras de arte callejeras o los museos existentes en la ciudad de destino, que bien puede ser la nuestra propia y, finalmente, zonas sociales, por si lo que queremos es una ruta gastronómica para poder probar la restauración del área donde queremos trazar la ruta.

Así, se obtendría un índice final en base a estas cuatro características, ponderadas dependiendo lo que se prime en cada elección de ruta: el índice de *agradabilidad* de ese trazado, de ese camino, que será directamente proporcional a lo que nosotros hayamos estipulado como primordial en el análisis. La ruta se basará en el trazado a pie, no se tendrán en cuenta los vehículos para realizarla, y se primará también la distancia, puesto que, aunque queramos que la ruta sea más agradable, tampoco es recomendable bordear toda una ciudad para conseguir llegar a nuestro destino.

Finalmente, se compararán las rutas obtenidas para ver cuál es el desvío que cada una nos proporciona en base a las características que hayamos estipulado como más importantes, y se realizará también una comparación con la ruta base, que es el resultado de simplemente aplicar el criterio de distancia, sin que pesen las otras variables.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN Y OBJETIVO

El objetivo principal de este trabajo es la generación de rutas alternativas de desplazamiento entre dos puntos, en base a variables que habitualmente no son tenidas en cuenta para los mismos por extender esta ruta, bien aumentando su tiempo de ejecución o el camino recorrido.

Es importante tener en cuenta que los desplazamientos que realizamos no se basan siempre en las mismas preferencias; por ejemplo, cuando estamos yendo al trabajo, o a un lugar de destino al que tengamos que llegar a tiempo, primará la ruta más corta o bien la más directa. Asimismo, si nos encontramos en un descanso vacacional, o conociendo una ciudad o país nuevo, podríamos primar otros factores como, por ejemplo, la existencia de zonas verdes, lugares paisajísticos de mayor belleza, calles con locales sociales tales como cafeterías, bares, restaurantes o cualquier otro tipo de comercios, zonas turísticas o de interés arquitectónico…

En nuestro caso, la hipotética ruta que queremos trazar se basa en la *agradabilidad* del camino que vayamos a recorrer. Así, podremos dotar de mayor o menor peso a aquellas variables existentes que nos resulten más interesantes.

ESTADO DE LA CUESTIÓN

Actualmente, hay diversos países en los que se ha implantado este tipo de diseño de rutas alternativas, o se han intentado implantar. Los países que tienen más desarrollado este tipo de aplicativo son Finlandia y Alemania, donde podemos encontrar una amplia selección de rutas a disposición del usuario final, para que dé importancia al tipo de camino que quiera seguir según el momento en que se encuentre. Así, algunos de estos estudios y desarrollos diferencian las rutas según el momento del día en que se encuentran[1], pero también los hay que se centran en rutas con menor contaminación, sea acústica o aérea, así como rutas más orientadas a algunos tramos de edad, como los niños[2], o tipos de ruta que no se acomoden a peatones, sino a vehículos sin tracción motora, como bicicletas[3].

Por otro lado, en España tenemos el ejemplo de la utilidad de la aplicación *Shadowmap*, una aplicación que genera en tiempo real una reproducción de las zonas de luz y de sombras que hay en las calles de la zona. Así, en nuestro país puede resultar muy útil[4] para, por ejemplo, evitar las calles con mayor exposición al sol durante las peores horas del día, mientras que en países del norte europeo puede servir para todo lo contrario. De hecho, la aplicación fue diseñada para "ayudar a los humanos a encontrar el sol"[5], sobre todo en la búsqueda de un nuevo hogar para comprobar las horas de sol que puede tener esta casa.

En general, este tipo de trabajos previos consultados se basan en la obtención de datos mediante Open Street Map[6], aunque algunos de los proyectos ya creados o en proceso de creación se basan en la obtención de datos por otros medios, como, por ejemplo, Google Maps o los propios portales de datos abiertos de cada zona. Dependiendo de los tipos de datos que necesitemos, tendremos que recurrir a fuentes alternativas y complementarias; de cualquier manera, cada ciudad, población o zona cuenta con datos abiertos diferentes, lo que dificulta un aplicativo estandarizado salvo, claro está, por el uso de OSM.

En los documentos revisados para conocer el estado de la cuestión se han visto numerosas alternativas de variables elegidas para trazar rutas diferentes o complementarias, dependiendo de la zona y de las necesidades de cada estudio y cada población. De esta manera, por ejemplo, encontramos la posibilidad de creación de rutas seguras[7], con puntos de referencia para poder reducir el posible estrés o ansiedad del viandante, rutas basadas en factores medioambientales, como evitar presencia de contaminación acústica o del aire[8], o incluso lumínica (Moelter y Lindley, 2015). También rutas que se basen en la accesibilidad de personas con movilidad reducida.

---

[1] Lo que se conoce como *safety routes*, o rutas de seguridad, sobre todo en Alemania (Siriaraya et al., 2020).
[2] *Less Traffic Noise Polluted Enviroments*, acorde al estudio de Wang et al. (2020).
[3] Poom et al., para Helsinki, Finlandia (2020).
[4] La aplicación depende directamente de la calidad de la cartografía existente en la zona que queremos consultar.
[5] *Help Humans find the Sun*.
[6] Conocido como OSM.
[7] Que se basan en variables tales como la luminosidad de las calles en cuanto a presencia de farolas, o en la anchura de las propias calles.
[8] Barton, Hine y Pretty, 2009.

En nuestro caso, las rutas que queremos generar se darán en base a la elección personal de cada individuo, pero la idea principal es darle al aplicativo un uso más vacacional o turístico, por lo que nuestras variables seguirán este camino, seleccionando rutas más tranquilas y relajadas, que tengan un interés personal que puede variar entre zonas de ocio o de turismo, o incluso zonas verdes: la consecución de un índice de agradabilidad que se adapte a cada usuario de manera particular según su elección.

Para esto, nos basaremos en el estudio reciente publicado por Novack, Wang y Zipf en 2018, que se basa en la creación de una aplicación que divide las calles en segmentos diferentes, unidos por nodos, a los cuales se otorgan unos valores[9] que se multiplicarán por el peso que el usuario otorgue, de 0 a 10, para poder así trazar esta ruta según las características que el individuo crea primordiales para él[10]; en este caso, además, se muestra de manera complementaria la ruta más corta para que el usuario pueda ver la diferencia entre ambos caminos.

Como se ha mencionado, cada proyecto, dependiendo de su objetivo y de los datos disponibles en la zona en que se encuentre, tomará unos datos, unas variables diferentes y, por tanto, generará rutas completamente distintas. Así, por ejemplo, habrá autores que se basan en la obtención de datos de Google Maps para obtener rutas cortas, pero de calidad, basándose en lo que llaman la taxonomía SWEEP[11], que será algo que buscaremos nosotros también en este proyecto, aunque siempre quedará a elección del usuario final las partes que prima.

De hecho, los mismos autores del proyecto en el que basaremos nuestra adaptación del aplicativo han elaborado otro documento, y, por tanto, otro proceso, mediante el cual se generan rutas tranquilas basadas en evitar la contaminación acústica el máximo posible[12], por lo que queda claro que las posibilidades de este tipo de proyectos son infinitas, y todo dependerá de los datos disponibles[13] y del objetivo que cada investigador y desarrollador plantee.

[9] Que serán también normalizados.

[10] De esta manera, podríamos obtener n rutas alternativas dependiendo del valor que le dé el peatón a cada una de las variables o características del aplicativo.

[11] *Safety, Wealth and Well-being, Effort, Exploration, Pleasure*, es decir, Seguridad, Salud y bienestar, Esfuerzo, Exploración y Placer, rutas placenteras que al mismo tiempo optimicen los desplazamientos (Siriaraya et al., 2020)

[12] Algo que también nosotros intentaremos en nuestro proyecto, basándonos en los datos de ruido disponibles en los datos abiertos (Wang, Novack, Yan y Zipf, 2020).

[13] Que no solo varían de una población a otra, entre países o ciudades, sino que también hemos de tener presente la dicotomía informativa entre los núcleos urbanos y rurales.

DATOS Y METODOLOGÍA

El desarrollo de este proyecto puede verse en el flujo de trabajo que se adjunta a continuación:

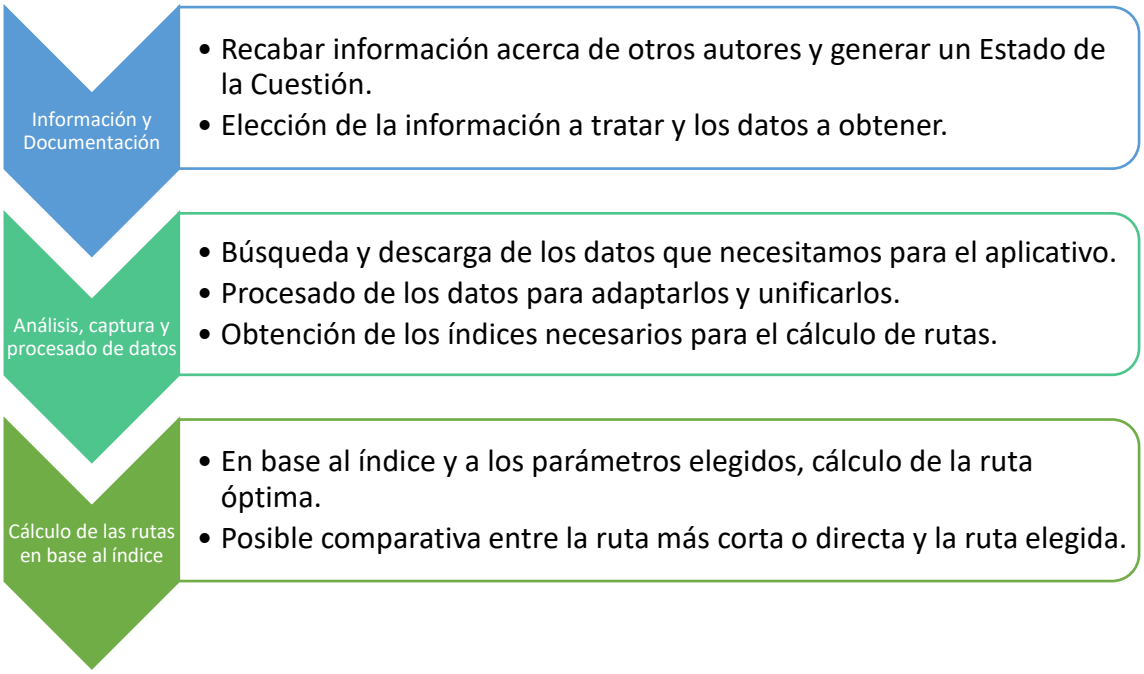


Ilustración 1. Flujo de trabajo general a seguir. Fuente propia.

a. *Información y documentación: datos*

En primer lugar, se recabará la información pertinente para poder ver qué autores han aplicado esta hipótesis que planteamos y qué resultados han obtenido, si han tenido o no éxito en el desarrollo y cuál han sido las premisas elegidas por ellos. De esta manera, encontramos varios documentos, datados en diferentes fechas y en diversos países[14], principalmente para poder evitar la contaminación y crear rutas mucho más saludables para el peatón, pero también para poder aliviar el estrés de un camino con calles muy concurridas o con elevada contaminación acústica.

En nuestro caso, pese a que estas son variables importantes para la consecución de una ruta óptima para el peatón, también queremos centrarnos en la posibilidad de que se puedan elegir zonas de ocio, restauración o de turismo cultural o arquitectónico. Nos basaremos en los datos abiertos de la ciudad de València, y obtendremos la información necesaria en base a la contaminación acústica, a las zonas de ocio y restauración, y también a lugares de turismo cultural y regional que se puedan encontrar en la ciudad.

Se utilizará, para la obtención de estos datos descritos, Open Street Map. Este aplicativo es un proyecto colaborativo, de uso libre y licencia abierta, que contiene mapas de todo el

[14] Principalmente en Alemania y Finlandia, aunque también en Reino Unido.

mundo editables por el usuario. Dependiendo de la zona en que estemos, o del tipo de suelo que queramos consultar, los datos serán más o menos acertados o completos, y que serán creados utilizando toda la información geográfica obtenida mediante captura de dispositivos móviles (GPS) o bien mediante orto imágenes u otras fuentes también libres; todo esto creado por una comunidad de individuos[15], abierta y creciente por todo el mundo[16]. Pero lo interesante e importante de esta herramienta no son los mapas en sí, es decir, su cartografía, sino los datos que podemos extraer de ellos y que serán su salida principal.

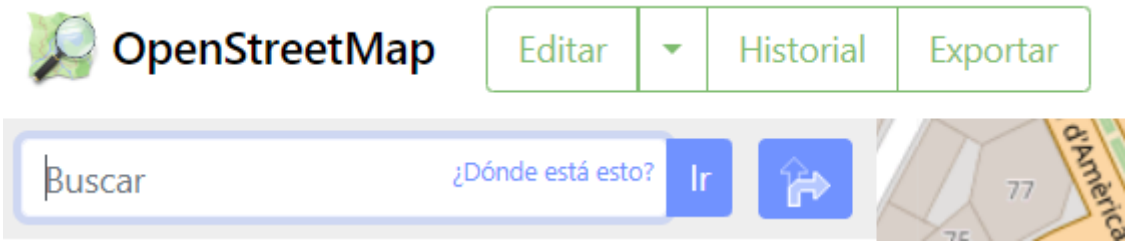


Ilustración 2. Open Street Map. Fuente: openstretmap.org

Open Street Map, conocido como OSM, surge de la necesidad de poder obtener una información geográfica de calidad en cualquier parte del mundo. Con anterioridad a la creación de esta herramienta, el usuario debía pagar por obtener esos datos[17], es decir, por la descarga de los mismos, puesto que esta información no era considerada como un servicio de orden público por las administraciones. Por ese motivo, principalmente por lo encarecido que resultaba el acceso a estos datos, el inglés Steve Coast funda OpenStreetMap en el año 2004, convirtiéndose, poco tiempo después, en una organización internacional sin ánimo de lucro[18].

A partir de ahí, fue progresando lenta pero inexorablemente hasta lo que hoy conocemos, diversas organizaciones permitieron la utilización de sus datos[19], los donaron o incluso financiaron proyectos relacionados con OSM; esta evolución permite ir integrando otros datos nuevos, tanto en localizaciones como en carácter. Además, diversos estudios han demostrado, en Alemania y Estados Unidos, que hace casi diez años ya que contiene una información bastante completa y muy útil[20], aunque, como ya hemos dicho, dependerá de la zona donde queramos utilizar el aplicativo. Se prevé, sin embargo, que siga mejorando con el tiempo.

[15] Denominados comúnmente como *Mappers*.
[16] A este tipo de colaboración abierta se la conoce como *Crowdsourcing*.
[17] Datos que realmente ya había pagado a través de sus impuestos, destinados a generar esa información (cartografía).
[18] Fue inscrita como tal en el año 2006 en el registro de Inglaterra y Gales.
[19] En el caso de España, OSM toma los datos principalmente del IGN, que es el principal organismo público encargado de la cartografía oficial del país, y que en 2008 modificó la licencia de utilización de los datos para liberar parte de estos, de manera que su uso es gratuito.
[20] Aunque en algunos casos, no se pueda garantizar la consistencia topológica de los elementos de la base de datos debido a la propia naturaleza del proyecto (Martínez, S., 2020).

El hecho de que sea una herramienta de acceso y descarga gratuita de datos e información, y que su uso sea tan fácil como acceder a una página web[21], hace que en la actualidad sea uno de los aplicativos más utilizados[22], preferidos por los usuarios por encima de otros como Google Maps, cuya API pasó a ser de pago en el cambio de la política de uso que Google realizó en 2012. Además del acceso mediante la página, también han puesto a disposición del usuario una aplicación denominada **OSM Contributor**, la cual, mediante la ubicación del dispositivo móvil donde se descarga e instala, podremos conocer y también editar estos mapas.

Para poder acceder a esta información de manera directa mediante un breve script que nos permita poder descargar todos los datos y almacenarlos conjuntamente, vamos a necesitar la librería que nos permite trabajar desde Python directamente con el servicio de OpenStreetMap.

Este acceso se puede hacer mediante unas consultas realizadas a través de la herramienta Overpass API que, aunque puede parecer difícil de utilizar en un inicio, nos proporciona una serie de facilidades para ver cómo funcionan este tipo de consultas. Así, por ejemplo, tenemos la herramienta Overpass Turbo, de minería de datos basada en web para OSM, que nos realizará la consulta ya preconcebida, como se puede ver a continuación, en el ejemplo que se ha determinado para, por ejemplo, poder encontrar los museos en la ciudad de València[23]:

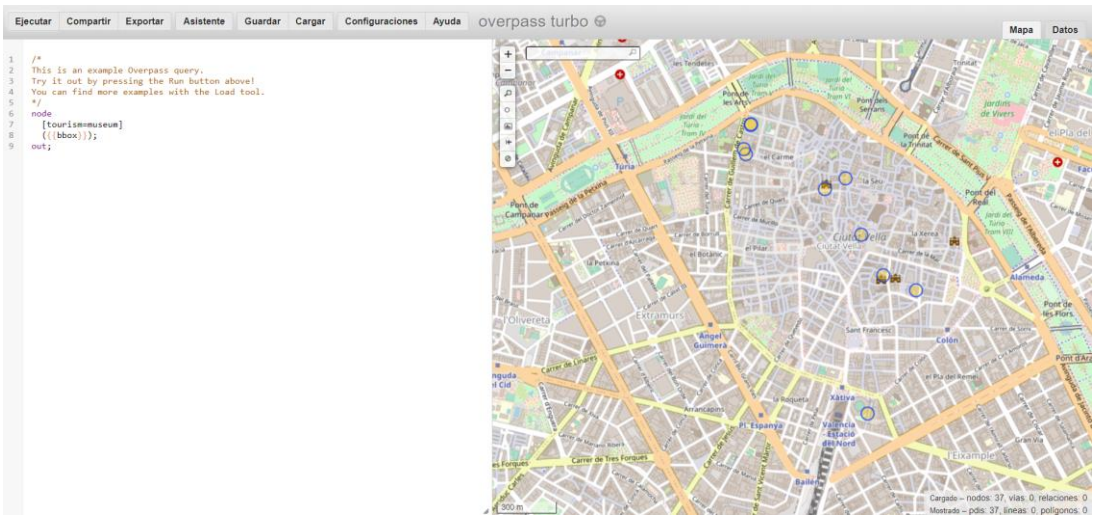


Ilustración 3. Open Street Map Turbo. Fuente propia.

Donde, como podremos ver, la consulta o *query* sería la siguiente:

[21] Encontrada en: www.openstreetmap.org

[22] La propia empresa Apple basa su software de imágenes en los mapas cartográficos de OpenStreetMap para realizar la geolocalización de las fotografías, en *iPhoto*.

[23] La ciudad predeterminada que aparece es Roma, pero se realiza la búsqueda de València para la posterior comprobación de la eficacia de nuestro programa en la obtención de datos, dejando de lado la capital italiana. Además, solo hemos tomado una parte de la ciudad para que se vea con mayor claridad.

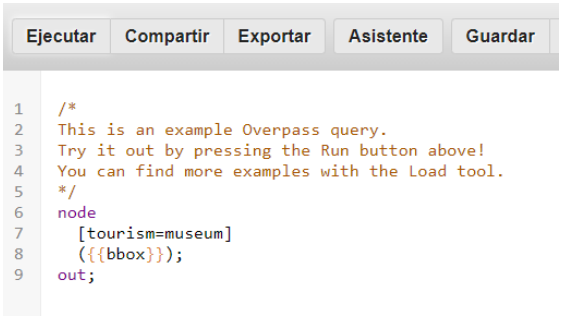


Ilustración 4. Detalle de la query para OSMTurbo. Fuente propia.

Y la devolución de la misma se especificaría en círculos de color amarillo, con transparencia, y bordeados en tono azul, como se puede ver a continuación:

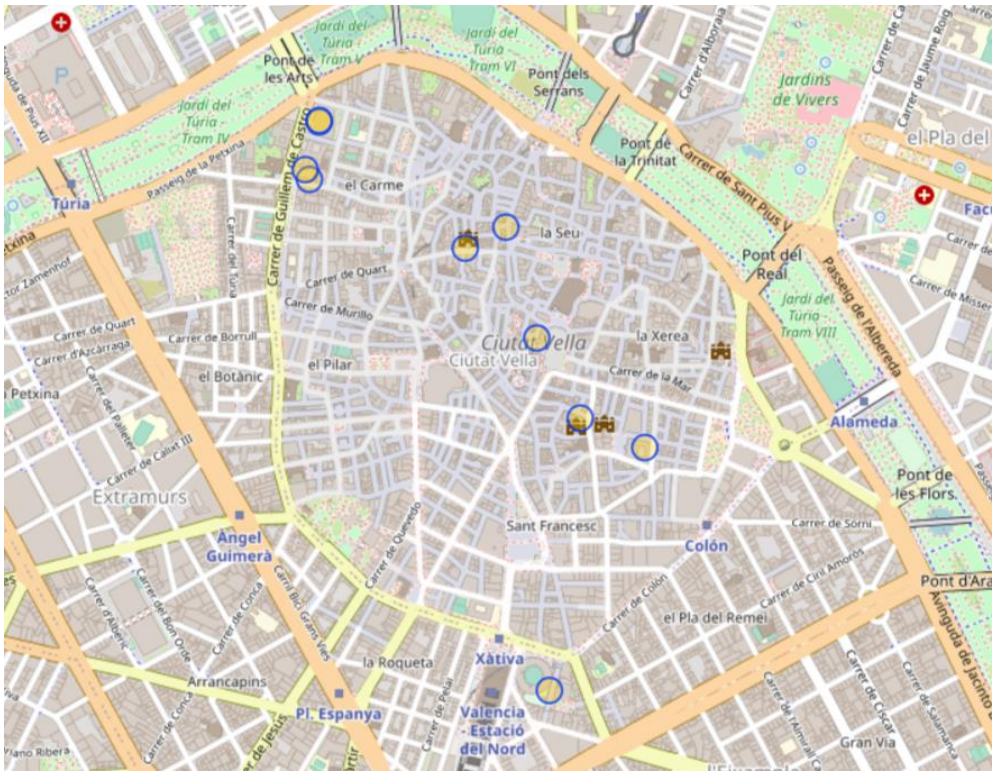


Ilustración 5. Resultado obtenido con OSMTurbo. Fuente propia

La consulta realizada nos toma todos aquellos objetos que, en pares clave-valor, contengan el identificador *tourism*, es decir, que sean elementos turísticos, y particularmente que tengan la etiqueta *museum*, para poder determinar cuántos museos podemos encontrar por la zona[24], aunque algunos de los elementos que nos señala no son estrictamente museos,

[24] Cabe destacar que, concretamente en el caso de València, hay muchos más museos de los que se muestran en la ilustración 5, como por ejemplo el Museo Fallero, que no lo identifica como museo, o la propia Ciudad de las Artes y las Ciencias.

como por ejemplo el Centro Cultural la Nau, de la Universitat de València, o la Torre de San Bartolomé Apóstol[25], pero no señala las Torres de Serranos, por ejemplo.

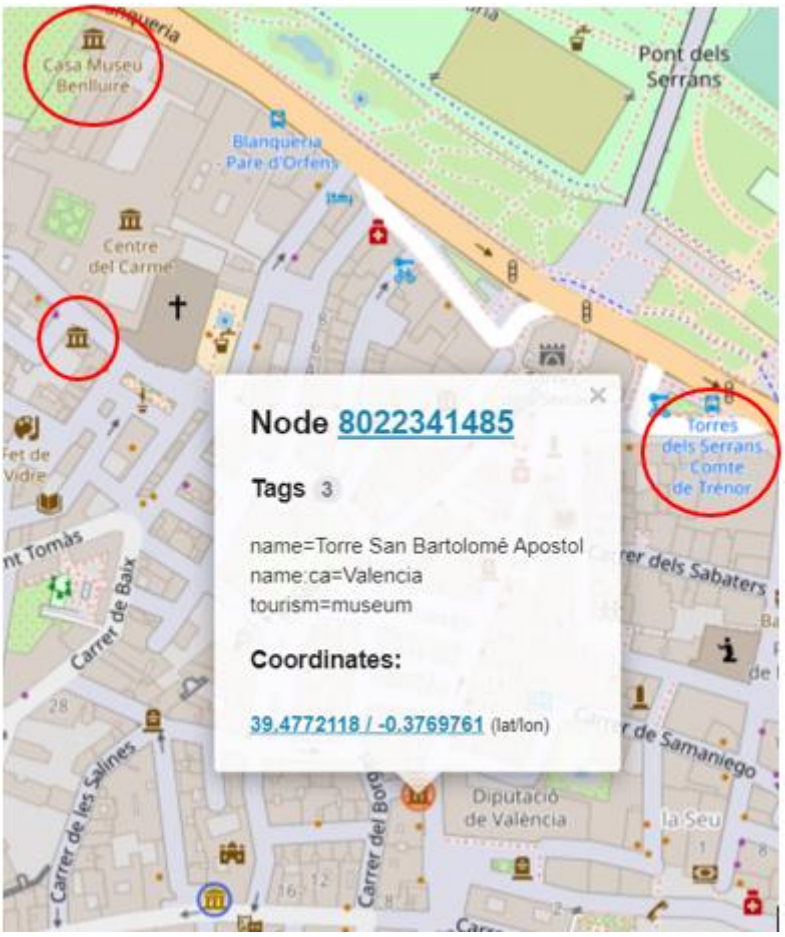


Ilustración 6. Carencias visualizadas en los resultados obtenidos. Fuente propia.

Por este motivo resulta fundamental una comprobación previa de nuestros datos a obtener, y para ello esta herramienta es muy efectiva, puesto que de esta manera podremos cambiar la etiqueta bien a todos aquellos elementos que pertenezcan al identificador *tourism*, o añadir otra búsqueda que complete la realizada sobre los museos (*museum*)[26].

Para poder acceder a todos estos datos de manera que la consulta quede encubierta, optimizando así nuestro programa, ya que no tendríamos que descargar todos los datos, sino únicamente aquellos sobre los que queramos trabajar y que, además, nos permitirá almacenarlos del modo que consideremos oportuno para trabajar con ellos.

Pero no todos nuestros datos podrán ser extraídos de la API de OpenStreetMap, por ejemplo, una de nuestras variables esenciales para determinar la ruta será el *Ruido* generado en las diferentes calles y avenidas, algo que se demuestra que puede aliviar el

[25] El único resto subsistente de la iglesia de San Bartolomé construida en 1239 tras ser conquistada València por Jaime I de Aragón.

[26] Para poder extraer todos aquellos elementos derivados del identificador "Turismo", la query necesaria sería: *[tourism=*]*, lo cual veremos más adelante en el desarrollo del script.

estrés del peatón y mejorar sustancialmente su calidad de vida (Barton, 2019). Estos datos con respecto a la ciudad de València los podemos encontrar en su portal de datos abiertos, con la posibilidad de descargarlos en diferentes formatos, dependiendo de cómo nos resulten más útiles para su manejo[27].

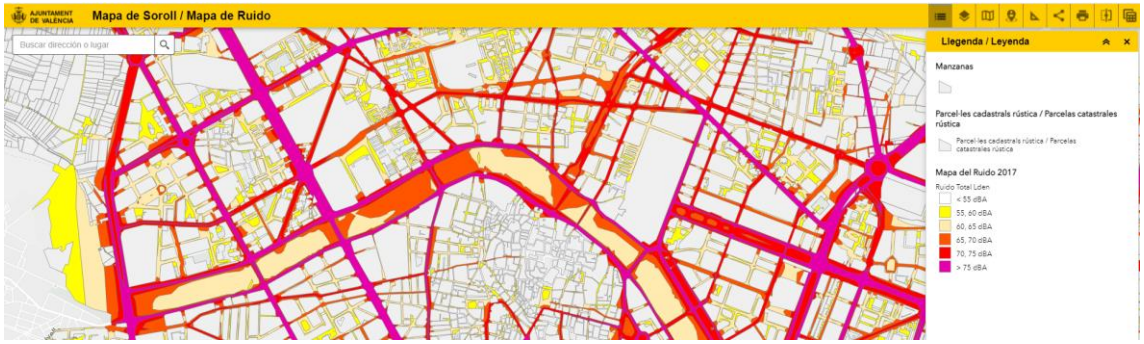


Ilustración 7. Mapa de ruido del Ajuntament de València. Fuente: Ayuntamiento de València

En el año 2017 por orden de la Generalitat[28] vio la luz un proyecto que nos proporciona información sobre los niveles de ruido distribuidos por zonas, englobando así las diferentes calles de toda la red de desplazamiento que hay en València. Anteriormente, la estimación del ruido de las calles se realizaba tomando la velocidad que tenía determinada cada tramo de carretera, y resultaba así un valor aproximado de niveles de ruido, donde por lógica una gran avenida que contuviese un tramo de autovía o autopista tendría un grado más elevado de decibelios que aquellas zonas internas entre urbanizaciones o de acceso más restringido.

De esta fuente de datos extraeremos la información derivada del ruido que hay en cada tramo, que irá escalado entre 55 dBA y 75 dBA, siendo aquellos que queden por debajo del primer valor las zonas más tranquilas, y los que superen el segundo, aquellas calles más concurridas y con mayor circulación de vehículos. Son datos que se basan en el tráfico y que podremos obtener en diferentes formatos: elegiremos aquel que mejor se adapte a las herramientas a utilizar, que sea manejable tanto por medio del SIG de escritorio seleccionado como por medio de nuestro lenguaje de programación.

[27] En este trabajo se ha elegido trabajar con formato *.shp*, dado que tanto el SIG de escritorio como el propio lenguaje de programación Python nos permite leerlo con facilidad y poder trabajar con la información que contiene.

[28] Aunque la página pertenece al Ajuntament de València, y está disponible online en el siguiente enlace: https://geoportal.valencia.es/portal/apps/webappviewer/index.html?id=056f38a5f7e8414b81f03617900b4e67&mobileBreakPoint=300

b. Tratamiento de datos y análisis: programas

I. Elaboración de código: entornos y programación

El lenguaje de programación que utilizaremos en este proyecto será Python.

Ilustración 8. Lenguaje de programación utilizado: Python. Fuente: Wikipedia

Python es un lenguaje de programación muy útil si queremos trabajar con un lenguaje orientado a objetos, puesto que soporta esta orientación parcialmente, y resulta un lenguaje dinámico, cambiante, que va actualizándose de manera periódica. Su licencia, de código abierto, ayuda a su entendimiento y a su legibilidad, lo que está haciendo que sea un lenguaje que va ganando popularidad por momentos, desbancando a otros como C++, PHP o incluso Java en algunos casos.

Python fue un lenguaje que se creó a finales de los años ochenta por el desarrollador Guido Van Rossum en los Países Bajos, y deriva, curiosamente, de la afición de este por el grupo británico de humor Monty Python. Desde este momento fue evolucionando, naciendo como un lenguaje de excepciones e incorporando el manejo de funciones, programación funcional y simplificando y limpiando su sintaxis para volverse más accesible al público general, dejando de ser el lenguaje matemático que era en un inicio, heredero del lenguaje ABC[29]; también se percibe en este lenguaje la influencia de Java en algunos paquetes completos o funciones simples[30].

Toda esa evolución de este dinámico lenguaje dio lugar a lo que hoy nos permite trabajar de manera práctica, sencilla y casi intuitiva. Mejorando a lo largo de los años, facilitándose y adaptándose tanto a los programadores del futuro como a aquellos alejados del mundo de la programación, y permite utilizar un gran conjunto de librerías, que aumentan cada día gracias al tipo de desarrollo open source, y que podremos incluir en nuestros programas

[29] Un lenguaje de programación imperativo, de los años setenta, destinado a la enseñanza y creación de prototipos, con pocos datos básicos, sin variables, y poco preciso. Denominado "aplicación monolítica", lo hacía un lenguaje pesado, poco dinámico, y con adaptación nula a aplicaciones o interfaces para que el usuario pudiese interactuar fácilmente con él.

[30] Por ejemplo, el package *logging*.

con tan solo la ejecución del comando *import* al inicio del mismo. Utilizaremos en el desarrollo de nuestro programa varios paquetes o librerías, aunque algunas serán más importantes que otras, para tratar los datos y también para poder acceder a ellos previamente con una sola consulta.

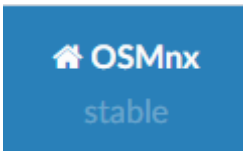


Ilustración 9. Librería para conectar con OSM. Fuente: osmnxdocs

La librería en cuestión con la que trabajaremos se denomina OSMnx, y es un paquete que nos permitirá descargar los datos geoespaciales directamente desde OSM y poderlos modelar, proyectar y visualizar en nuestro aplicativo, así como poder analizar las redes y geometrías espaciales y, gracias a las numerosas y variadas etiquetas que posee la herramienta, acceder solamente a los datos que necesitemos o queramos incluir en nuestro proyecto. Permite descargar y modelar una red para diferentes tipos de vehículo o incluso para peatones, que es la que utilizaremos ya que nuestro aplicativo se enfoca en los viandantes principalmente.

Sin embargo, para poder tratar los datos con esta librería primero necesitaremos configurar Python en nuestro sistema. Para ello, serán eliminadas todas las versiones que tengamos y procederemos a la instalación de Anaconda, una herramienta diseñada por y para científicos de datos que nos permitirá crear el entorno necesario para desarrollar en este lenguaje, puesto que es la manera más sencilla de trabajar con Python. Además de ser una herramienta completa, es gratuita y open-source.

Ilustración 10. Entorno de trabajo: Anaconda. Fuente: anaconda.com

En este caso, la versión de Python que se incluye en esta herramienta es la 3.8, y contendrá un sistema de entornos virtuales que nos va a permitir descargar configuraciones, pero también aislarlas de manera particular unas de otras, entornos que serán seleccionados por el usuario según sus preferencias.

Una vez hemos obtenido Anaconda en nuestro sistema, tendremos que descargar e introducir las librerías necesarias para el desarrollo de nuestro proyecto. Como hemos dicho, la librería que nos permitirá un fácil, dinámico y optimizado acceso a los datos de

OpenStreetMap será OSMnx[31]. Anaconda nos ha instalado en el sistema varios programas, pero los que utilizaremos son *Anaconda Promp*, que es una CMD similar a la que encontramos en cualquier sistema Windows, y *Anaconda Navigator*, que serán los entornos que tengamos para trabajar con sus correspondientes programas y elementos particulares.

Desde Anaconda Prompt tendremos que instalar la librería OSMnx siguiendo los pasos y las sentencias correspondientes según la documentación de la misma:

```
conda config --prepend channels conda-forge
conda create -n ox --strict-channel-priority osmnx
```

Ilustración 11. Comandos necesarios para configurar Anaconda. Fuente: osmnxdocs

Esto nos añadirá, en primer lugar, el servidor de *conda-forge*, para poder descargar las librerías necesarias introduciendo el segundo comando que se ve en la imagen anterior. Así, ya tendremos creado nuestro entorno virtual llamado *ox*, con todas las librerías necesarias[32], algo que podremos ver sin problemas en la interfaz de Anaconda Navigator, como se adjunta a continuación:

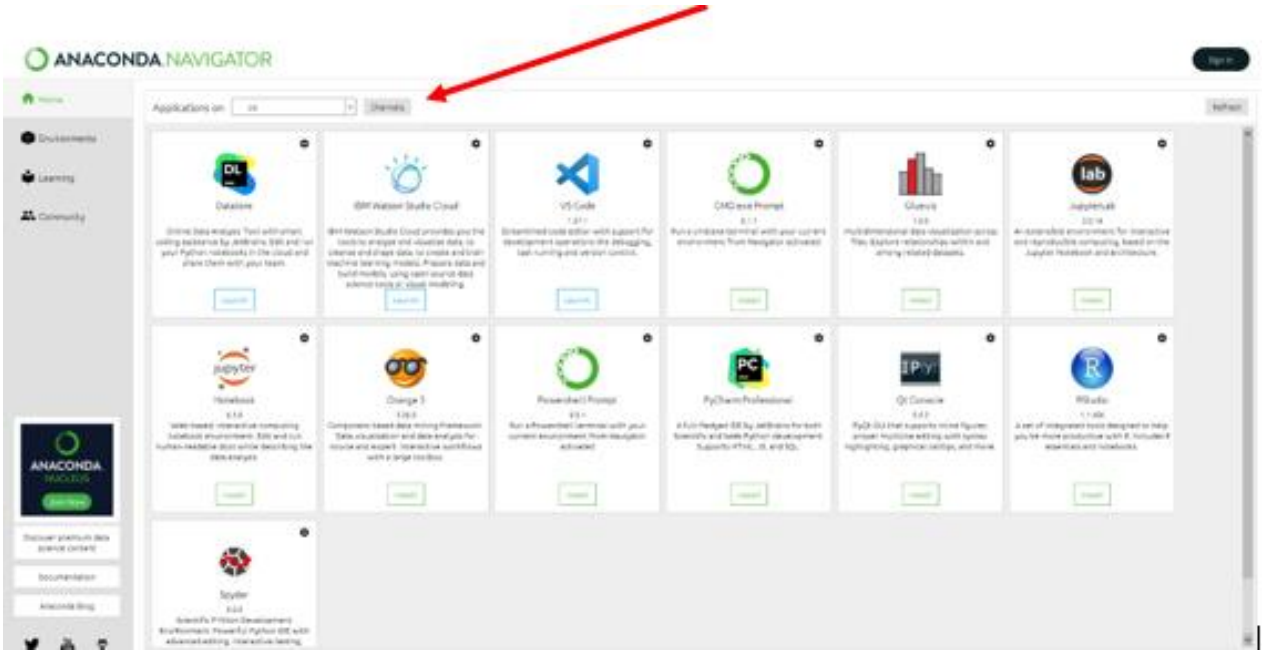


Ilustración 12. Escritorio del entorno. Fuente propia.

[31] Los pasos de instalación de este paquete de Python se muestran en la siguiente página web: https://osmnx.readthedocs.io/en/stable/

[32] Aquellas que no se hayan instalado de manera automática habrán de ser instaladas por la ventana de comandos, Anaconda Prompt, con la sentencia *conda install* y la librería que se quiera instalar, por ejemplo, *folium*.

En la zona superior, donde señala la flecha en la imagen, tendremos el entorno que viene predeterminado por la aplicación, que se denomina *Base*, y el entorno *ox* que, como hemos dicho, es el que hemos creado con las descargas necesarias para poder utilizar OSMnx.

Como hemos visto, Python ha ido desarrollándose a lo largo de los años hasta adaptarse a las necesidades rápidamente cambiantes de la vida moderna. Gracias a ello, se han desarrollado programas que lo hacen más interactivo, como IDLE, que permite una interfaz amigable al usuario, más intuitiva. En nuestro caso, en este proyecto, trabajaremos con el programa Spyder.

Ilustración 13. Programa de desarrollo: Spyder. Fuente: spyder-ide.org

Spyder es un entorno científico diseñado para trabajar con y por el lenguaje Python. Está diseñado por científicos, ingenieros y analistas de datos, por lo que es un programa completo, que contiene herramientas de exploración, edición, debugging y funcionalidades que lo convierten en un entorno de desarrollo fácilmente comprensible y adaptable al usuario, interactivo y agradable a la vista.

Es también un programa que, al estar escrito en el propio lenguaje Python, permite contribuir a su desarrollo al ser de código abierto. Habitualmente, las contribuciones ayudan a arreglar aquellos errores y *bugs* que se puedan dar en el programa o simplemente reportando los problemas surgidos a nivel de usuario, sin necesidad de aportar código alguno. La comunidad de Spyder anima a cualquiera a poder contribuir con cualquier ayuda, ya sea simple reporte de errores, código, o incluso redactar documentos de ayuda o traducir los ya existentes para llegar a un mayor número de personas y hacerlo de un modo más sencillo y asequible.

Con este programa y esta librería mencionada, OSMnx, podremos realizar el acceso a la API de OpenStreetMap y así obtener los datos que necesitemos para nuestro proyecto.

Ilustración 14. Librería de grafos y análisis de redes. Fuente: networkx.org

Una vez hayamos obtenido los datos necesarios para nuestro análisis, necesitaremos crear una imagen que nos pueda dar la ruta completa que tenemos que seguir. Para ello, necesitaremos utilizar una librería cuya función es convertir estas redes obtenidas en grafos, en nuestro caso hemos elegido NetworkX.

NetworkX es un paquete de Python para el manejo y creación de redes complejas. Nos dará herramientas para poder estudiar cualquier tipo de estructura, o infraestructura, con una

interfaz que nos permitirá la implementación de grafos en numerosos tipos de aplicativos, y un entorno de desarrollo de proyectos multidisciplinares y colaborativos y poder trabajar con distintos tipos de lenguaje de programación y de datasets.

En el caso del paquete con nuestro lenguaje de programación, Python, nos permitirá representar de manera simple y flexible de redes y también algoritmos de red claros y concisos. Dentro del paquete principal, que es NetworkX, hay una serie de paquetes menores que ayudarán a realizar todas estas funciones en pos de representar estas rutas, con bases algebraicas y de dibujos.

Pero ¿qué es un grafo?

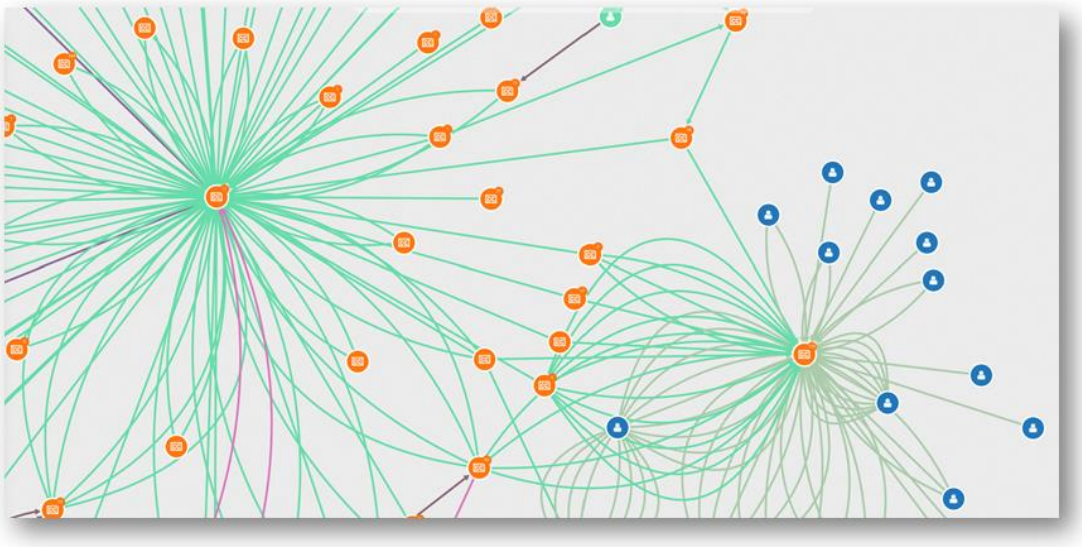


Ilustración 15. Definición genérica y visual de un grafo. Fuente: grapheverywhere

La explicación más sencilla es que el grafo es el resultado de una composición de nodos relacionados entre sí, a través de unas conexiones que obtienen el nombre de aristas.

Un grafo es un término que procede de Grecia, que podríamos traducir del lenguaje griego como *dibujo*, por ejemplo, o incluso *imagen*, y que en este caso se aplicaría al ámbito digital, pudiendo así obtener diferentes tipos de grafos en nuestros proyectos. Estos elementos permiten casi infinitas posibilidades de soluciones y también ser aplicados a multitud de campos diferentes, lo que les convierte en elementos multidisciplinares de incontables características, de gran utilidad para cualquier sector[33], ya que resultan visualmente muy fáciles de comprender siempre que los datos sean claros y concisos y no incurramos en la sobrecarga de información.

Desde los inicios del estudio de este tipo de elemento, el grafo, en el primer cuarto del siglo XVIII[34], se enfocó directamente a la resolución de hipótesis[35] en rutas de desplazamiento,

[33] Aunque se aplica con mayor frecuencia al ámbito matemático e informático.
[34] Concretamente a mediados del año 1736.
[35] La hipótesis de Euler, la teoría de grafos, que parte de recorrer una ciudad (Könisberg, Prusia Oriental, actual Kaliningrado, Rusia) atravesando cada uno del total de siete puentes que contiene, volviendo al

por lo que coincide perfecta y explícitamente con el objeto de nuestro trabajo a desarrollar aquí. El grafo es, por tanto, una estructura relacional entre nodos o puntos (no intrínsecamente espaciales, aunque en nuestro caso coinciden), un *par ordenado* con un número habitualmente finito de vértices y aristas. Existen diferentes tipos de grafos, entre los que se encuentran los dirigidos, los no dirigidos y los etiquetados, que resultan los más conocidos:

- Los grafos dirigidos son aquellos cuyas aristas van en una única dirección, y tienen un lugar de entrada y otro de salida, que corresponderá con el nodo al que van de destino.
- Los no dirigidos son aquellos que no siguen una única dirección, sino que las conexiones pueden recorrerse desde cualquier punto y en diferentes direcciones de manera indistinta.
- Los etiquetados son aquellos que contienen algún peso, es decir, cuyo valor se verá modificado por información adicional que posee este *factor* de peso; al no ser lineal y tener unos valores aleatorios, son los que mejor coinciden con la vida real.

En este caso, con nuestro proyecto a desarrollar, el tipo de grafo que mejor se corresponde con el desarrollo de este aplicativo será el del grafo etiquetado, puesto que no todas las variables tendrán el mismo peso para diferentes usuarios, y sería recomendable que fuesen multiplicadas por este valor o *factor* que las diferenciase.

En nuestro proyecto, este factor de peso que se incluirá en cada variable modificará el valor de la misma, que dependerá del índice individual de cada variable para cada tramo y que, a su vez, generará un índice de *agradabilidad* mayor o menor para que esa ruta sea la elegida, o no, con respecto a las alternativas.

Para poder generar estos índices, tendremos que comprobar si la información generada en archivos *shapefile* resulta o no compatible y, de no ser así, trabajar en estos datos para poder casar los resultados obtenidos. Para ello vamos a necesitar recurrir a un SIG de escritorio, que nos permita visualizar y editar los datos, para depurarlos de manera cómoda y fácil, y tratarlos espacialmente. Los que utilizaremos en este proyecto serán QGIS y ArcGIS[36].

punto inicial; la conclusión obtenida es que se debe pasar al menos dos veces por uno de esos puentes para poder realizar este recorrido. Hoy en día esta teoría es parte fundamental tanto de la informática como de la gestión y comprensión de las bases de datos.

[36] QGIS se utiliza por su facilidad de uso y por ser un software gratuito, pero en algunas dinámicas a seguir se utilizará ArcGIS porque tiene funcionalidades más completas que se han visto durante el desarrollo del máster.

II. Análisis espaciales y tratamientos de datos: SIG de escritorio

Ilustración 16. SIG de escritorio de código abierto: QGIS. Fuente: qgis.org

Previamente conocido como Quantum GIS, es un Sistema de Información Geográfica o SIG, con código abierto y software libre multiplataforma, que opera en todos los sistemas operativos que existen en la actualidad. El proyecto, que vio la luz en el año 2008, permite visualizar, editar y modificar datos tanto ráster[37] como vectoriales[38] por medio de la biblioteca GDAL[39] y también operar con bases de datos[40]. Lo más interesante de este proyecto es lo interactivo que resulta gracias a su interfaz gráfica de usuario (GUI), desarrollado en C++, que permite un entorno más sencillo y asequible para usuarios de todos los niveles.

Al ser un producto de software libre permite añadir funciones más específicas y diversas a las ya existentes, motivo por el cual han surgido también productos nuevos vinculados a este, el producto raíz: QGIS Browser (un navegador de QGIS independiente) y QGIS Server (para poder trabajar con Python y agilizar el desarrollo, sobre todo si trabajamos con servicios WMS).

Pese a que su versión tal y como la conocemos hoy comienza en el año 2008, QGIS ya permitía, como proyecto, importar y visualizar datos desde el año 2002, añadiendo poco a poco las diferentes funcionalidades que podemos encontrar en la actualidad[41].

Otro de los SIG de escritorio fundamentales para el análisis y tratamiento de los datos será ArcGIS, el cual también nos permite, en una de sus funcionalidades, generar rutas de desplazamiento que podamos visualizar[42].

[37] Archivos GeoTIFF, TIFF, JPG…
[38] Shapefiles, DXF…
[39] Geospatial Data Abstraction Library, que nos permite leer y escribir en formatos geoespaciales y obtenerla como biblioteca en diversos softwares. Unifica los datos de todos los formatos que soporta en un modelo básico, traduce y procesa estos mediante sus comandos o funciones de utilidad.
[40] Gracias a lo cual soporta la extensión espacial PostGIS, del sistema de gestión de bases de datos relacional, con orientación a objetos, PostgreSQL.
[41] Primero PostGIS, después los formatos vectoriales y funcionalidades y finalmente los datos ráster.
[42] Aunque es una de las opciones de visualización, nosotros elegiremos otra derivada de la programación.

Ilustración 17. SIG de escritorio de pago: ArcGIS. Fuente: Wikipedia

ArcGIS es otro de los softwares de SIG fundamentales para el análisis, modificación y extracción de datos. La diferencia con respecto a QGIS es que esta herramienta es un producto de pago, no es un software ni libre ni gratuito, que pertenece a ESRI y que fue comercializada por esta empresa a finales del año 1999. También contiene aplicaciones complementarias, tales como ArcGIS Server y ArcGIS Móvil, para la gestión y publicación web, así como para el tratamiento de datos capturados en campo, respectivamente. Al igual que QGIS, está desarrollado en C++, pero también en Python.

ArcGIS cuenta con un gran número de herramientas internas, que contienen diversas funciones que podemos implementar en nuestros análisis y proyectos, desde las aplicadas al mapeo, incluso a generación de rutas automáticas[43], pasando por análisis espaciales y geoestadísticos, así como una caja de herramientas con diferentes conversores de archivos y obtenciones de datos y ubicaciones, muy enfocado a la geolocalización.

En este caso, ArcGIS será utilizado para poder extraer los índices de los diferentes segmentos de red o vía que vamos a analizar. Esta extracción de datos podría realizarse de igual manera mediante la programación, pero el objetivo de este proyecto era, además de proporcionar herramientas y una metodología directa para poder desarrollar un aplicativo en base a nuestro planteamiento, poder ser capaces de aplicar todo lo que se ha conocido y aprendido durante el estudio del máster. De este modo, se ha preferido alternar entre las diferentes herramientas disponibles para poder dar un enfoque completo a un posible desarrollo posterior: prácticamente todo podría realizarse del lado de la programación, así como también puede hacerlo de la parte de un Sistema de Información Geográfica.

[43] Aunque no utilizaremos esta vía para trazar las rutas.

III. Obtención de los valores y parámetros: los índices

Los índices de los que hablamos, que tenemos que extraer gracias a ArcGIS, son unos valores que nos servirán posteriormente para ayudar con la ponderación de los pesos con respecto a la ejecución del algoritmo que nos dará la ruta, o rutas, a seguir.

Estos índices serán valores enteros, aleatorios, concedidos mediante el criterio personal del desarrollo de esta hipótesis; esto significa que pueden ser mutables y adaptarse a otros criterios en diferentes tipos de análisis. En este caso en concreto, se establecerán tres valores, que serán 0, 1 y 2, teniendo en cuenta que 0 sería la ausencia lógica de elementos de una característica[44] en concreto, 1 sería la presencia de un elemento, y el valor 2 englobaría la presencia de más de un elemento, indistintamente del número. Para que sea más visual, sería como se adjunta en la tabla que se muestra a continuación, que, aplicándose, nos dará como resultado simplemente la presencia o ausencia de los elementos, sin entrar en más detalles[45]:

Variable genérica	*Valores*		
	0	**1**	**2**
Turismo	Ningún museo, atracción turística u obra de arte callejera	Algún museo, atracción turística	Varios museos o atracciones turísticas
Ocio	Zonas tranquilas, sin lugar de restauración o bares	Algún bar o restaurante, incluso cafeterías	Zona con varios elementos de restauración
Zonas Verdes	Ausencia total de zonas verdes, ni parques, ni jardines	Presencia leve de jardines o parques, un solo lugar	Presencia de varias zonas verdes en derredor

Tabla 1. Planteamiento de los valores de los índices. Fuente propia.

Pero para la elaboración de rutas necesitaremos primar, dependiendo de nuestro tipo de resultado deseado, unas características por delante de otras, siendo así que, por ejemplo, si queremos una ruta tranquila no primaremos la presencia o ausencia de los mismos elementos que si lo que queremos es recorrer la ciudad en busca de elementos culturales o arquitectónicos, o incluso sociales. Para ello también se proporcionarán unos valores ponderados, que serán unos modificadores del índice obtenido en base al peso individual que cada usuario quiera proporcionar, de manera totalmente subjetiva, a cada característica en su toma de decisiones.

Estos valores se obtendrán en base, nuevamente, a una catalogación orientativa. Como sabemos, el planteamiento de nuestra hipótesis es la generación de rutas alternativas en

[44] Será excluido de esta catalogación la variable *Ruido*, ya que tiene la suya propia obtenida con los datos.

[45] Se proporcionará una tabla más detallada en el momento del desarrollo del programa, cuando se hayan determinado las características y elementos que se quieren rescatar de los datos de manera más concreta. Aquí solo utilizaremos los indicadores generalizados, las variables que hemos querido tomar desde el inicio.

base a cuatro grandes indicadores, dentro de los cuales, después, podremos obtener información complementaria. Así, por ejemplo, nosotros podríamos elegir entre las posibles rutas derivadas de las características que se muestran a continuación, y que seguirán el esquema adjunto:

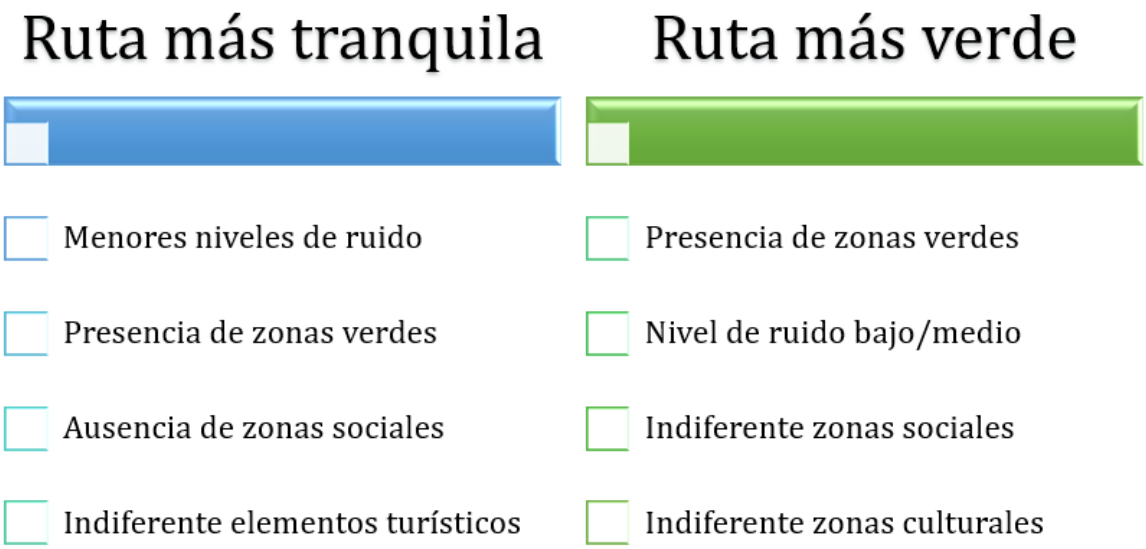


Ilustración 18. Indicaciones de los componentes de las rutas: tranquilas y verdes. Fuente propia.

Ilustración 19. Indicaciones de los componentes de las rutas: social y cultural. Fuente propia.

Estas características tendrán unos valores que concederemos de manera que figuren entre el 0, el 1, el 2 y el 3, siendo el 0 nada importante, el 1 indiferente, el 2 un poco importante, y el 3 muy importante, útil o determinante para la elección de la ruta. De esta manera, la configuración de los índices será realizada como sigue:

Rutas	*Valores*			
	Ruido	**Zonas verdes**	**Zonas culturales**	**Zonas sociales**
Más tranquila	3	2	1	0
Más verde	2	3	1	1
Más social	0	1	1	3
Más cultural	1	1	3	1

Tabla 2. Valores establecidos para las características de las rutas. Fuente propia.

Como podemos ver, en la tabla hemos dado unas prioridades a las variables, basadas en las características elegidas y en el esquema informativo que se puede ver en la ilustración 19. Sin embargo, con respecto al factor *Ruido*, esto no tendría sentido, puesto que cuanto más ruido hubiese en la zona, más peso tendría esta variable, cuando queremos hacer todo lo contrario. Por tanto, para solventar esta cuestión, y dado que nuestros niveles de ruido van en orden del valor 0 al 6, en ascendente, siendo 0 aquellos lugares con menos ruido; si multiplicamos por el valor de peso de importancia de la tabla anterior, obtendríamos un índice muy alto que nos daría la ruta completamente opuesta a la que queremos obtener. Por este motivo, esos valores se restringirán se modificarán a la inversa[46], pese a que la importancia es de 3, es decir, es muy importante, multiplicaremos por la ponderación[47] que se muestra a continuación:

Rutas	*Ponderación*			
	Ruido	**Zonas verdes**	**Zonas culturales**	**Zonas sociales**
Más tranquila	0.0	0.2	0.1	0.0
Más verde	0.05	0.3	0.1	0.1
Más social	0.1	0.1	0.1	0.3
Más cultural	0.1	0.1	0.3	0.0

Tabla 3. Ponderaciones de los valores con respecto a las rutas. Fuente propia.

El valor de este índice sería el que emplearíamos para que el algoritmo elegido pueda proporcionarnos resultados. Pero, como además del peso que va a tener cada ruta con respecto a las demás también necesitaremos la longitud de las mismas, buscando así rutas más agradables, pero igualmente directas o de menor distancia, habremos de tener esto en consideración.

[46] Donde se reescalarán los valores, siendo 1 indiferente, 0.5 algo importante y 0 muy importante. Se ha buscado que el 1 resulte coincidente entre todos los valores. Además, se ha aproximado que nadie buscará una zona de ruido *per se*, sino que les puede resultar indiferente en mayor o menor medida.

[47] La ponderación serán los valores normalizados, para poder obtener un índice manejable.

IV. El algoritmo que desarrolla las rutas: Dijkstra

La librería comentada previamente, NetworkX, nos permite realizar rutas en base a diversos y muy numerosos algoritmos. Así, por ejemplo, tendríamos algoritmos clúster, de conectividad, de cobertura, de distancias regulares, de aleatoriedad y no-aleatoriedad, de reciprocidad… dependiendo de la salida que necesitemos, nuestros datos de entrada y la hipótesis que queramos demostrar, las posibilidades son muy amplias.

En este caso, y para este proyecto, elegiremos el algoritmo conocido como *Shortest Paths*, es decir, aquel que halla la ruta más corta, entre cuyos atributos de entrada al algoritmo encontramos uno particular orientado al peso que queremos añadir al cálculo. El método del algoritmo se conoce como método *Dijkstra*.

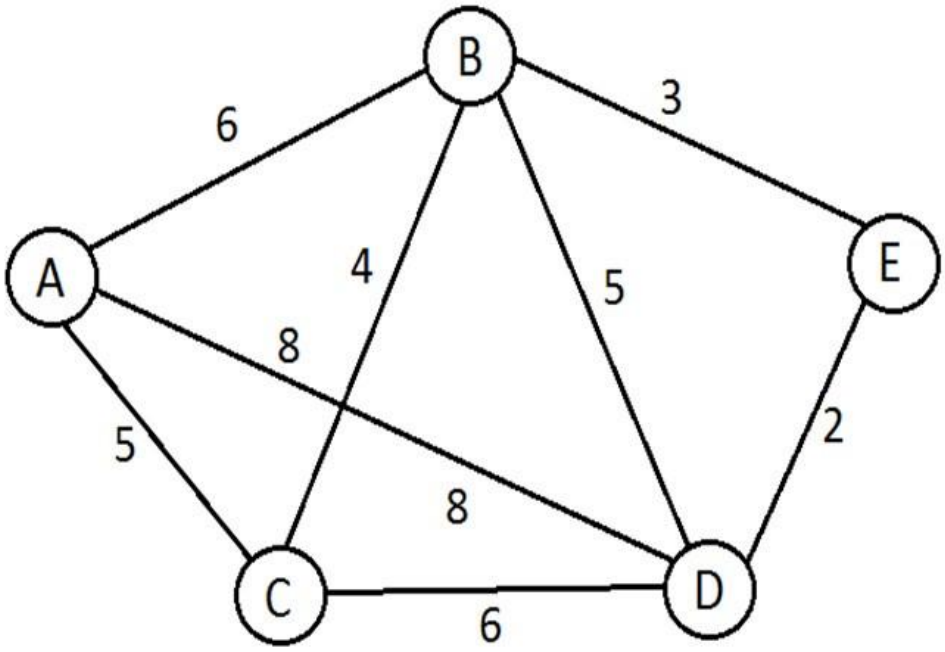


Ilustración 20. Ejemplo básico del algoritmo Dijkstra. Fuente: Universidad INCCA de Colombia

El algoritmo de Dijkstra es conocido también como *algoritmo de caminos mínimos*, es decir, que determina la ruta más corta partiendo de un vértice hacia el resto de los mismos que componen un grafo, teniendo cada arista un peso diferente, como podemos ver en la ilustración 20, adjunta superior.

El algoritmo, que como de costumbre debe su nombre a su creador, procede de un holandés, Edsger Wyde Dijkstra, nacido en 1930, físico teórico y programador en el Centro Matemático de Ámsterdam. Dada su base teórica, dedicó su carrera a la investigación de los problemas surgidos en la programación, para desarrollar soluciones a los mismos. En el año 1959 Dijkstra presentó su algoritmo, aquel de caminos mínimos, ruta más corta, o árbol mínimo, que son los tres nombres con los que se conoce, aparte del apellido de su creador.

Para poder aplicar este algoritmo se deben seguir una serie de pasos previamente establecidos:

- Lo primero sería la selección de un nodo inicial. A este, se le asignará un peso, que dependerá del valor del nodo predecesor (si lo hubiera), del valor del recorrido que se sigue y del número de iteración en que nos encontremos[48].
- Se repite el proceso de asignación de pesos para todos los nodos que preceden al inicial siguiendo la estructura anteriormente mencionada.
- Se recorren los nodos de acuerdo al resultado de los pesos calculados, tomándose siempre el nodo de valor inferior, que será el siguiente en el recorrido.
- Repetiremos el proceso de los dos pasos anteriores tomando siempre el mínimo peso obtenido[49].
- Hay que tener en cuenta que los nodos podrán ser visitados una única vez.

Las aplicaciones de este algoritmo son numerosas, ya que actualmente, con la necesidad del consumismo rápido como el negocio del *fast food* o el *fast delivery*, la ruta más corta siempre tiene prioridad superlativa, por lo que se aplica principalmente a sistemas de navegación y movilidad, así como de geolocalización, en distintos tipos de dispositivos y para diversos tipos de ruta (marítima, aérea o terrestre) así como para todos los tipos de vehículos que puedan permitir el desplazamiento, además, claro está, del que nos ocupa a nosotros: el camino a pie.

[48] Siguiendo la estructura [X,Y](N), donde X es el valor del recorrido, Y es el nodo predecesor y N el número de iteración.

[49] Para ver un ejemplo gráfico, se puede acudir a la ilustración 20 que determina la estructura, comenzando en el nodo A y con el valor de los pesos adyacente a su correspondiente arista.

c. Cuerpo central de la investigación

I. Introducción

El desarrollo de nuestro proyecto se hará conjuntamente entre scripts desarrollados en lenguaje Python, a través del programa Spyder, y el software QGIS, tal y como se ha detallado en la metodología.

Lo primero que tenemos que hacer es obtener la información de OpenStreetMap a través de la librería OSMnx, mediante la cual vamos a acceder a los elementos que figuran en OSM, en sus distintas y diversas capas, gracias a las etiquetas que podemos encontrar en la página de documentación de esta propia herramienta[50], para ir almacenando todo lo obtenido en un DataFrame.

Para comenzar, vamos a necesitar conocer el área de trabajo en que nos vamos a centrar. Al ser este un proyecto destinado a desgranar la metodología seguida y aplicada para poder realizar y obtener rutas alternativas de desplazamiento, y estar continuamente realizando pruebas o testeos, no podemos tomar toda una ciudad desde un inicio, ya que la capacidad de procesamiento no sería la adecuada y lo que buscamos es la máxima optimización de los resultados de nuestra hipótesis para que, de ser necesario, en un futuro puedan aplicarse a mayor escala en el desarrollo de una aplicación o simplemente una página web. Una vez se hayan realizado las pruebas oportunas y hayamos determinado los elementos y objetos que queremos obtener de la base de datos de OSM, ampliaremos la zona a toda la ciudad de València[51].

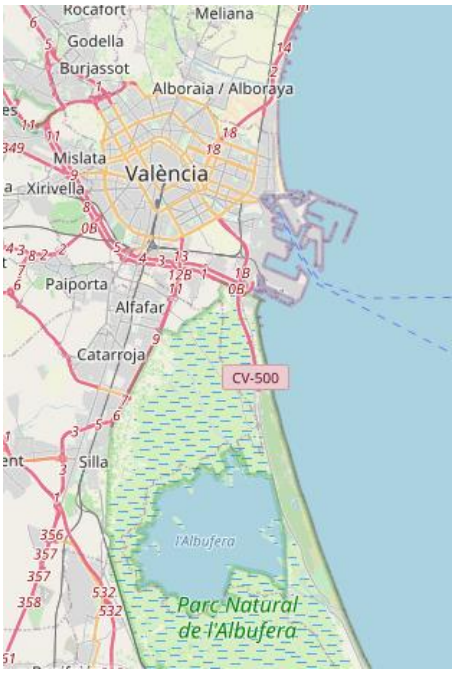


Ilustración 21. Mapa aproximado de la zona a trabajar. Fuente: OSM.

[50] La misma que figura en la bibliografía, accesible en: https://wiki.openstreetmap.org/wiki/Tags
[51] Se tardará un poco más en procesar, pero se obtiene toda la información completa.

Una vez establecida el área de trabajo con los matices detallados que se estimen oportunos, se obtendrán los datos, se unificarán e integrarán para poder trabajar con ellos cómodamente y se procederá entonces a la obtención de los diferentes índices necesarios[52], y finalmente estos se utilizarán para, mediante un análisis a través de Python, determinar los índices finales, que serán aquellos que tomarán la ponderación oportuna dependiendo del tipo de ruta que seleccionemos, y trazar así la ruta que resulte más acorde a nuestras indicaciones como usuarios.

Para poder llegar a la obtención de la ruta, que es nuestro objetivo final, seguiremos el siguiente flujo de trabajo:

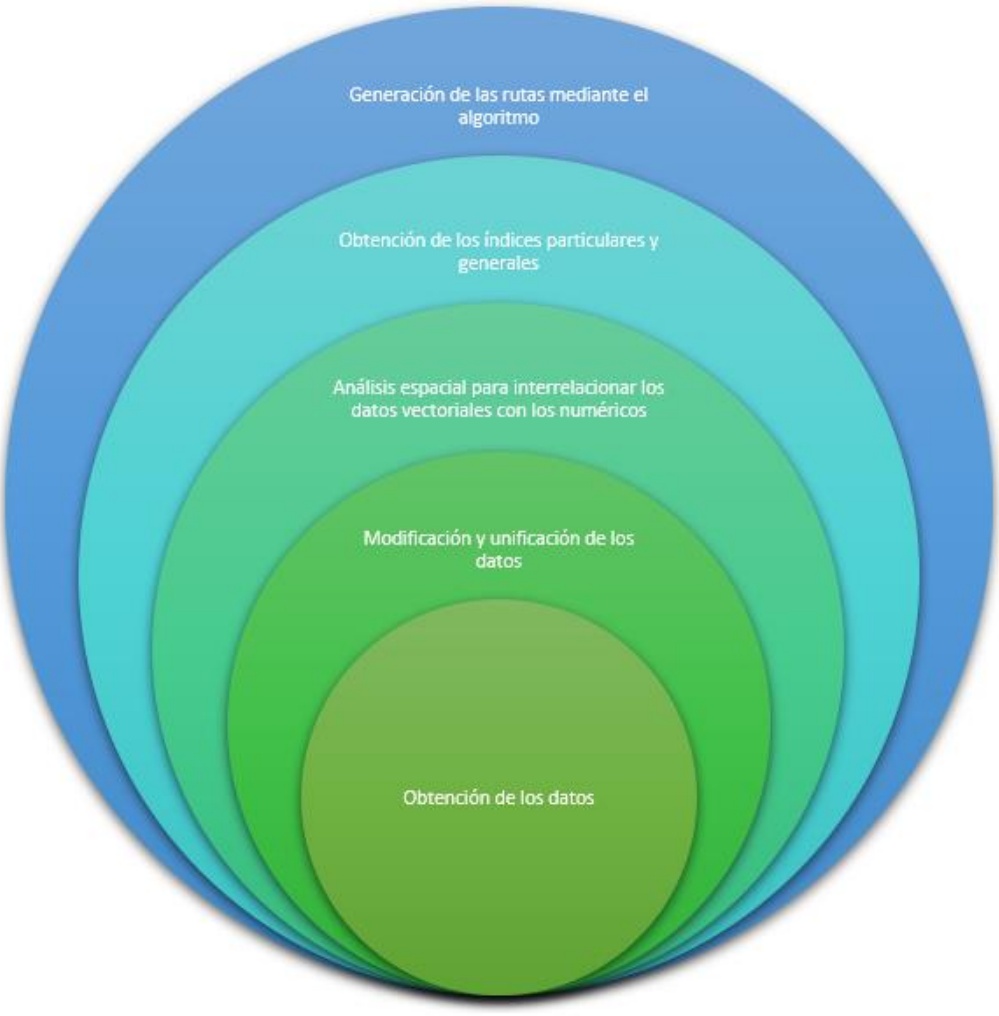


Ilustración 22. Flujo del trabajo general del proyecto. Fuente propia.

[52] De acuerdo a los criterios mencionados en el apartado anterior, de *Metodología*.

II. Obtención de los datos

a. *Introducción y planteamiento*

El origen de nuestros datos es tan diverso como los propios datos que vamos a manejar en sí mismos. De esta manera, tendremos que obtener los datos de ruido del portal de la Generalitat[53], los datos de la red de calles sobre la que nos vamos a basar y, finalmente, los datos de los elementos que querremos incluir en esta red para el análisis de nuestras rutas, ambos obtenidos de OpenStreetMap.

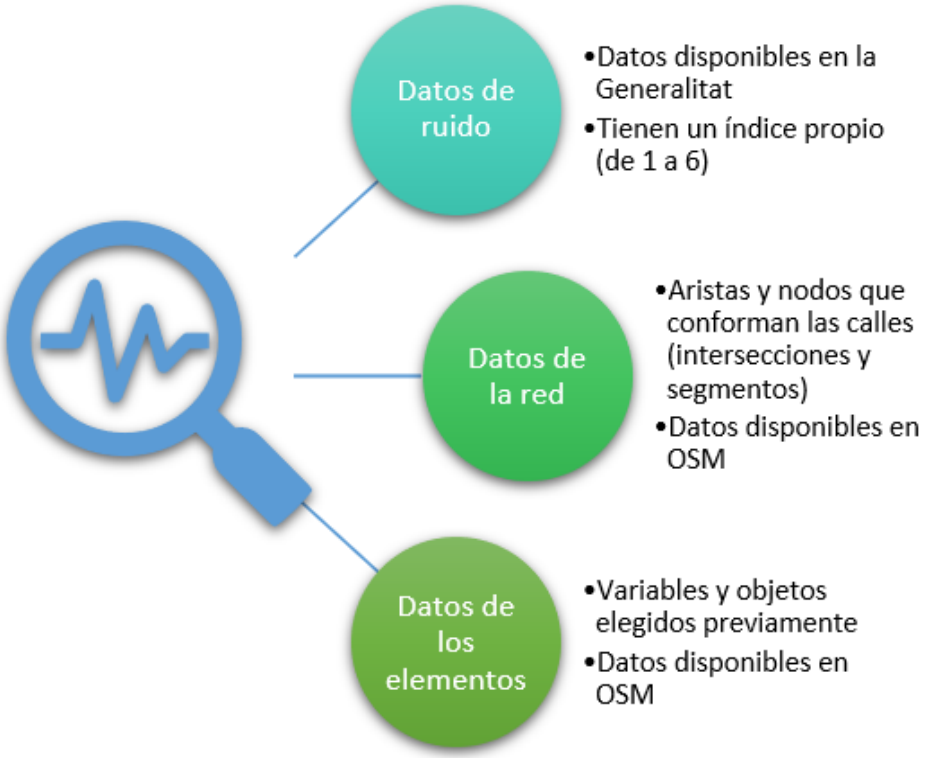


Ilustración 23. Flujo de trabajo particular de la obtención de datos. Fuente propia.

[53] Disponibles en https://www.valencia.es/dadesobertes/es/resource/?ds=mapa-soroll-lden-24-h&id=b2f14356-c4c9-4f0a-8047-eceded493eba para el formato *.shp*.

b. Obtención de los datos de ruido

Como se ha mencionado previamente, los datos que necesitamos de ruido en la ciudad de València se obtienen directamente de un aplicativo desarrollado por el Ayuntamiento de València, que nos dará unos valores en base al tráfico que circule por los diferentes tipos de vías y tramos de la red que recorre la ciudad. Si bien la aplicación web nos presenta una división en seis tramos en base a los decibelios que se generan en cada calle o avenida, nosotros vamos a descargar los datos en un formato manejable para este proyecto, y estos datos nos darán un valor numérico entero, de un solo número, en una escala del 1 al 6, siendo el 1 aquellas calles con menor cantidad de ruido y el valor 6 aquellas avenidas mayores, autovías o autopistas con más tráfico[54].

Valor en datos descargados	Valor en decibelios
1	<55 dBA
2	55-60 dBA
3	60-65 dBA
4	65-70 dBA
5	70-75 dBA
6	>75 dBA

Tabla 4. Valores del ruido obtenidos y decibelios. Fuente propia, datos del Ayuntamiento de València

Estos datos descargados en el archivo tipo *.shp*, que es el que hemos elegido para este proyecto en particular, puesto que permite una lectura rápida tanto en el SIG de escritorio como en el lenguaje de programación que hemos seleccionado, Python, así como un manejo de los datos eficaz, una vez cargado en QGIS se verá como aparece a continuación[55]:

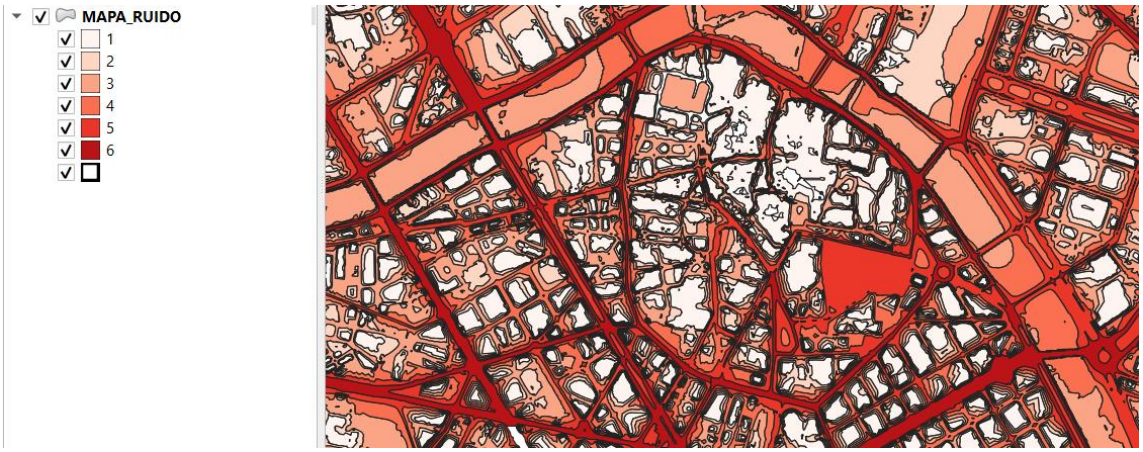


Ilustración 24. Mapa de ruido obtenido de los datos abiertos. Fuente propia.

[54] Esta escala del 1 al 6 es coincidente con los datos de la aplicación, como se ve en la tabla 4.
[55] En la ilustración 24 aparece enfatizada la zona de Ciutat Vella. Aquellos valores en blanco son todos aquellos de los cuales no se tiene información, que aparece en los datos como *all other values*.

c. Obtención de la red de calles

Para poder seguir el flujo de trabajo visible en la ilustración 23, emplearemos la librería OSMnx, que nos permite trazar un bounding box, o zona de referencia, dentro de la cual ejecutaremos nuestros análisis[56]. De esta zona, vamos a extraer los nodos y las aristas[57], que serán las intersecciones de los segmentos de las calles y los propios segmentos de las calles, respectivamente. Necesitaremos descargar esta información para viandantes o peatones, por lo que el tipo de ruta deberá ser *walk*, ya que, como sabemos, OpenStreetMap nos permite además obtener la ruta para diferentes tipos de vehículos además de rutas a pie. Debido al planteamiento de nuestro proyecto, elegiremos este método, y es importante indicarlo ya que, de no hacerlo, OSM podría tomar un vehículo como objeto a desplazar y por tanto tendría en cuenta el sentido de las calles[58] o también la velocidad de las mismas[59].

```
import osmnx as ox
query = open("query_completa.geojson", "w")


place = "Valencia, Spain"
ruta="D:\\UNIVERSIDAD\\TERCER_CURSO\\TFM\\RESULTADOS"
datos = ox.graph_from_place(place, network_type='walk')
proyectar=ox.project_graph(datos)
ox.save_graph_shapefile(proyectar,ruta)
```

Ilustración 25. Código para la obtención de la red de calles, tramos y nodos. Fuente propia.

Donde la variable *datos* almacenará el grafo que se crea del lugar elegido, que hemos almacenado en la variable *place*, y el tipo de red que queremos descargar en este caso, que sería *walk*. Una vez obtenemos esta información, quedará almacenada en la variable, por tanto, dentro del programa: necesitaremos volcar esa información para poder visualizarla correctamente. Para ello, tendremos que proyectar estos datos en un CRS por defecto[60], un paso necesario y a su vez obligatorio para poder después guardar el archivo, que almacenaremos en la variable *ruta* que habremos establecido al inicio y que guardará la proyección por defecto de la vía que hemos solicitado. El archivo que se obtiene al ejecutar este programa es un archivo tipo *shapefile* (*.shp*).

[56] Como hemos indicado previamente, el bounding box será utilizado para el testeo y las pruebas de ejecución, aunque finalmente se seleccionará la ciudad de València al completo. En el Anexo 6 tendremos una alternativa para otra ciudad.
[57] Nomenclatura idéntica a la de los grafos, porque a fin de cuentas en este caso es coincidente el resultado de lo que obtenemos con este elemento.
[58] Giros y direcciones prohibidas, por ejemplo.
[59] Mientras que para los peatones se establece una velocidad estándar de alrededor de tres kilómetros por hora, para un coche influirá el tipo de calle por la que circule y la velocidad a la que esta esté restringida.
[60] En la variable *proyectar*.

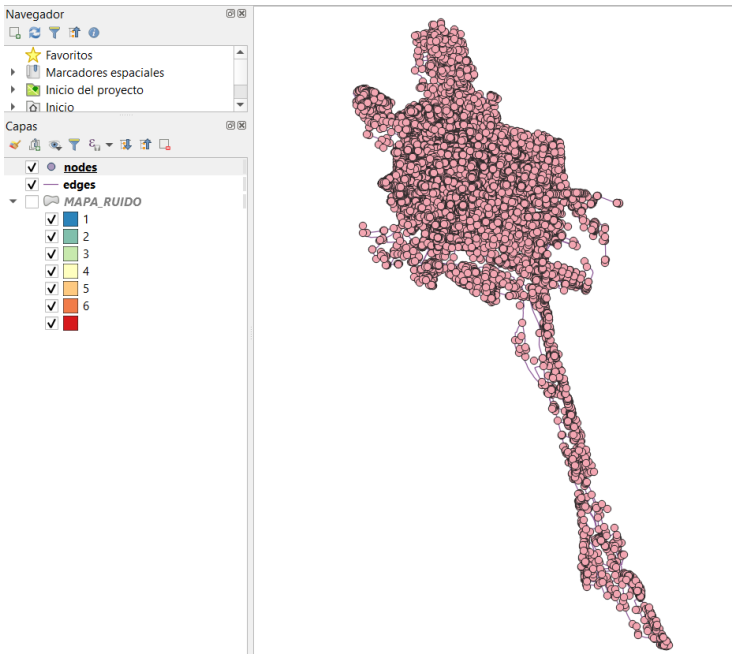


Ilustración 26. Visualización de la red de tramos y nodos, general. Fuente propia.

Así, si abrimos el archivo SHP en nuestro SIG de escritorio, que como hemos indicado será QGIS, podremos visualizar las aristas y los nodos, es decir, los segmentos de las calles y las intersecciones que los unen en aquella zona de València[61] que hayamos determinado para el estudio, de manera que obtendremos el esqueleto de la zona, la base sobre la que tendremos que trabajar, ya que será aquella zona por la que los viandantes podrán caminar, sus cruces y sus intersecciones.

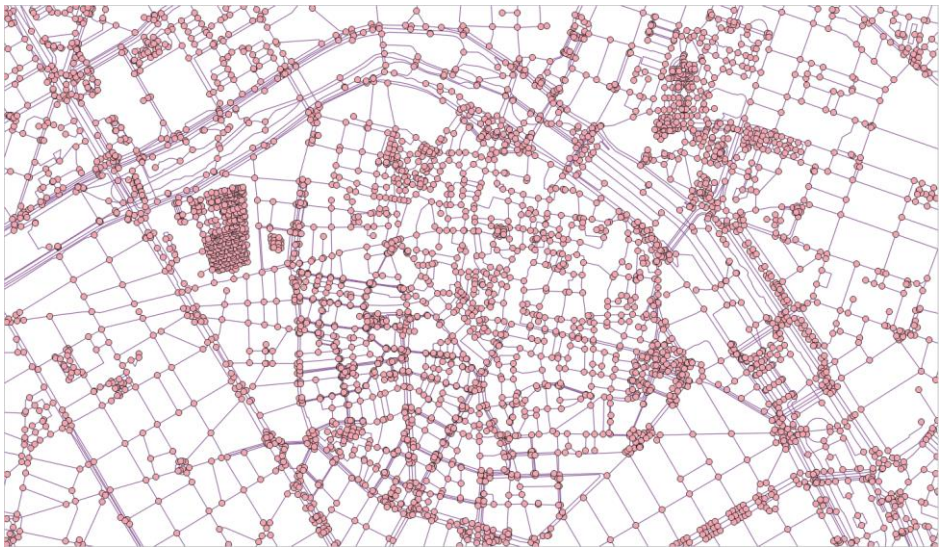


Ilustración 27. Visualización de la red de calles, tramos y nodos particular: zona Ciutat Vella. Fuente propia.

[61] Se realizarán los testeos en el área de Ciutat Vella, pero después se trazarán los resultados completos, pese a que el tiempo de ejecución es mayor.

Pese a que esto lo estamos visualizando con un SIG de escritorio, como hemos dicho, en nuestro caso será QGIS, también la propia librería de OSMnx nos permite crear una imagen en base a los datos obtenidos de nodos y aristas, o segmentos, creando un plot al ejecutar el programa, aunque resulta mucho menos visual que la imagen obtenida en el SIG, como podemos ver a continuación[62]:

Ilustración 28. Plot creado con la librería osmnx. Fuente propia.

Y esto se consigue gracias a una única línea de código, así de completa es la librería que estamos utilizando. Tendremos que indicarle la variable donde tenemos obtenida la red de aristas y nodos, antes de proyectarla[63], como se muestra a continuación:

```
place = "Valencia, Spain"
datos = ox.graph_from_place(place, network_type='walk')
fig, ax = ox.plot_graph(datos)
```

Ilustración 29. Código para generar el plot de tramos y nodos. Fuente propia.

[62] Como podemos ver, la calidad es muy baja y no nos permite realizar un análisis real, aunque para zonas menores pueda resultar más ilustrativo.
[63] Esto solo hace falta para la obtención del archivo *shapefile*.

d. Obtención de los elementos

Ahora solo quedará establecer qué datos u objetos necesitamos extraer de OSM para poder trazar las rutas, es decir, cuáles serán nuestras variables a elegir para generar desplazamientos alternativos.

Como el planteamiento de este proyecto es obtener unas rutas alternativas que puedan ser más cómodas o bonitas, *agradables*, sin dejar de lado la inclinación hacia la distancia menor de desplazamiento, elegiremos variables que nos devuelvan objetos acordes a esta hipótesis:

- Niveles de ruido de las calles. Para poder evitar en su mayor medida las avenidas que puedan tener mayor contaminación acústica derivada del tráfico, lo cual alivia el estrés del viandante (Novack, Wang, Zipf, 2018).
- Presencia de zonas verdes. Para poder así evitar todas aquellas zonas que, ante la ausencia de árboles, parques o plazas con presencia vegetal, presenten una mayor contaminación del aire[64].
- Existencia de zonas turísticas tales como museos, librerías, o fachadas que sean de interés cultural[65] que hagan del paseo algo más agradable o bien que nos permita trazar una ruta de turismo cultural para conocer estos elementos en una ciudad nueva.
- Existencia de zonas sociales, que contengan cafeterías, bares u otros elementos destinados a la restauración, así como tiendas o incluso cines. Para poder trazar rutas que nos permitan poder disfrutar de un tiempo relajados.

De todas estas variables que elegimos, tendremos diferentes tipos de objetos. Así, por ejemplo, en el identificador tendremos que indicar qué tipo de elemento estamos buscando, a qué identidad pertenece[66], y después indicaremos cuál es el tipo del objeto que queremos extraer. Por ejemplo, para poder obtener la información y la presencia señalizada de todos los museos de una zona en concreto, necesitaremos indicar al programa que estamos buscando aquellos objetos que pertenezcan al identificador *Turismo* y que, además, estén contenidos con la etiqueta de *Museo*.

Todas las etiquetas con las que OpenStreetMap puede llegar a trabajar se encuentran dentro del apartado *Tags* en la Wiki OpenStreetMap, pero gracias a la página Tag Finder, desarrollada por Gwerder y actualizada recientemente, con fecha del año pasado, podremos introducir en el buscador qué tipo de identificador necesitamos y este nos devolverá todas las etiquetas que contiene para así ver cuáles son las que mejor pueden adaptarse a nuestro proyecto.

[64] Lo cual se puede convertir en un serio problema de salud (Barton, Hine, Pretty, 2009 y Moelter, Lindley, 2015).
[65] Elementos conocidos como BIC, *Bien de Interés Cultural*.
[66] Turismo, social, zona verde…

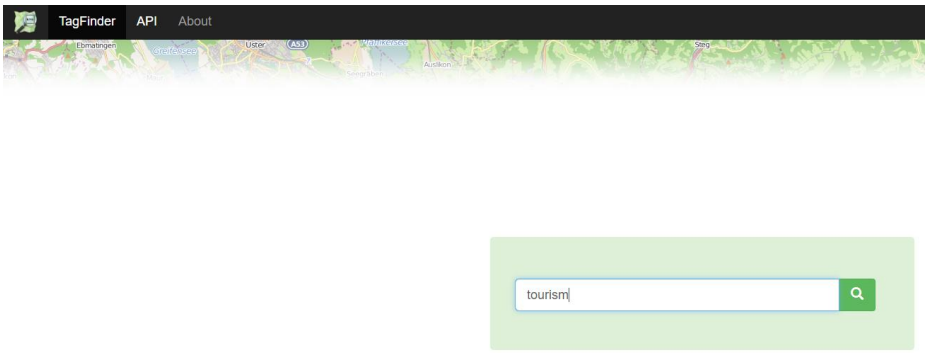


Ilustración 30. Uso del programa TagFinder para encontrar etiquetas afines. Fuente: TagFinder.

Así, si por ejemplo queremos ver qué zonas turísticas podemos encontrar en el área seleccionada, solo tendríamos que introducir[67] la palabra *tourism* y buscar, con lo que obtendríamos algo como lo que se adjunta a continuación:

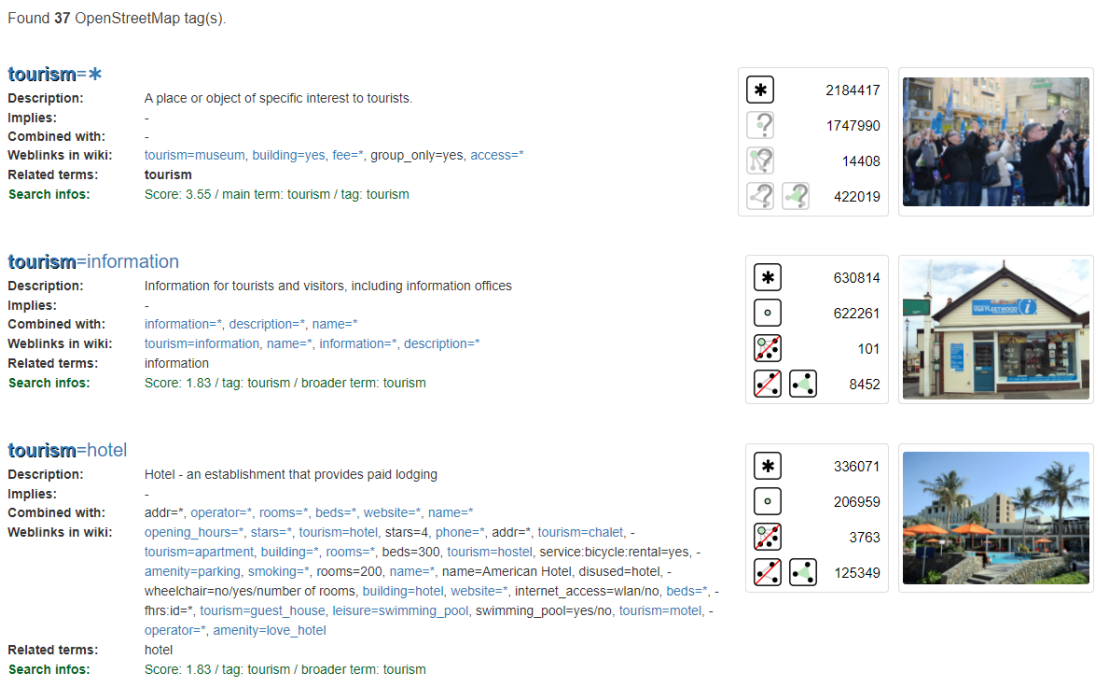


Ilustración 31. Ejemplo de etiquetas obtenidas en TagFinder con la búsqueda de turismo. Fuente: TagFinder.

Donde se muestra el número de etiquetas que corresponden con nuestra búsqueda y se nos detalla la información que nos proporcionará cada una, con una pequeña descripción. Si quisiéramos encontrar todas aquellas zonas que se correspondan con un identificador en concreto, por ejemplo, en este mismo caso, solo tendríamos que introducir un asterisco (*) para indicar al programa que queremos descargar todos los objetos.

[67] Siempre en inglés, puesto que tanto la documentación como los aplicativos están en este idioma.

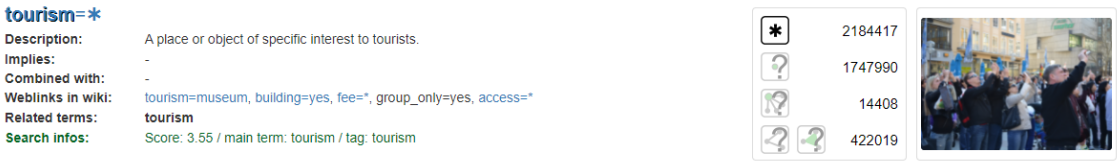


Ilustración 32. Abreviatura para obtener todos los elementos de una etiqueta. Fuente: TagFinder.

Como hemos indicado en el apartado de *Metodología*, podemos realizar una búsqueda rápida, visual, previa al desarrollo del script, para ver si aquellos elementos que queremos buscar tienen o no cabida en nuestra zona, con la aplicación Overpass Turbo[68], cuya consulta nos devolverá la localización de los objetos, si los hubiera, en la zona que hayamos indicado[69].

```
buscar={"tourism":"museum", "tourism":"attraction", "tourism":"artwork",
        "amenity":"place_of_worship", "leisure":"park", "leisure":"garden",
        "amenity":"bar", "amenity":"pub",
        "amenity":"cafe", "amenity":"restaurant"}
```

Ilustración 33. Código para la búsqueda de los elementos elegidos. Fuente propia.

Hemos seleccionado aquellas atracciones turísticas que sabemos que puede haber por la zona, y que hemos comprobado previamente con el aplicativo Overpass Turbo, y posteriormente han sido añadidas al programa[70], así como aquellas zonas verdes que se puedan ver de utilidad para la selección de una ruta[71] y zonas consideradas de ocio[72], como aquellas donde se pueda parar a tomar algo, bien sea un café o una copa, o bien comer.

Dentro de las zonas turísticas que se prevén de interés, encontramos:

- Museos, zonas de atracción turística que engloban no solo edificios destinados a la exposición de obras y objetos, sino también edificios históricos conservados en la actualidad, como las Torres de Serrano.
- Obras de arte callejeras, que pueden ir desde fuentes a esculturas en diversas localizaciones al aire libre.

[68] Aquí se realizarán testeos previos del área de València, primero del área de Ciutat Vella para poder agilizar el proceso, posteriormente de la ciudad completa, y finalmente esta información se pasará a la programación en Python, que será lo que finalmente utilizaremos para el desarrollo de la hipótesis.
[69] Como puede verse en las ilustraciones 3 Y 5.
[70] Otras, como por ejemplo *gallery* o aquellos edificios catalogados como hoteles, moteles, o apartamentos, que en OSM pertenecen también a la categoría *tourism*, se han descartado por no ser motivo de este proyecto.
[71] Algunas de ellas, como los Jardines Botánicos de la Universidad de València (Jardí Botànic de la Universitat de València), pertenecen a su vez a las dos categorías: zona verde y zona de interés turístico.
[72] Aunque aquí se han considerado como *Ocio* aquellos elementos que proceden del inglés *amenity* por adaptarse más a la nomenclatura castellana, dejando la traducción estricta de *leisure*, que significa "ocio", pasar a denominarse en este proyecto *Zonas verdes*.

- Zonas de culto, aunque nos hemos dado cuenta de que OpenStreetMap no recoge todas aquellas zonas de culto que realmente existen. Esto puede ser útil si se quiere visitar iglesias, catedrales[73], sinagogas o mezquitas[74].

Para un área más social, se han contemplado elementos que engloben cualquier facilidad de detenerse en el camino a disfrutar de un tentempié[75]:

- Bares, pubs, restaurantes y cafeterías, han sido aquellos locales elegidos considerados en este proyecto como *Ocio*, lugares de descanso ocasional que permitan algún tipo de refrigerio, algo orientado a socializar.

Finalmente, en las zonas verdes se han tomado aquellas más importantes, puesto que dentro de las mismas puede haber un sinfín de posibilidades[76]:

- Parques, jardines.

CATEGORY	TAG	CATEGORÍA	ELEMENTO
TOURISM	Museum Attraction Artwork	**TURISMO**	Museo Atracción turística Arte Callejero
AMENITY	Place of Worship Bar Pub Restaurant Cafe	**OCIO**	Lugar de culto Bar Pub Restaurante Cafetería
LEISURE	Park Garden	**ZONAS VERDES**	Parque Jardín

Tabla 5. Traducción de los elementos seleccionados y su categoría. Fuente propia.

[73] Aunque estas habitualmente se enmarcarán también en zonas de interés turístico.
[74] En nuestro caso estas últimas no aparecen en la zona elegida, pero sí en otras localizaciones, pues también se ha probado el desarrollo del programa en una zona concurrida de la ciudad de Madrid y produce resultados más completos, pues ha sido previamente seleccionada para tal fin.
[75] No se han considerado lugares de fiesta, o locales nocturnos.
[76] Todo esto se explica con más detalle en su anexo correspondiente, el Anexo 1.

El resultado de esta consulta, en la zona que hemos determinado[77], es la imagen que se verá a continuación:

Ilustración 34. Visualización de los datos existentes en la zona de Ciutat Vella. Fuente: OSMTurbo.

Claro que aquí aparecerán todas las consultas indicadas a la vez, con el mismo tipo de marcador, lo cual puede resultar confuso, pero esta solamente es una prueba inicial sobre la zona que hemos delimitado para comprobar el funcionamiento de las consultas[78].

Buscaremos por tanto aquellas entidades que hayamos dictaminado como óptimas para nuestro análisis, y lo haremos siempre con respecto del área que hemos indicado al inicio del script y que permanece almacenada en nuestra variable *place*, añadiendo después aquellos objetos a obtener que serán guardados en la variable *buscar*, lo cual generará un GeoDataFrame[79] que quedará almacenado en la variable *df_todos*.

```
buscar={"tourism":"museum", "tourism":"attraction", "tourism":"artwork", "amenity":"place_of_worship",
        "leisure":"park", "leisure":"garden", "amenity":"bar", "amenity":"pub",
        "amenity":"cafe", "amenity":"restaurant"}
df_todos=ox.geometries_from_place(place,buscar)
resultado=df_todos.to_json()
n = query.write(resultado)
query.close()
```

Ilustración 35. Código para la obtención de los datos y su guardado en GeoJSON. Fuente propia.

[77] Recordamos, zona de Ciutat Vella.

[78] Como hemos visto, en algunos casos diversos elementos no aparecen señalados, como algunas iglesias o museos, pero este es un problema directamente de la fuente de datos, porque OSM no los tiene almacenados con esta información, como tal. Por ese motivo es importante tener en cuenta este problema, que resulta ajeno a nosotros.

[79] Si queremos añadir cualquier otra entidad adicional, simplemente hemos de indicarlo en la variable *buscar*.

Tras esto, convertiremos el GeoDataFrame obtenido en un archivo de tipo GeoJSON para que las geometrías obtenidas, que pertenecen a una colección[80], no genere problemas en el análisis posterior. Si descargamos esto e introducimos el GeoJSON obtenido en cualquier página de validación de archivos JSON[81], obtendríamos como resultado lo siguiente:

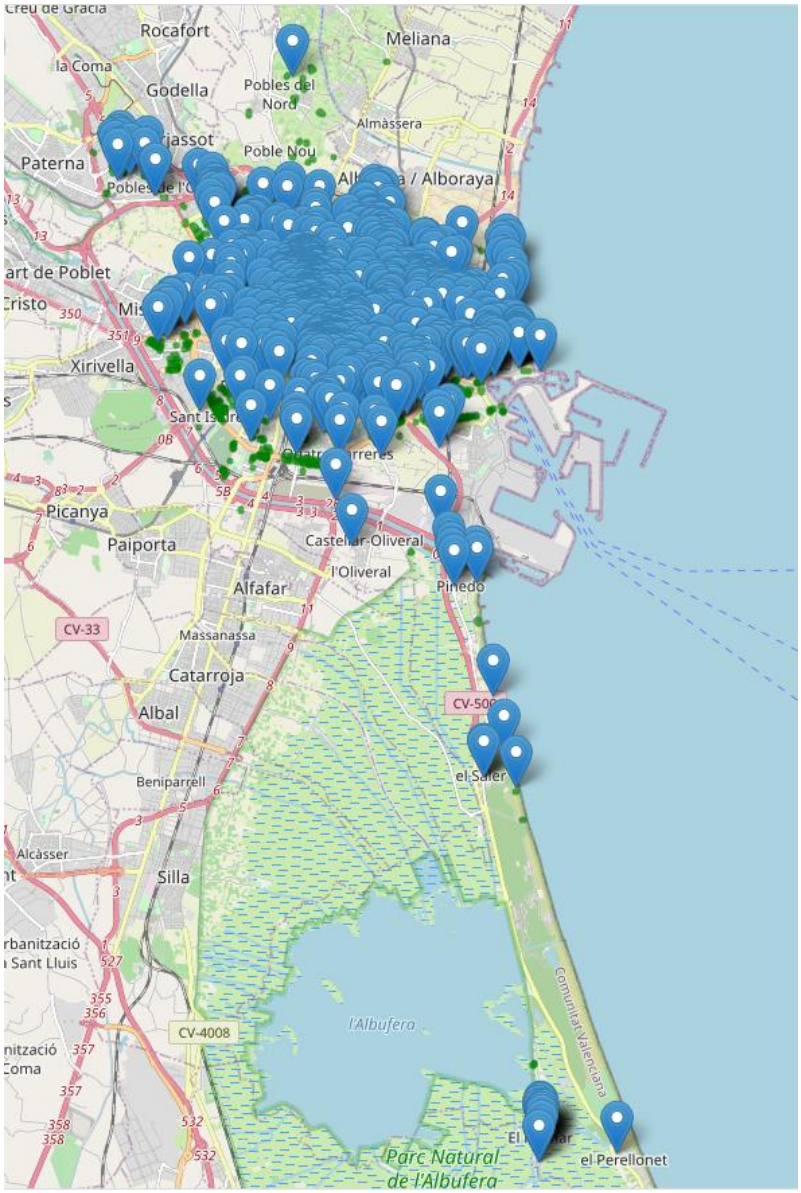


Ilustración 36. Visualización de los datos obtenidos online. Fuente: utahemre

Aparecerán todos los resultados de todos los elementos encontrados en la zona de València, que es el área que le hemos indicado a la variable *place*. Cuando queramos establecer la ruta, sin embargo, tendremos que seleccionar un nodo origen y otro destino, y ahí se restringirán los datos y se darán solo aquellos que estén en la zona.

[80] Porque contienen polígonos, multipolígonos, puntos…
[81] En este caso se ha utilizado la página que figura en la bibliografía, https://utahemre.github.io/geojsontest.html, pero hay muchas otras válidas.

III. Conexión de los datos

En primer lugar, vamos a comprobar si los datos obtenidos en el apartado anterior y el previo a este tienen una relación directa, esto es, que nuestra red de calles se superpone completamente con los datos de ruido de los tramos que hemos descargado, eso ayudará a saber si la información sobre la que estamos trabajando es óptima en cuanto a su geolocalización y, por tanto, el sistema de coordenadas en que se encuentran. Como vemos a continuación[82], ambas capas se superponen completamente en QGIS, nuestro SIG de escritorio:

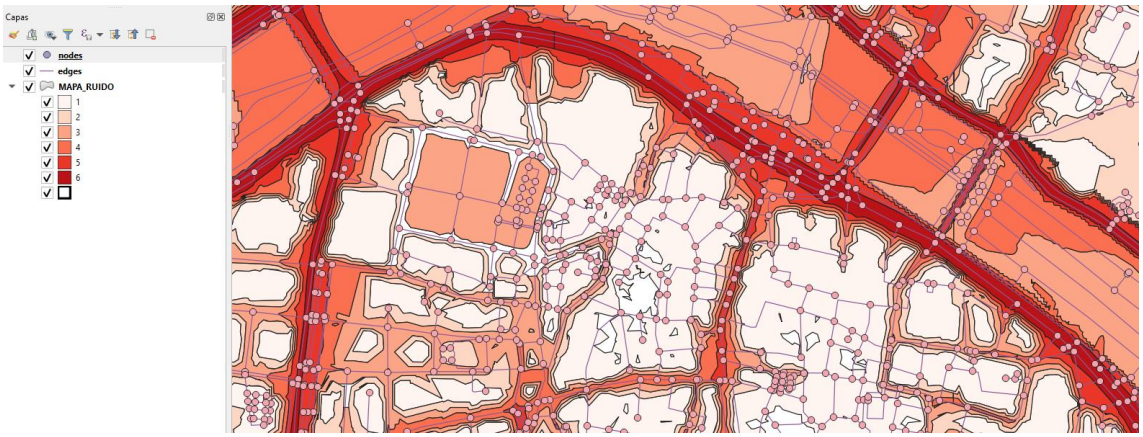


Ilustración 37. Relación entre el mapa de ruido y los tramos y nodos descargados de OSM. Fuente propia.

Tras constatar que la información de la que partimos es óptima, y certera, vamos a proceder a unificar los datos en un solo proyecto. Para ello, necesitaremos el archivo obtenido con los elementos que hemos rescatado de OpenStreetMap, y que podemos encontrar en el apartado anterior, y vamos a transformarlo al mismo tipo de archivo con el que estamos trabajando con respecto a los demás datos para poder hacerlo desde el SIG. Así, para poder obtener un archivo tipo *shapefile* desde el archivo original, GeoJSON, tenemos diferentes formas de enfocarlo: hemos elegido la opción de la conversión en línea, puesto que este no es el objetivo del trabajo y porque el método del programa no nos resultaba de utilidad, pero también existe la opción que nos proporciona el propio ArcGIS en su paquete de herramientas *Conversor*, donde encontramos el método *JSON to Features*[83].

Una vez se han introducido los datos de carácter puntual en nuestra localización, podremos visualizar los puntos de referencia obtenidos desde OSM de manera que nos señalizan aquellos lugares que cumplen los requisitos que solicitamos, aunque lo hacen de manera indistinta, es decir, todos los puntos son iguales, no discrimina en cuanto a visualización los tipos de elementos que señaliza. Esto no nos resulta determinante, ya que solo queremos obtener la información en la tabla de atributos y poder ponerla en común con el resto de los

[82] De nuevo se ha enfocado la zona de Ciutat Vella.

[83] Que convertirá este archivo JSON en entidades, con campos, geometría y referencia espacial tal y como se define en el archivo de origen. A fin de cuentas, un archivo JSON y un archivo GeoJSON tienen la misma estructura. Nuestro archivo de origen no era soportado por esta herramienta, por eso utilizamos la opción del conversor online.

datos que estamos manejando, ya que el posterior script desarrollado para el análisis de los tramos será el que nos proporcione la información desglosada.

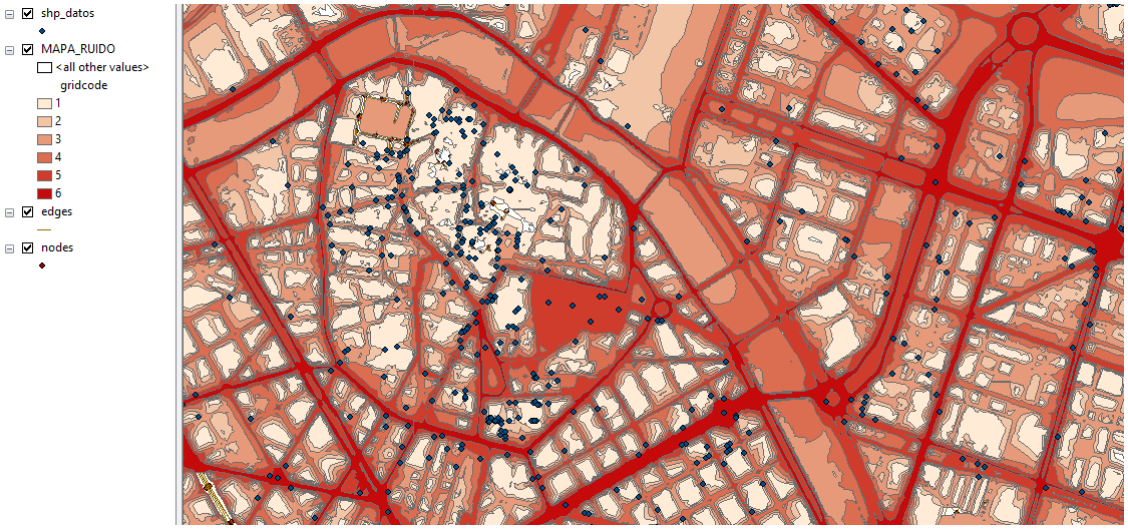


Ilustración 38. Datos de los elementos en la zona de Ciutat Vella. Fuente propia.

Una vez dispongamos de todos los datos en un mismo proyecto, podremos manejarlos para ponerlos en común. En primer lugar, vamos a unificar la información de ruido con respecto a la capa *edges*, que es aquella que contiene la información de los diferentes tramos y segmentos de las calles, ya que necesitaremos el valor del ruido[84] de cada uno de esos segmentos[85]. Para esto utilizaremos el método *Join*, que se encuentra dentro del paquete de herramientas del propio programa.

Antes de poder unificar la información de ambas capas, tendremos que analizar el conjunto de datos que obtenemos del ruido. En la tabla de atributos de la capa del mapa, tendremos cuatro campos, aunque solo tres de ellos contienen información, como podemos ver a continuación en un ejemplo:

MAPA_RUIDO

FID	Shape	id	gridcode
0	Polygon		1
1	Polygon		1
2	Polygon		1
3	Polygon		1

Ilustración 39. Información de la tabla de atributos de la capa MAPA_RUIDO. Fuente propia.

[84] En esta ocasión elegiremos la media de estos valores, ya que un mismo tramo tiene diferentes niveles de ruido y eso nos dividiría los segmentos, y los necesitamos completos para el análisis.

[85] Para poder aplicar la mayor amplitud de disciplinas aprendidas y utilizadas en este máster, se ha elegido trabajar con ArcGIS para realizar esta serie de análisis espaciales, aunque también habría posibilidad de programarlo en diferentes lenguajes.

Donde el FID será ese número de identificación correlativo, automático, que se genera en las bases de datos a modo de identificador único, *Shape* será la forma o el formato de esa geometría, *id* un identificador que aparece vacío y *gridcode* el valor que nos interesa, que es la escala numérica que nos da la información de ruido. Como no podemos editar esta tabla y para lograr que todo sea más intuitivo y efectivo, añadiremos un nuevo campo que contenga los mismos valores que *gridcode* pero que se denomine *Ruido*, del tipo *short integer*, porque los valores son numéricos enteros[86]. La tabla final quedará así:

MAPA_RUIDO

FID	Shape	id	gridcode	Ruido
0	Polygon		1	1
1	Polygon		1	1
2	Polygon		1	1
3	Polygon		1	1
4	Polygon		1	1

Ilustración 40. Resultado de añadir una nueva columna a la tabla MAPA_RUIDO. Fuente propia.

Con los datos actualizados de esta manera, podemos proceder a la unión de las capas[87]. Para realizar esta función, necesitaremos introducir aquellas capas de origen sobre las que queremos operar. Como lo que buscamos es el resultado de la unificación de la información de las calles de la capa *MAPA_RUIDO*[88] y los segmentos de calles que encontramos en la capa *edges*, estas serán las dos capas de entrada de información. La capa de salida será el resultado de dicha unión, donde obtendremos, además, un apartado con la media de ruido del tramo, para que no se subdividan los tramos de las calles:

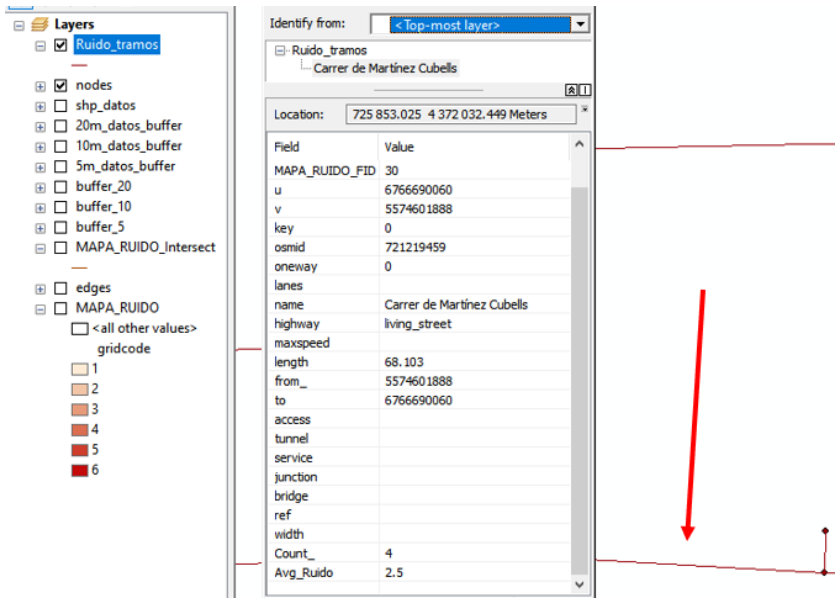


Ilustración 41. Resultado de la unificación de las capas. Fuente propia.

[86] Una vez añadido este campo, el campo *gridcode* puede ser eliminado si se desea; en este trabajo se han conservado todos los campos, incluso aquellos que no nos resultan de utilidad.
[87] Todos los pasos detallados de este proceso se explican en el Anexo 2.
[88] Que están en formato polígono.

Donde, como vemos al seleccionar uno de los tramos[89], nos dará la información del nivel de tráfico en el campo *Avg_Ruido*, que en este caso tiene valor 2.5, es decir, tiene un ruido bajo, se encuentra entre 60 y 65 decibelios de media, aunque contendrá a su vez tramos de valores de ruido bajos (1) y otros de niveles más elevados (3). Un ejemplo gráfico de lo explicado se puede ver a continuación:

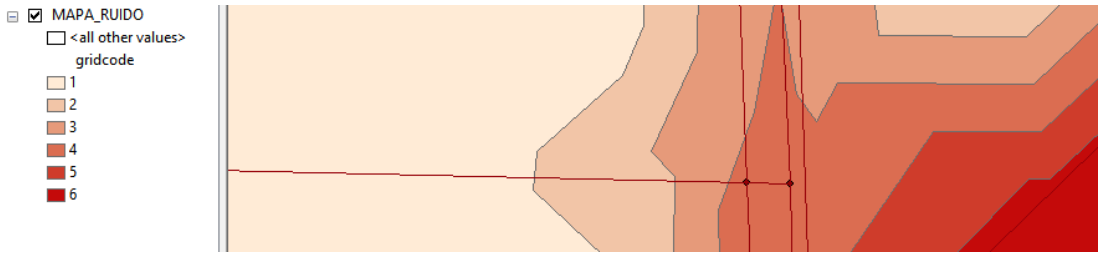


Ilustración 42. Ejemplo gráfico del resultado de una intersección. Fuente propia.

Donde los polígonos de tonos distintos siguen la distribución de niveles detallada en la leyenda del mapa.

Una vez obtenida esta información, solo nos quedará conocer qué elementos de los rescatados de OpenStreetMap quedan cerca de los diferentes segmentos de calles, para poder averiguar los índices que tiene cada tramo. Para ello, en primer lugar, estableceremos un *buffer* en toda la red para amplificar la zona de la calle que puede contener información sobre estos elementos, puesto que habitualmente las aristas pasan por el centro de la calle o carretera y es difícil encontrar un restaurante, museo o parque en el centro mismo de cualquier tramo[90].

Se establecerá por tanto un *buffer* de ocho metros para que pueda contener la mayor información de la calle, cuyo cálculo[91] nos dará como resultado los mismos segmentos, pero más gruesos, conteniendo la información relativa a los puntos de los elementos buscados, como se puede ver en la imagen adjunta debajo:

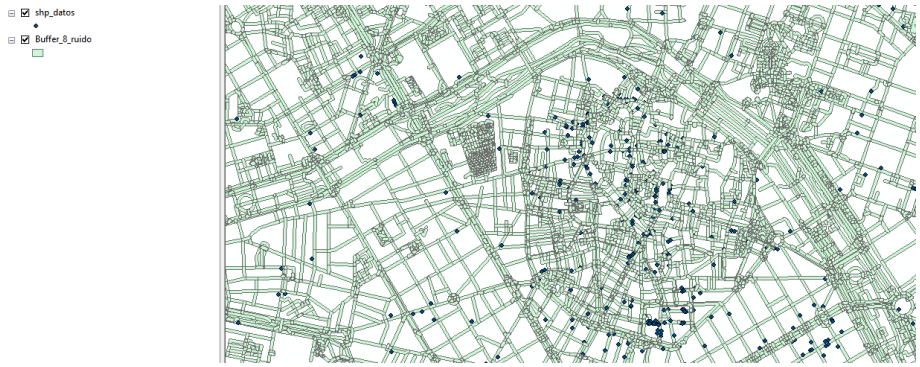


Ilustración 43. Resultado del buffer de 8 metros con los elementos, zona Ciutat Vella. Fuente propia.

[89] El que aparece en la imagen señalado con la flecha roja.
[90] Se han realizado varias pruebas con respecto a esta información para ver qué valor métrico de *buffer* era el óptimo para el análisis. Para ver los resultados obtenidos, será necesario acudir al Anexo 3.
[91] Con la herramienta *Buffer (Analysis)*, también detallada en el Anexo 2.

Una vez obtenido este resultado, la tabla de atributos que tendremos en la capa virtual[92] *Buffer_8_ruido* contendrá toda la información de los tramos de las calles y, además, el valor de ruido que le corresponda. Ahora solo quedará concretar qué puntos de la capa *shp_datos* pertenecen a qué tramos de calle teniendo en cuenta este buffer de ocho metros, para lo cual recurriremos a la función *Join*, que, como hemos mencionado previamente, se encuentra también dentro del propio programa.

Antes de poder realizar esta operación de análisis espacial, tendremos que crear los índices en base a lo acordado en el apartado de *Metodología*, por el cual se establecían unos valores dependiendo del elemento que estuviésemos tomando como referencia. De esta manera, por ejemplo, todos aquellos que sean catalogados como *amenity* y que pertenezcan al sector de la restauración, tendrán unos índices como los que se pueden ver a continuación:

amenity	VerdorIndi	CulturalIn	SocialInd
restaurant	1	1	3

Ilustración 44. Ejemplo de la aplicación de los índices. Fuente propia.

Con los tres índices que se han añadido, de igual manera que se hizo previamente en la capa *MAPA_RUIDO* con los valores del tráfico: añadiendo un nuevo campo de tipo *short integer* puesto que nos moveremos entre un rango de valores de 0 a 3[93]. Así, aquellos lugares que pertenecen al sector de la restauración tendrán un índice predominante de valor 3, que será el *Índice Social*, mientras que los demás índices quedarán con un valor indiferente, al que asignamos el número 1 en el detalle de la metodología.

Una vez tenemos todos los índices, podremos realizar la función *Join* para poder unificar los datos de nuestro buffer creado con respecto a aquellos elementos puntuales que se encuentren dentro de nuestra zona de análisis. El resultado obtenido de esto será el siguiente, donde, como vemos, los segmentos de calle toman diversos colores en base a los valores que contienen:

Ilustración 45. Resultado de la unión de los datos y la capa del buffer. Fuente propia.

[92] Hasta que no exportemos los datos, algunos de los resultados del análisis solamente serán visibles dentro del SIG de escritorio.
[93] Esta adjudicación de valores se hará en base a la tabla 2.

Los colores de los tramos de calles van de acuerdo con la leyenda que aparece en el margen izquierdo de la ilustración, numérica. Se ha tomado como referencia la suma del índice de verdor, que nos dará como resultado el valor de todos aquellos elementos que se encuentren en ese tramo; como vemos, hay tramos con 0, muchos, pero también hay tramos con 4 y hasta 9 elementos de zonas verdes[94].

Tras tener todos los datos numéricos y espaciales en común, solo necesitaremos unificar, en último lugar, los nodos con las calles creadas. Los nodos serán necesarios para poder trazar la ruta, ya que se solicitará un lugar de origen y otro de destino para cada tramo, para así, concatenados, poder obtener el recorrido en base a estos índices que se han extraído y calculado, siempre en base a lo que el usuario quiera dar preferencia.

Para poder unificar la información de los nodos, que ha sido obtenida directamente desde OpenStreetMap al inicio de nuestro desarrollo[95], necesitaremos en la capa final, *Buffer_8_ruido_datos*, que es la que ya contiene toda la información detallada en este apartado, dos columnas nuevas, una que contenga el nodo de inicio de cada tramo y otra cuyos valores sean el nodo de fin de estos.

Esto se conseguirá mediante un *Join*, que nos permitirá unificar esa información[96], por la que se tomarán los datos de *from* y *to* desde la capa *nodes*, y se obtendrán así dos columnas, *FIDfrom* y *FIDto*, que serán, respectivamente -y a su vez indistintamente-, los nodos de inicio y de fin del tramo:

	FID	name	length	Avg_Ruido	Sum_Verdor	Sum_Cultur	Sum_Social	FIDfrom	FIDto
	17597	Carrer del Convent de Santa Clara	50.306	2.5	7	7	21	117	13

Ilustración 46. Tabla con la información necesaria para el análisis. Fuente propia.

Ya tendríamos toda la información necesaria para poder pasar al análisis de los datos. La información fundamental que manejaremos será la capa con el buffer creado, que sería la red de calles -con el buffer de 8 metros-, los elementos que hemos rescatado y el mapa de ruido[97]

Si recordamos, hemos mencionado previamente que todas estas capas creadas como resultado de un análisis espacial no se almacenan en el equipo, sino que lo hacen de forma virtual dentro de nuestro programa. Por ese motivo, será necesario exportar esos datos para poder guardar la capa como un archivo *shapefile*, que será de donde extraigamos la información contenida para poder realizar el siguiente paso del proyecto, en el apartado que sigue a continuación, tras la visualización de los mapas.

[94] Para ver el resto de los índices, acudir al Anexo 4.

[95] En el primer script se obtiene la distribución de la ciudad sobre la cual se va a trabajar, y se proyecta esta información en dos capas: *edges*, que es la capa de tramos o aristas con la que hemos trabajado previamente, y *nodes*, que es la capa de los vértices, intersecciones o conexiones entre tramos cuya información necesitaremos ahora.

[96] Por el tipo de método que es *Join*, solamente nos permite realizar una unión entre una capa y otra, por lo que necesitaremos duplicar la capa *nodes* para obtener la información. El detalle del proceso se muestra en el Anexo 2.

[97] Podemos ver toda esta información en los mapas adjuntos a continuación, ilustraciones 47 y 48 figurando en la leyenda. Al tomar todo València, el primer mapa (47) no se aprecia bien, pero en el segundo (48), que hemos modificado la escala, podremos verlo con mayor claridad.

IV. Mapas

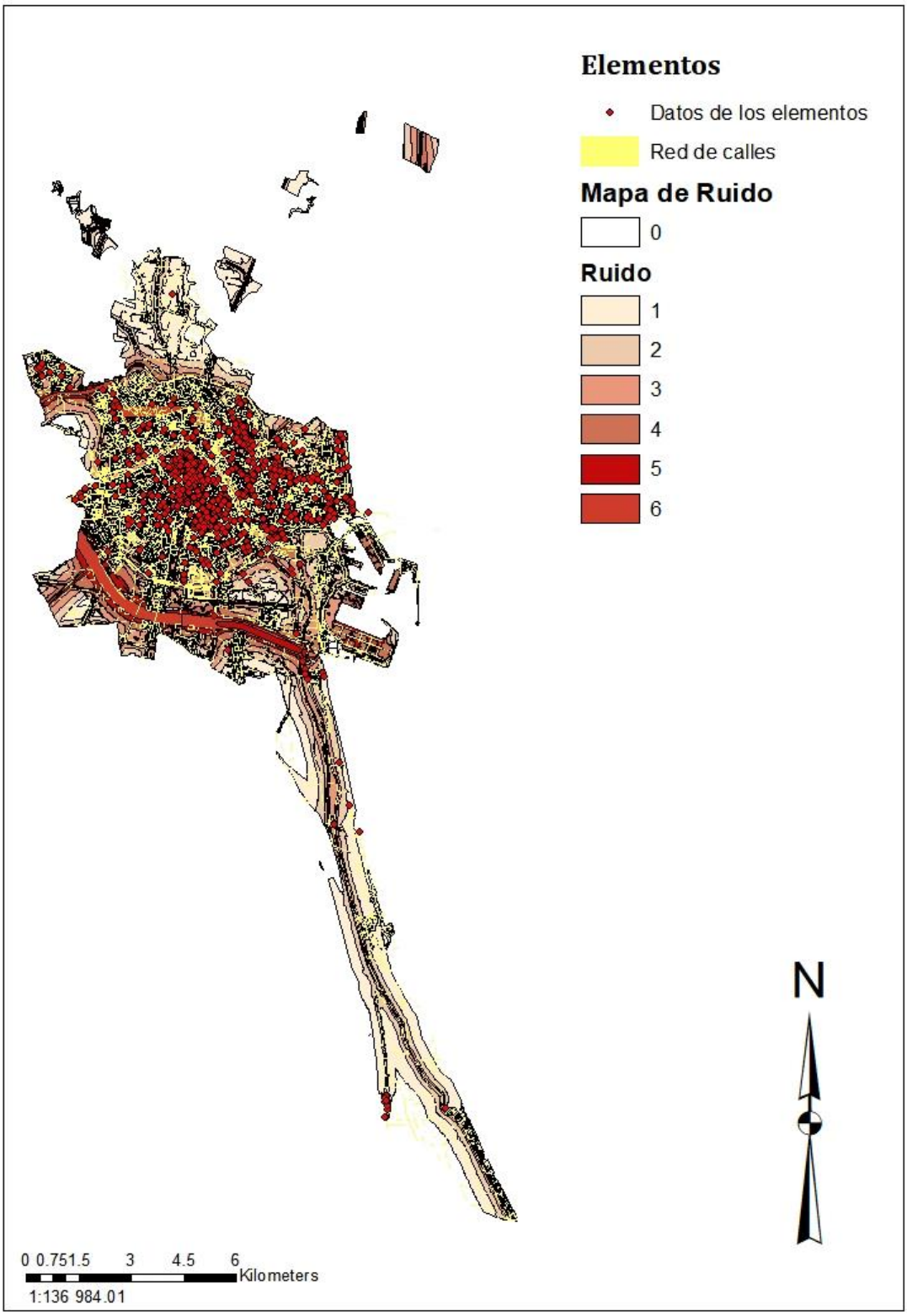


Ilustración 47. Mapa de València, informativo, general. Fuente propia.

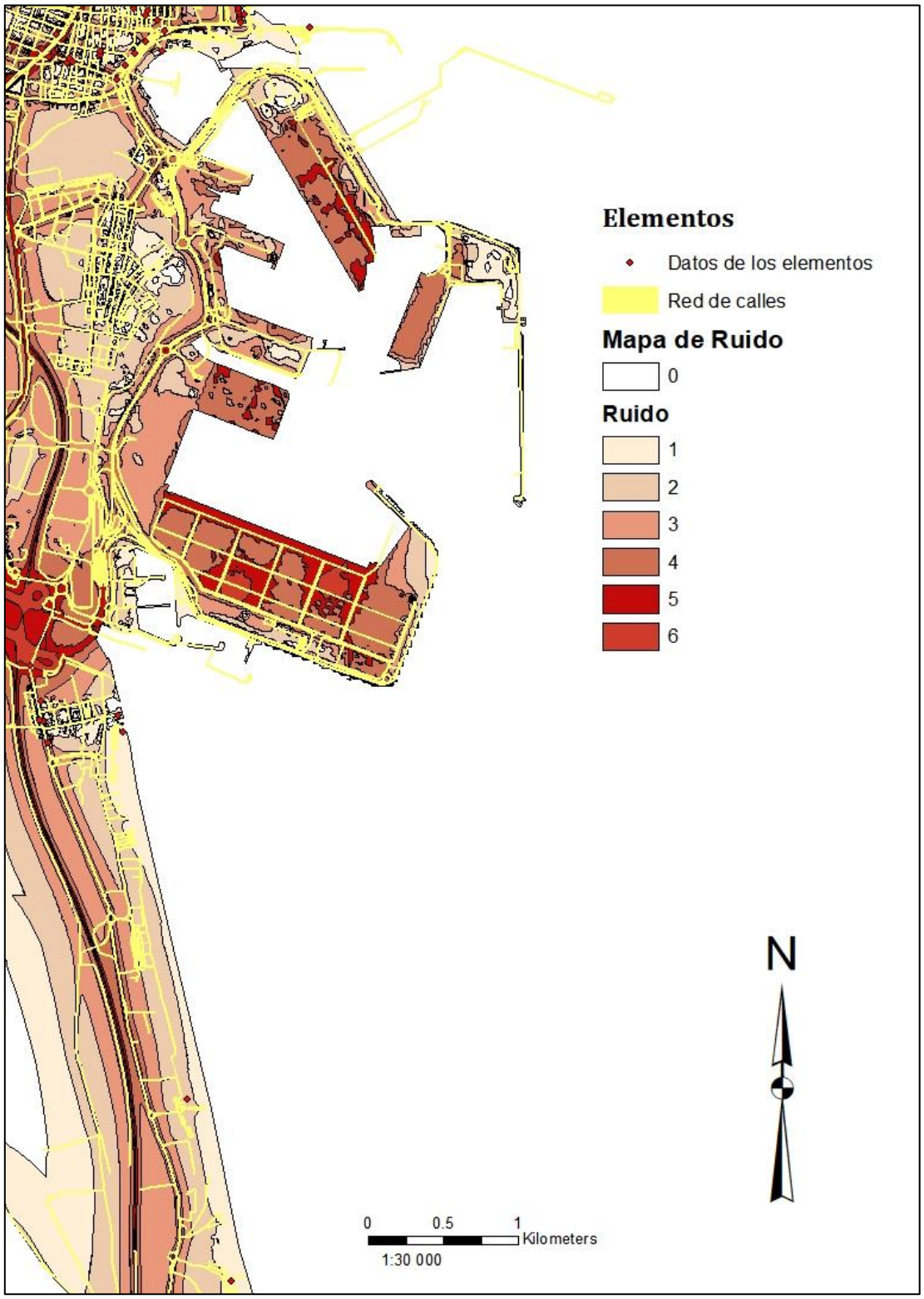


Ilustración 48. Mapa de València, detalle en el Puerto. Fuente propia.

V. Análisis de los datos obtenidos

Una vez se ha realizado todo el análisis espacial en nuestro SIG de escritorio, ArcGIS, tendremos que obtener la información deseada para podérsela facilitar al algoritmo que hemos elegido, el de Dijkstra.

El objetivo del programa a desarrollar es poder extraer toda la información que hemos hallado mediante el análisis espacial y ponerla en común para poder trabajar con ella según nuestros criterios. Como base, tomaremos por tanto esta capa última que se ha exportado, la que contiene todos los datos, al menos, todos los que necesitamos aquí, como podremos ver en la ilustración 46[98]; también se tomará como punto de partida el criterio del usuario a la hora de seleccionar una de las rutas, pues esto será lo que determine el índice que utilizaremos para añadir el peso de cada tramo al algoritmo.

En primer lugar, tendremos que seleccionar la capa tipo *shapefile* para poder abrir el archivo y recorrerlo línea a línea, para, así, poder obtener toda la información[99] de cada uno de los tramos que componen nuestro fichero, que serán los segmentos de esas calles. Se establecerán las variables a utilizar, que serán los índices y el peso[100], y se creará un *input*, es decir, una solicitud al usuario para que seleccione qué tipo de ruta quiere crear, y así, se dará predominancia al índice respectivo[101].

De acuerdo con la tabla de ponderaciones[102] detallada en la metodología del trabajo, se establecerán tres tipos de valores que darán diferentes pesos a los índices generales derivados de la tabla, aunque, como hemos matizado previamente, esta tabla se compone de preferencias que se han investigado de manera personal[103] y por tanto podrán ser modificadas en un futuro dependiendo del análisis oportuno.

Así, por cada tramo, obtendremos un índice para cada una de las variables que estamos manejando, siguiendo estas ponderaciones, como se muestra en los apartados correspondientes del script adjunto a continuación:

[98] Hay muchas más columnas, pero en la imagen se han seleccionado aquellas que nos serán útiles como ejemplo, puesto que las demás no las necesitaremos en este proyecto, pero pueden resultar de utilidad en otros análisis o en un futuro.

[99] Tomaremos el atributo *properties* del DataFrame y se extraerá la información del ruido, verdor, lugares culturales y lugares sociales, así como la longitud de la calle: los campos *Avg_Ruido, Sum_Verdor, Sum_Cultur, Sum_Social* y *length* respectivamente.

[100] *Índice_verdor, índice_social, índice_cultural, índice_ruido* y *peso*, que será la selección final del índice según criterio del usuario.

[101] Esto es ideal en el desarrollo del aplicativo, aunque aquí, al tratarse de metodología aplicada, tomaremos simplemente el programa con la llamada a la función para introducir los datos de entrada.

[102] Ver tabla 3.

[103] Mediante consultas a posibles usuarios medios, sobre qué tipo de importancia pueden tener cada una de las variables en el tipo de ruta que podemos seleccionar.

```
pond_verde_v = 0.3
pond_cultural_v = 0.1
pond_social_v = 0.1
pond_ruido_v = 0.05
```

Ilustración 49. Ponderaciones para la selección de 'Zona Verde'. Fuente propia.

```
pond_verde_s = 0.1
pond_cultural_s = 0.1
pond_social_s = 0.3
pond_ruido_s = 0.1
```

Ilustración 50. Ponderaciones para la selección de 'Zona Social'. Fuente propia.

```
pond_verde_c = 0.1
pond_cultural_c = 0.3
pond_social_c = 0.0
pond_ruido_c = 0.1
```

Ilustración 51. Ponderaciones para la selección de 'Zona Cultural'. Fuente propia.

Tendremos, de manera diferente, la obtención del índice de ruido, puesto que en este caso tomaremos proporciones dadas inversamente al nivel de ruido que estemos manejando, ya que aquellas rutas con un mayor índice de ruido (niveles 5 y 6) son avenidas y rondas muy transitadas, con tráfico elevado, que se prefieren evitar. Se establecerán por tanto cuatro rangos de ruido y se le dará un índice acordado tal y como se muestra a continuación:

Nivel de ruido	**Índice de ruido**
Valores entre 0 y 2	Peso de 3
Valores entre 2 y 4	Peso de 2
Valores iguales a 5	Peso de 1
Valores iguales a 6	Peso de 0

Tabla 6. Ponderaciones de ruido según criterio propio. Fuente propia.

Esto, traducido al lenguaje de programación Python, se mostraría tal y como sigue:

```
if ruido <=2:
    indice_ruido = 3

elif ruido > 2 and ruido <= 4:
    indice_ruido = 2

elif ruido == 5:
    indice_ruido = 1

else:
    indice_ruido = 0
```

Ilustración 52. Código para la selección del índice de ruido. Fuente propia.

Toda esta información será almacenada en un DataFrame para poder acceder a ella desde cualquier punto del código. Este DataFrame contendrá, además, información rescatada de la capa de origen que nos será útil en la localización de los tramos, tales como las coordenadas, los nodos que suponen el inicio y el final de cada segmento y el identificador del tramo, que es el autonumérico que identifica unívocamente cada entidad. La estructura del DataFrame obtenido puede verse a continuación:

df - DataFrame

Index	IdTramo	NodoFrom	LonFrom	LatFrom	NodoTo	LonTo	LatTo	Length	Ruido	Cultural	Social	Verdor
0	0	1	-0.375814	39.4703	0	-0.375852	39.4703	3.304	0	0.6	0.6	0.3
1	1	2	-0.375939	39.4702	0	-0.375852	39.4703	18.027	0	0.6	0.6	0.3
2	2	3	-0.376108	39.4704	0	-0.375852	39.4703	22.729	0	0.55	0.55	0.275
3	3	70	-0.375526	39.4704	1	-0.375814	39.4703	24.782	0	0.566667	0.566667	0.283333
4	4	12917	-0.375852	39.4704	1	-0.375814	39.4703	8.395	0	0.55	0.55	0.275
5	5	12924	-0.375777	39.4703	1	-0.375814	39.4703	7.532	0	0.55	0.55	0.275

Ilustración 53. DataFrame obtenido con los campos para el análisis. Fuente propia.

Con estas premisas previamente detalladas son con las que habremos obtenido los valores de los índices; estos índices, serán a su vez los que se utilizarán dependiendo de lo que el usuario haya indicado en el *input* que le solicita la información de ruta o, en nuestro caso, en la llamada a la función en el programa. Así, si queremos trazar una ruta social, se elegirá la zona social indicando *zona social*, y este hecho hará que nuestro índice predominante por cada tramo sea el índice social. La parte del código que detalla esto puede verse a continuación[104]:

[104] Donde *ponderaciones* es uno de los parámetros de entrada de la función o, posteriormente en un aplicativo, el *input* al que nos referimos. El código completo y detallado se encuentra en el Anexo 5.

```
peso = 0

if ponderaciones == 'zona verde':
    print('elegido zona verde')
    maximo = max(df['Verdor'])
    peso = (maximo-df['Verdor'])
    df['Peso'] = peso

elif ponderaciones == 'zona social':
    print('elegida zona social')
    maximo = max(df['Social'])
    peso = (maximo-df['Social'])
    df['Peso'] = peso

elif ponderaciones == 'zona cultural':
    print('elegida zona cultural')
    maximo = max(df['Cultural'])
    peso = (maximo-df['Cultural'])
    df['Peso'] = peso

elif ponderaciones == 'no ruta':
    print('no se ha elegido ruta particular')
    peso = df['Length']

else:
    print('elegida ruta más tranquila')
    maximo = max(df['Ruido'])
    peso = (maximo-df['Ruido'])
    df['Peso'] = peso

df['weight'] = df['Peso']*df['Length']
```

Ilustración 54. Selección del índice dependiendo de la zona elegida por el usuario. Fuente propia.

En este código, se ha tomado el parámetro *ponderaciones*, que en nuestro programa es uno de los cinco parámetros de entrada que tiene la función creada para generar una ruta, y dependiendo lo que se establezca según el tipo de ruta que se prefiera por parte del usuario, será tomado un índice u otro, y este se añadirá a una nueva columna de nuestro DataFrame, llamada *Peso*, gracias a la cual podremos obtener la columna final del peso que queremos darle a nuestro algoritmo al ser multiplicado el valor de peso de cada tramo por su longitud[105], y cuyo resultado se almacenará en una nueva columna del DataFrame, *weight*.

VI. Creación de las rutas

Una vez tenemos los pesos para cada tramo en base al tipo de ruta que elijamos, solo quedará aplicar el algoritmo de Dijkstra, que tomará como datos de entrada nuestro DataFrame y, de este, las columnas *NodoFrom*, que es el nodo de origen de cada tramo, *NodoTo*, que es el nodo de destino por tramo, y *weight*, que son estos pesos calculados en la imagen superior. Asumiremos que es un grafo no dirigido, ya que nuestra ruta es a pie y no por medio de ningún vehículo, es decir, que podemos ir por cualquier calle sin restricciones de sentido o giros.

```
G = nx.from_pandas_edgelist(df, 'NodoFrom', 'NodoTo', 'weight')
route = nx.shortest_path(G, source=source, target=target, weight='weight')
print(route)
```

Ilustración 55. Ejecución del algoritmo para generar las rutas. Fuente propia.

[105] Como hemos dicho, no queremos que la ruta se desvíe demasiado del camino más corto, aunque preponderemos otros índices.

Mediante este *route* será donde obtengamos los valores de los nodos por los que pasa nuestro camino, que analizaremos en el apartado siguiente, de *Resultados*. Los parámetros del algoritmo *shortest_path* destinados al origen y destino (*source* y *target*) serán también parámetros de entrada de nuestra función.

Una vez obtenida la ruta numéricamente, pues el resultado de este algoritmo es simplemente el identificador del nodo de los tramos, tendremos que extraer las coordenadas de cada uno de los nodos de inicio y de fin y guardarlas a modo de puntos para así poder localizar esos puntos en un mapa posteriormente.

```
coords_lon = []
coords_lat = []
points = []

for i in route:
    l = df.loc[(df['NodoFrom'] == i)]
    if len(l)>0:
        coords_lon.append(l.iloc[0].loc['LonFrom'])
        coords_lat.append(l.iloc[0].loc['LatFrom'])
    else:
        l = df.loc[(df['NodoTo'] == i)]
        if len(l)>0:
            coords_lon.append(l.iloc[0].loc['LonTo'])
            coords_lat.append(l.iloc[0].loc['LatTo'])
        else:
            print('No se encuentra el nodo', i)


for n in range(len(coords_lat)):
    points.append([coords_lat[n], coords_lon[n]])
```

Ilustración 56. Obtención de las localizaciones geográficas de los nodos, latitud y longitud. Fuente propia.

Aquí tomaremos la latitud y la longitud dependiendo de a qué columna corresponda el nodo que estamos seleccionando, si es el origen del tramo (*NodoFrom*) o si, por el contrario, es el destino del mismo (*NodoTo*), y estas coordenadas serán almacenadas en una lista que contendrá los puntos con la estructura [*latitud, longitud*], que serán aquellos que imprimiremos en el mapa.

Para poder visualizar las rutas, primero necesitaremos utilizar un mapa base donde poder ubicarlas. Para ello en este proyecto hemos utilizado la librería *folium*, que nos permite crear mapas fácilmente utilizando únicamente la localización y el tipo de zoom que queremos otorgarle. En este caso la localización atenderá al punto de origen de nuestra ruta, y el zoom tendrá un valor de 15 para que el mapa quede centrado al abrirse:

```
p = points[0]
m = points[-1]

map = folium.Map(location=p, zoom_start=15)
folium.map.Marker(p,icon=folium.Icon(color='black', icon='home'),popup="origen").add_to(map)
folium.map.Marker(m,icon=folium.Icon(color='blue', icon='flag'),popup="destino").add_to(map)
```

Ilustración 57. Creación del mapa y los marcadores de origen y destino. Fuente propia.

Además de crear el mapa, aprovecharemos para añadirle dos marcadores, uno de ellos con el icono de una casa, que señalizará el lugar de partida, en color negro en el cual, si pulsamos,

obtendremos la información "*origen*", que nos indica que es el punto de origen[106]; otro, con el icono de una bandera blanca sobre un fondo de color azul, que nos indicará "*destino*", ya que es el nodo final de la ruta[107].

Una vez tenemos el mapa, necesitaremos trazar la ruta sobre éste, y para ello le pasaremos la lista de puntos generados, que serán localizados gracias a las coordenadas almacenadas, y cuya función *PolyLine* de la propia librería *folium* unirá para trazar la línea que vaya de principio a fin. El mapa será guardado y utilizaremos la librería *webbrowser* para que se abra automáticamente el navegador con el mapa, la ruta y la información:

```
folium.PolyLine(points, color=color).add_to(map)
map.save("map.html")
webbrowser.open("map.html")
```

Ilustración 58. Creación de la polilínea, trazado de la ruta. Fuente propia.

VII. Ejecución del código

Ya habríamos obtenido toda la información necesaria. Como hemos indicado previamente, este código ha sido generado dentro de dos funciones, a las que llamaremos para ejecutar nuestro análisis, la primera de ellas que procesará los datos del archivo *shapefile* y nos devolverá ese DataFrame con toda la información, cuyo único parámetro de entrada será precisamente el archivo a procesar:

```
df = process_data_from_shapefile("D:\\UNIVERSIDAD\\TERCER_CURSO\\TFM\\RESULTADOS\\Datos_completos_buffer_8m_nodos.shp")
```

Ilustración 59. Muestra de uso de la función process_data_from_shapefile. Fuente propia.

```
def process_data_from_shapefile(file):
    shape = fiona.open(file)
```

Ilustración 60. Función process_data_from_shapefile. Fuente propia.

En segundo lugar, la función para generar una ruta, que tendrá varios parámetros de entrada; en primer lugar, el DataFrame que analizará, que será en este caso el obtenido como resultado de la función anterior, *df*; lo que hemos llamado *ponderaciones*, que no es sino el tipo de ruta que queremos seleccionar, *source* y *target*, que son, respectivamente, el origen y el destino entre los cuales queremos trazar y obtener nuestra ruta, y que serán de carácter numérico; por último, el color de la ruta, que, como hemos visto en la ilustración 58, será tomado por el comando *PoliLine* para trazar el camino en ese color. Hemos elegido un color para cada ruta, como se puede ver a continuación en el ejemplo:

[106] Que será *p*, que toma la localización del primer punto de la lista.
[107] Y que será *m*, que toma la localización del último punto de la lista.

```
generate_route(df, 'zona verde', 14510, 18732, 'green')
generate_route(df, 'zona social', 14510, 18732, 'brown')
generate_route(df, 'zona cultural', 14510, 18732, 'yellow')
generate_route(df, 'menor ruido', 14510, 18732, 'orange')
generate_route(df, 'no ruta', 14510, 18732, 'red')
```

Ilustración 61. Ejecución de la función generate_route y parámetros. Fuente propia.

```
def generate_route(df, ponderaciones, source, target, color):
```

Ilustración 62. Función generate_route. Fuente propia.

VIII. Resultados

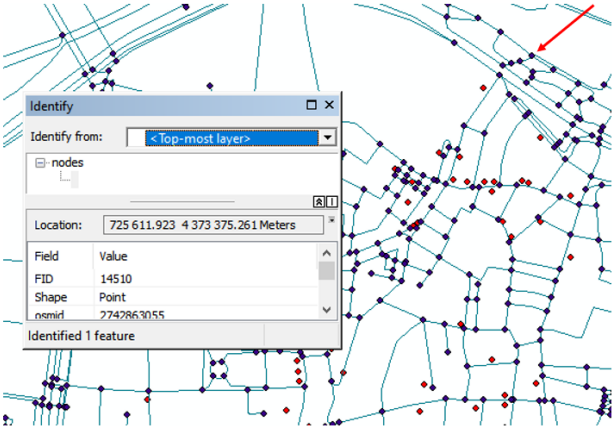


Ilustración 63. Ejemplo de toma de punto de origen, FID=14510. Fuente propia.

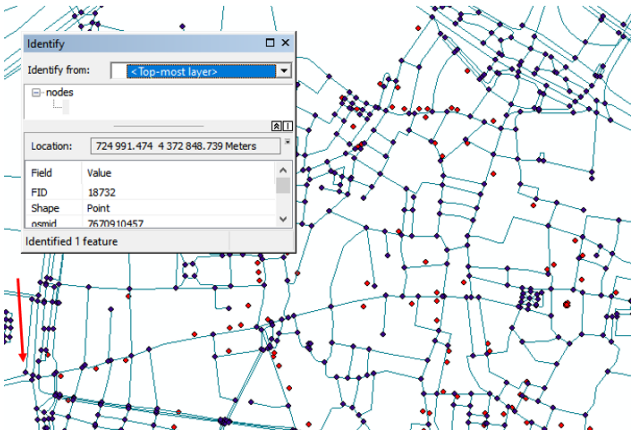


Ilustración 64. Ejemplo de toma de punto de destino, FID=18732. Fuente propia.

Con estos nodos de partida, y buscando una creación de ruta que simplemente siga el peso de los índices sin normalizar, se obtiene el siguiente resultado numérico:

```
elegido zona verde
[14510, 17833, 21377, 21379, 16160, 22317, 16159, 2403, 1786, 1784, 1785, 1853, 1845, 1851, 2085, 2084, 2088,
1817, 2093, 1821, 1819, 22272, 22270, 22269, 2099, 2100, 2101, 1726, 1702, 1725, 2122, 1714, 1731, 1718, 1686,
1669, 1667, 1668, 1670, 1672, 1674, 537, 19382, 19386, 21158, 21157, 18731, 624, 18732]
elegida zona social
[14510, 17833, 21377, 21379, 16160, 22317, 16159, 2403, 1786, 1784, 1785, 1853, 1845, 1851, 2085, 2084, 2088,
1817, 2093, 1821, 1819, 22272, 22270, 22269, 2099, 2100, 2101, 1726, 1702, 1725, 2122, 1714, 1731, 1718, 1686,
1669, 1667, 1668, 1670, 1672, 1674, 537, 19382, 19386, 21158, 21157, 18731, 624, 18732]
elegida zona cultural
[14510, 17833, 21377, 21379, 16160, 22317, 16159, 2403, 1786, 1784, 1785, 1853, 1845, 1851, 2085, 2084, 2088,
1817, 2093, 1821, 1819, 22272, 22270, 22269, 2099, 2100, 2101, 1726, 1702, 1725, 2122, 1714, 1731, 1718, 1686,
1669, 1667, 1668, 1670, 1672, 1674, 537, 19382, 19386, 21158, 21157, 18731, 624, 18732]
elegida ruta más tranquila
[14510, 17833, 21377, 21378, 21381, 18722, 1810, 1564, 1563, 1552, 16237, 13867, 13866, 1664, 1661, 1658,
1655, 1654, 1656, 624, 18732]
no se ha elegido ruta particular
[14510, 17833, 21377, 21378, 21380, 22316, 18722, 1810, 18721, 22330, 1814, 16158, 1787, 2079, 2080, 2081,
2083, 1863, 1860, 1861, 2096, 1816, 2098, 1818, 2099, 2100, 2101, 1726, 2104, 2107, 1701, 1696, 1694, 1697,
1700, 1691, 1682, 1692, 1679, 1677, 1675, 532, 1672, 1674, 537, 19382, 19386, 21158, 21157, 18731, 624, 18732]
```

Ilustración 65. Resultado numérico obtenido directamente desde el algoritmo. Fuente propia.

Como vemos en la ruta generada por el algoritmo, que nos devuelve los nodos óptimos por los que tiene que pasar la ruta para seguir los diferentes criterios que imponemos, avisando previamente el tipo de ruta en la que nos estamos basando, las tres primeras rutas son iguales, puesto que su índice de verdor, social y cultural ponderarán los mismos tramos de calle, mientras que la ruta trazada para poder elegir una ruta caracterizada por su tranquilidad da un rodeo mayor, evitando la zona más céntrica de la ciudad, bordeando por la zona de parque y la *Carrer Guillem de Castro*, la calle mayor del mapa 67, cuya media de ruido es elevada (6), pero que resulta la manera más directa de ir desde la zona de los Jardines del Turia hasta el destino[108].

Todo esto se verá mejor plasmado en un mapa, pues nos dará la ruta que se adjunta a continuación, donde el marcador negro con el icono de una casa será el punto de origen y el azul con un icono de bandera, el punto de destino:

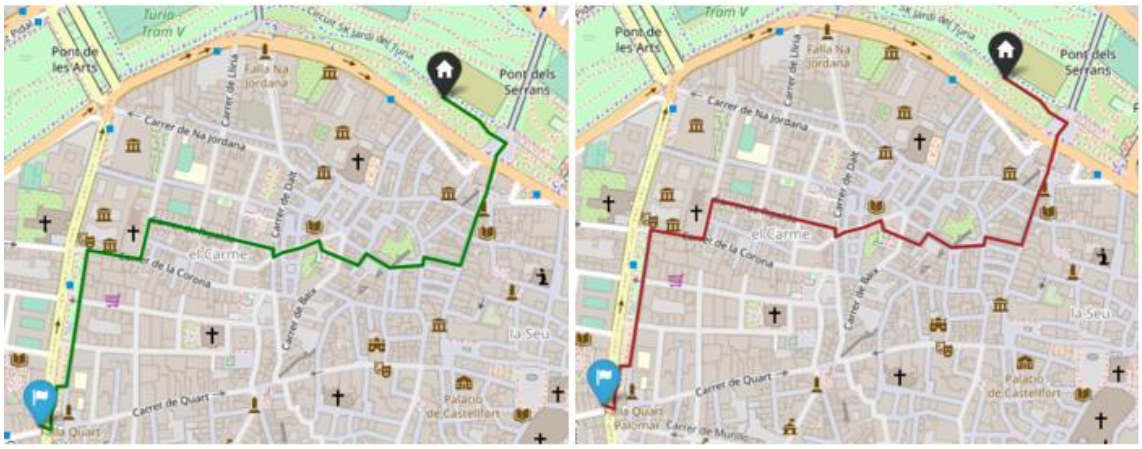


Ilustración 66. Ruta visual sobre mapa, zona verde (izquierda) y zona social (marrón). Fuente propia

[108] Como esta ruta no nos genera confianza por parecer demasiado diferente a la que seguiríamos según los criterios indicados, se realizarán pruebas alternativas a continuación.

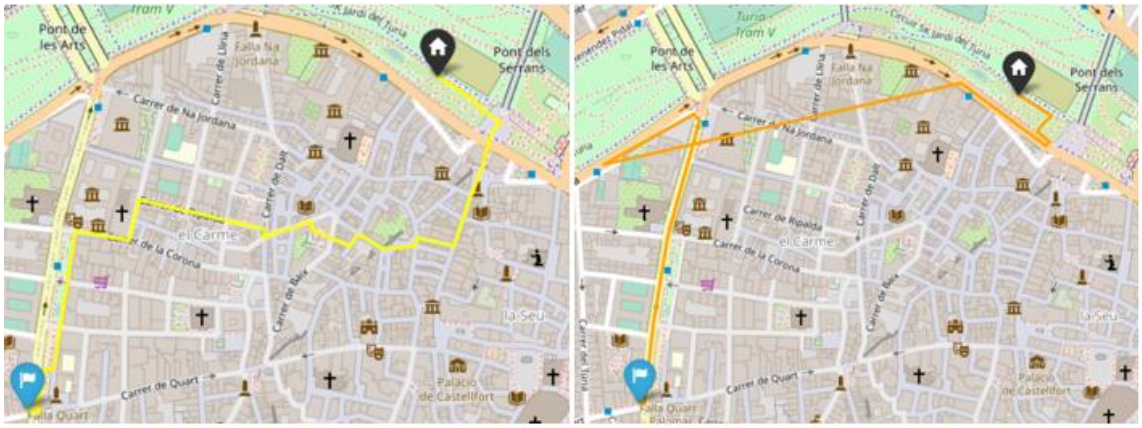


Ilustración 67. Ruta visual sobre mapa, zona cultural (izquierda) y zona de menor ruido (derecha). Fuente propia.

En comparacion con la ruta genérica, que sería aquella que solamente toma el valor de la longitud:

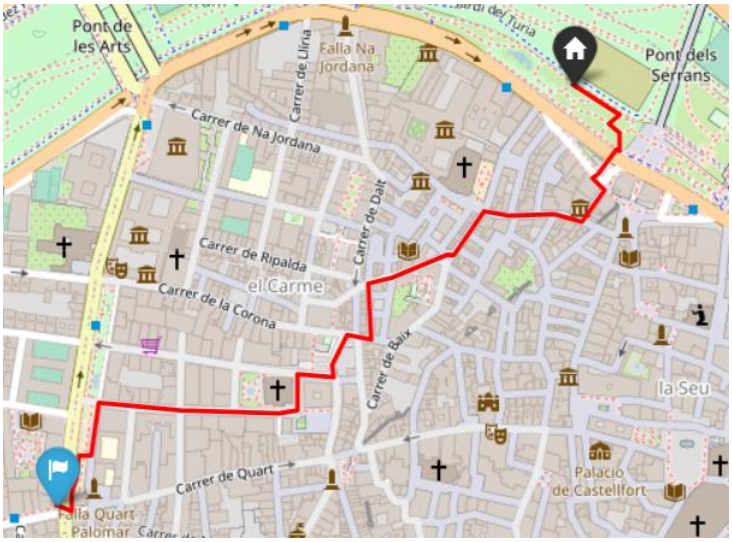


Ilustración 68. Ruta más corta sin parámetros añadidos, solo longitud. Fuente propia.

Como este resultado, sobre todo aquel basado en el ruido, nos parece un poco extraño, se procederá a invertir los valores de los índices, lo haremos como se ha indicado en el apartado anterior, hallando el máximo de cada conjunto de valores de cada índice en particular y restándolo al valor del índice que se analice en ese momento[109], para que así los valores sean más asequibles y el algoritmo los pueda utilizar de mayor peso a menor.

Obtendremos el siguiente resultado:

[109] Para la explicación y el script, acudir a la ilustración 54.

```
elegido zona verde
[14510, 17834, 22320, 21382, 22319, 21381, 18722, 1810, 18721, 22330, 1814, 16158, 1787, 2079, 2080, 2081,
2083, 1863, 1860, 1861, 2096, 1816, 1815, 1819, 1821, 1820, 1704, 1703, 1084, 1081, 1083, 552, 551, 526, 524,
525, 527, 528, 530, 531, 534, 535, 18731, 624, 18732]
elegida zona social
[14510, 17834, 22320, 21382, 22319, 21381, 18722, 1810, 18721, 22330, 1814, 16158, 1787, 2079, 2080, 2081,
2083, 1863, 1860, 1861, 2096, 1816, 1815, 1819, 1821, 1820, 1704, 1703, 1084, 1081, 1083, 552, 551, 526, 524,
525, 527, 528, 530, 531, 534, 535, 18731, 624, 18732]
elegida zona cultural
[14510, 17834, 22320, 21382, 22319, 21381, 18722, 1810, 18721, 22330, 1814, 16158, 1787, 2079, 2080, 2081,
2083, 1863, 1860, 1861, 2096, 1816, 1815, 1819, 1821, 1820, 1704, 1703, 1084, 1081, 1083, 552, 551, 526, 524,
525, 527, 528, 530, 531, 534, 535, 18731, 624, 18732]
elegida ruta más tranquila
[14510, 17834, 22320, 21382, 21378, 21380, 22316, 18722, 1810, 18721, 22330, 1814, 16158, 1787, 2079, 2080,
2081, 2083, 1863, 1860, 1861, 2096, 1816, 2098, 1818, 2099, 2100, 2101, 1726, 1702, 1715, 1695, 1688, 1691,
1682, 1692, 1679, 1677, 1675, 532, 1672, 1673, 18734, 19384, 21159, 1656, 22440, 18732]
no se ha elegido ruta particular
[14510, 17833, 21377, 21378, 21380, 22316, 18722, 1810, 18721, 22330, 1814, 16158, 1787, 2079, 2080, 2081,
2083, 1863, 1860, 1861, 2096, 1816, 2098, 1818, 2099, 2100, 2101, 1726, 2104, 2107, 1701, 1696, 1694, 1697,
1700, 1691, 1682, 1692, 1679, 1677, 1675, 532, 1672, 1674, 537, 19382, 19386, 21158, 21157, 18731, 624, 18732]
```

Ilustración 69. Resultado del algoritmo con los índices a la inversa. Fuente propia.

Donde, como vemos, de nuevo las tres primeras rutas son iguales entre sí, pero difieren de las halladas previamente; asimismo, la última ruta, la más directa, permanece igual, pero la ruta más tranquila es también diferente con respecto a las demás y con respecto al análisis anterior. Lo veremos a continuación, a modo de ruta visual:

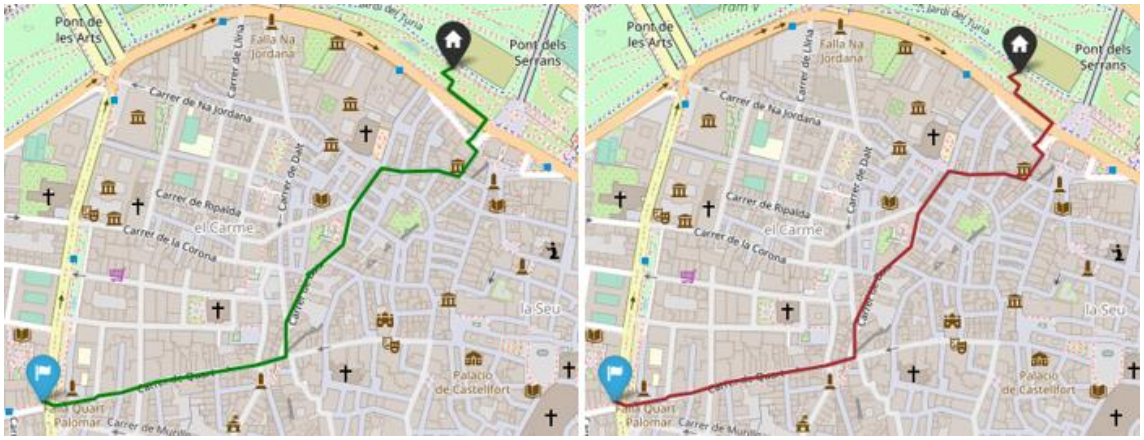


Ilustración 70. Rutas con índices a la inversa. Ruta verde (izquierda) y ruta social (derecha). Fuente propia.

Ilustración 71. Rutas con índices a la inversa. Ruta cultural (izquierda) y ruta de menor ruido (derecha). Fuente propia.

Los cambios no son significativos, salvo en el caso de la ruta más tranquila, que sí traza un camino más acorde al índice que hemos obtenido y sobre todo a la ponderación que pretendemos dar a los diferentes niveles de ruido, evitando la *Carrer de Guillem de Castro*, lo cual genera un recorrido que nos parece más fiable.

Y en la comparativa con respecto a la ruta más directa, que sigue igual que en el análisis anterior, veremos:

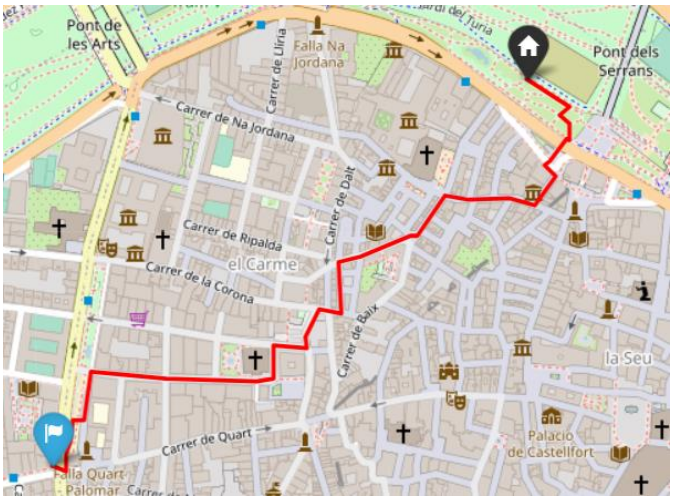


Ilustración 72. Ruta más directa, sin ponderaciones. Fuente propia.

Por ese motivo, dejaremos los valores de los índices a la inversa.

Desde el inicio del proyecto, la mayoría de las pruebas se han basado en la zona de Ciutat Vella, pero al tener registros y haber obtenido los datos de toda la ciudad, realizaremos

diversas pruebas[110] que engloben el total de la zona, València. Los parámetros de entrada de la función son los que figuran en la tabla adjunta a continuación[111]:

Zona	FIDfrom	FIDto	Color
Benimaclet	6843	10916	Rojo
Poblats Marítims	14345	14285	Verde
L'Eixample	80	4211	Azul
Extramurs	4813	17502	Morado

Tabla 7. Zonas donde generar las rutas de comprobación. Fuente propia.

A continuación, vamos a probar a generar este tipo de rutas alternativas en diferentes partes de València, para ver qué influencia pueden tener ciertas zonas, si la hubiera, y para poder extraer unas conclusiones válidas del análisis y de todo el desarrollo realizado en base a los resultados obtenidos. Basándonos en la tabla anterior, cuyo campo *Color* nos indica qué color tiene el círculo que engloba la zona, para localizarla mejor, en el siguiente mapa basado en nuestros tramos, nodos y datos extraídos de OSM.

Ilustración 73. Señalización de las zonas donde se trazarán las rutas. Fuente propia.

[110] Se adjuntarán aquí un total de cuatro, cada una en una zona diferente de València, aunque hemos realizado más para tomar las más significativas, aquellas que impliquen cambios o desvíos.
[111] Los nodos de referencia se tomarán desde el SIG de escritorio para que sea más sencillo, mantendremos los colores de las rutas para que resulten más identificativas.

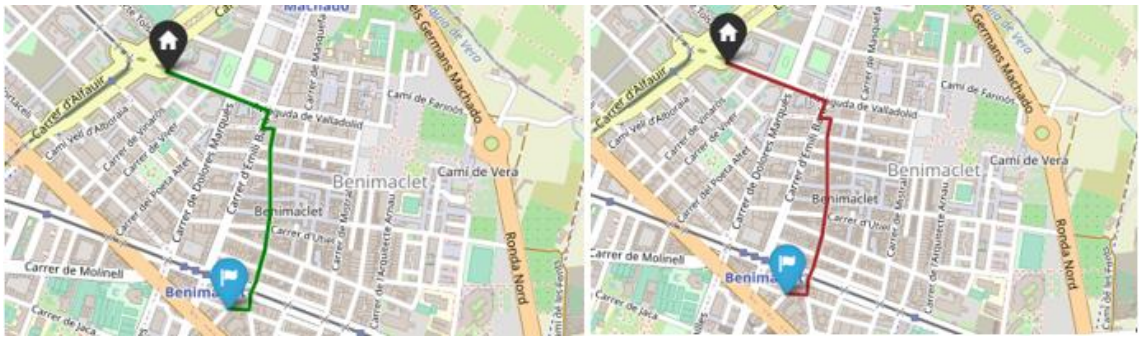


Ilustración 74. Ruta Benimaclet, zona verde (izquierda) y cultural (derecha), idénticas. Fuente propia.

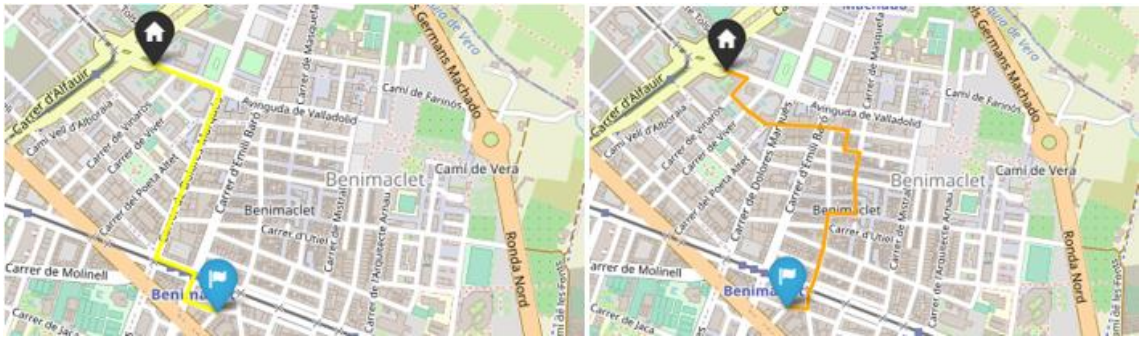


Ilustración 75. Ruta Benimaclet, zona social (izquierda) y de menor ruido (derecha), diferentes entre sí y a las anteriores. Fuente propia.

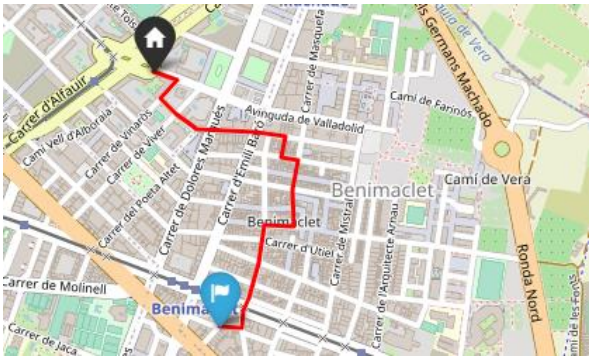


Ilustración 76. Ruta Benimaclet, más corta, sin parámetros añadidos. Fuente propia.

Benimaclet es un área conocida por su gran oferta de bares, restaurantes y pubs, es decir, zonas sociales, como podemos ver en el desvío que toma la ruta a través de la calle más principal, *Carrer de Dolores Marqués*, donde hay gran oferta gastronómica. En este caso la ruta de menor ruido y la ruta más rápida son coincidentes, pues salgo las dos avenidas más principales, es una zona tranquila, con una media baja de ruido; la ruta verde y la ruta cultural en este caso también coinciden, pero no la social con ninguna de las demás.

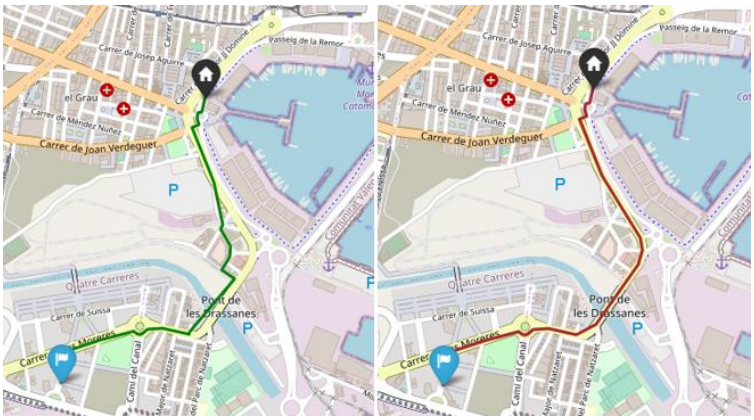


Ilustración 77. Ruta Poblats Marítims, zona verde (izquierda) y cultural (derecha), diferentes entre sí. Fuente propia.

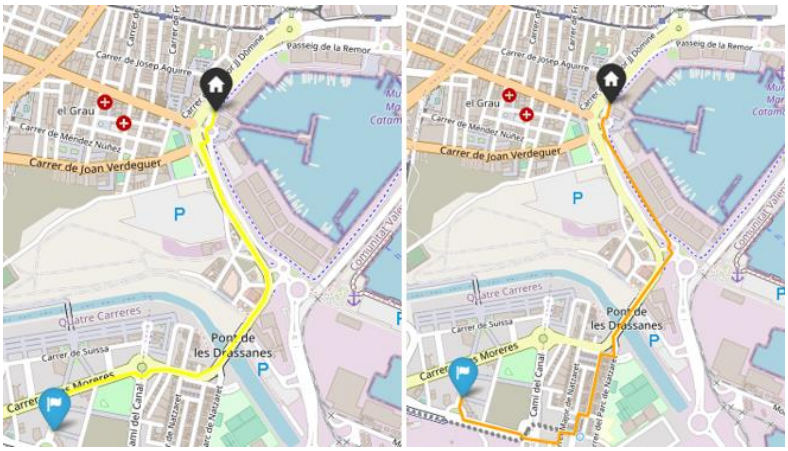


Ilustración 78. Ruta Poblats Marítims, zona social (izquierda) y de menor ruido (derecha), diferentes. Fuente propia.

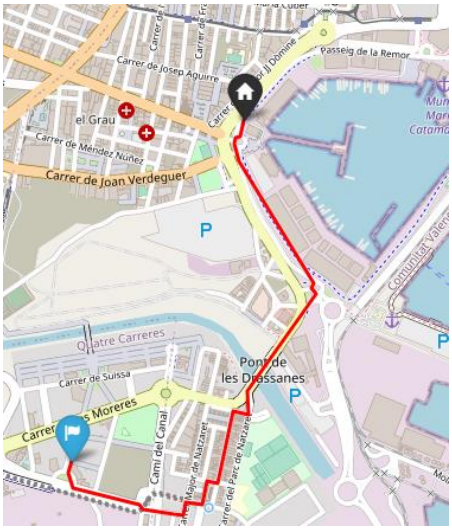


Ilustración 79. Ruta Poblats Marítims, distancia más corta, sin parámetros. Fuente propia.

La zona de Poblats Marítims es aquella cercana al mar, y en la ruta que se plantea se intenta evitar tanto en la elección de ruta verde como en la elección de ruta de menor ruido la que aparece como avenida predominante en las ilustraciones, *Carrer del Doctor JJ Dòmine*, por

considerarse una zona de mucho tráfico y demasiado ruido. Así, se consiguen generar tres rutas diferentes, una para las zonas verdes, otra para las zonas sociales y culturales, que es igual, y una última para la ruta de menor ruido y la más directa o rápida, basada solamente en la longitud de los tramos, que también son iguales entre sí[112].

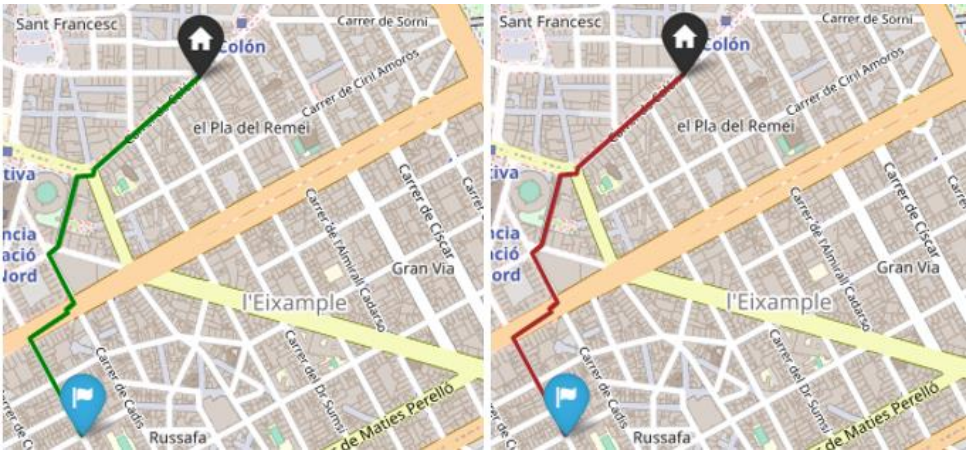


Ilustración 80. Ruta L'Eixample, zona verde (izquierda) y zona cultural (derecha), coincidentes. Fuente propia.

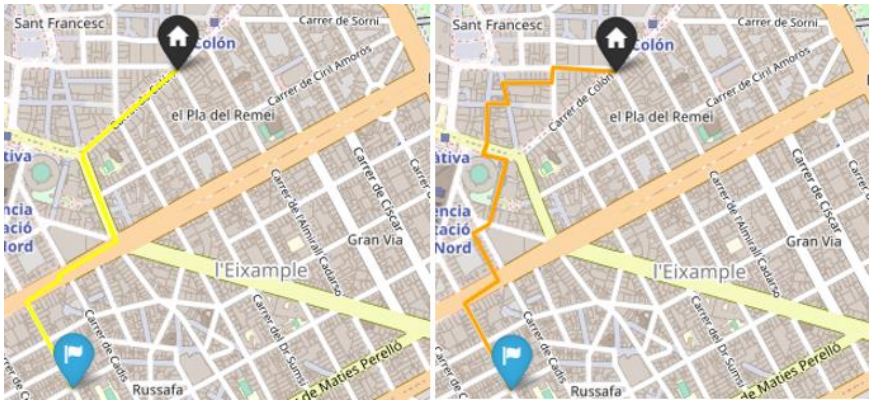


Ilustración 81. Ruta L'Eixample, ruta social (izquierda) y de menor ruido (derecha), no coincidentes. Fuente propia.

[112] Este es un patrón que vemos repetirse en todas las rutas trazadas, que han sido muchas. La ruta cultural siempre coincide con alguna otra ruta, y la ruta de menor ruido siempre suele coincidir con la basada en la longitud.

Ilustración 82. Ruta L'Eixample sin parámetros, solo con la longitud. Fuente propia.

Nuevamente, en este ejemplo vemos como la ruta que no coincide es la social, después de las numerosas pruebas realizadas no se ha encontrado una ruta no coincidente entre la ruta verde y la ruta cultural, ni tampoco se han hallado diferencias entre la ruta obtenida con el criterio de menor ruido y utilizando únicamente la longitud. Se han encontrado varias donde solo difiere la ruta social, de nuevo, un total de tres rutas (verde y cultural, social, y menor ruido y más corta).

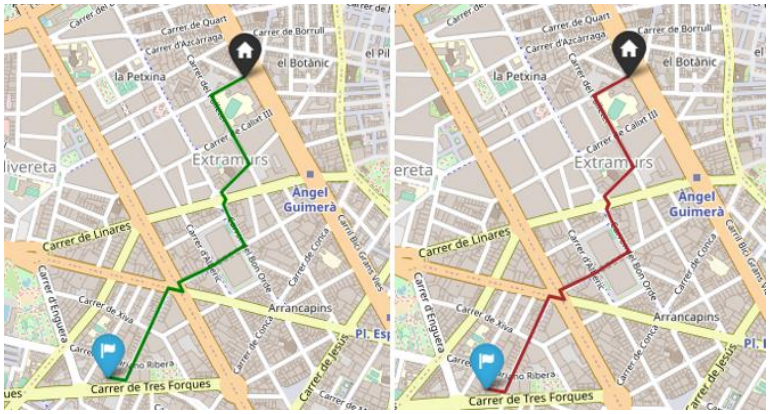


Ilustración 83. Ruta Extramurs, ruta verde (izquierda) y ruta cultural (derecha), coincidentes. Fuente propia.

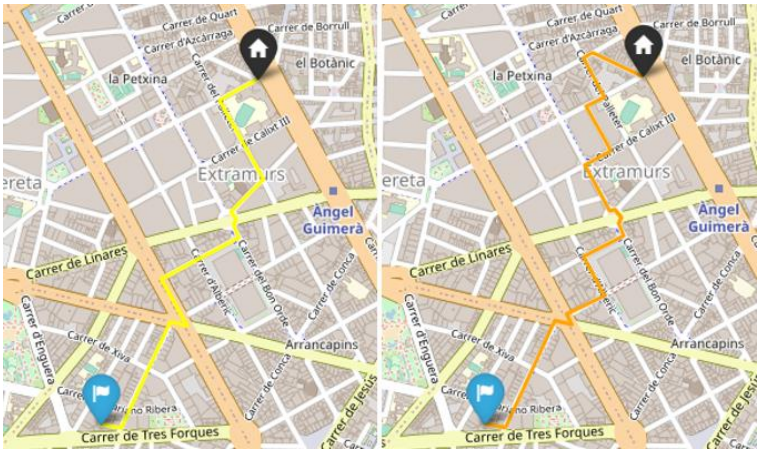


Ilustración 84. Ruta Extramurs, ruta social (izquierda) y de menor ruido (derecha), no coincidentes. Fuente propia.

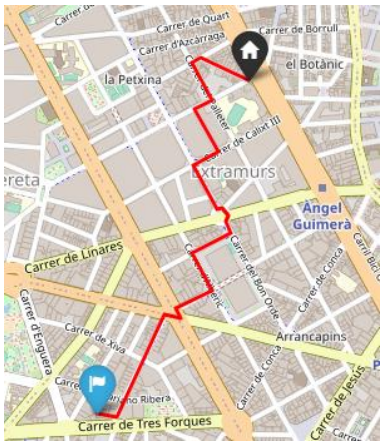


Ilustración 85. Ruta Extramurs, ruta más corta sin otros criterios. Fuente propia.

Nuevamente en la zona de Extramurs, que es la última área delimitada sobre la cual realizar la comprobación de rutas, volvemos a encontrarnos con el mismo problema: las rutas generadas son coincidentes, todas ellas, en cuanto al índice de verdor y cultural, pues sus valores son similares, ya que el baremo en que nos movemos es parecido; sin embargo, siempre destacan las rutas sociales, sobre las cuales quizás disponemos de más información.

CONCLUSIONES

Una vez se ha desarrollado todo el trabajo y basándonos en los resultados obtenidos, partiendo siempre de la hipótesis planteada en el apartado inicial, habiendo cubierto completamente los objetivos que se abordaban al principio de nuestro programa, nos basaremos en los resultados visibles en el apartado *Resultados* para poder concretar las conclusiones a las que hemos llegado con este proyecto.

En primer lugar, podemos decir que las rutas generadas e incluidas en el apartado *Resultados* no son el total de rutas sobre las cuales hemos realizado comprobaciones, pero sí los resultados medios de todas ellas. Así, vemos que en general la ruta que sigue los criterios culturales no tiene nunca una disposición diferente a las demás, bien sea a la ruta verde o a la ruta social, se adapta a una de las dos; esto puede deberse a la falta de datos que se pueden dar bien derivados de las etiquetas que hemos elegido para obtener los diferentes elementos o bien a la propia carencia de OpenStreetMap, puesto que en algunos casos hemos contrastado que, sobre todo en cuanto a zonas verdes y elementos culturales, no nos devuelve toda la información solicitada, como podemos ver en los siguientes ejemplos:

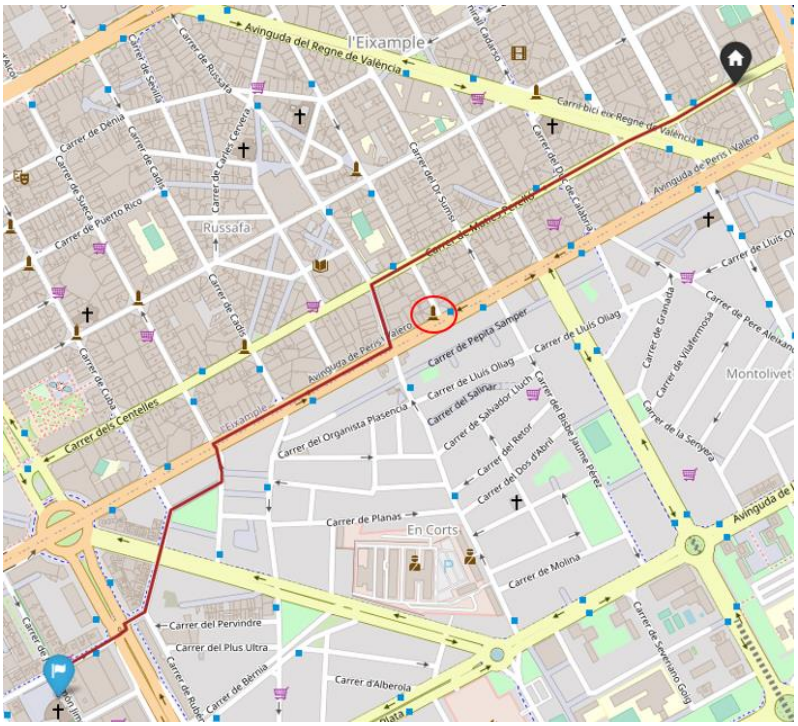


Ilustración 86. Ruta cultural que no pasa por un elemento cultural. Fuente propia.

Este primer error detectado, visible en la ilustración superior, corresponde a una ruta cultural, en color marrón, que, teniendo la posibilidad de pasar por un elemento cultural perteneciente a la categoría de arte al aire libre, o *artwork* dentro de nuestras etiquetas, no lo hace, quizá porque el elemento no está catalogado como tal y por ese motivo el programa principal de obtención de datos no lo rescata o bien porque ni siquiera figura en la base de datos y, por tanto, es imposible que lo podamos obtener.

Ilustración 87. Ruta verde que no pasa por la zona verde. Fuente propia.

En segundo lugar, tenemos esta imagen adjunta arriba, donde encontramos como zona verde aquella que se encuentra en la *Gran Vía de Ferran el Catòlic*, cuya mediana, es decir, el espacio entre los carriles que separa las dos direcciones de la vía, está catalogado como zona verde, un parque, concretamente, pero la ruta verde ni siquiera hace amago de pasar por esta calle, más bien al contrario[113]

Evidentemente, aquellos elementos que no figuren en la base de datos de la que estamos extrayendo toda esta información, no aparecerán en nuestros resultados. Este quizá pueda ser uno de los motivos por los que los índices de verdor y cultural se mueven en base a baremos tan bajos, y en muchas ocasiones coinciden completamente entre sí, ya que hemos realizado numerosas pruebas por toda el área de València y siempre hemos encontrado el mismo resultado, llegando, de este modo, a la misma conclusión.

Por otro lado, la ruta de menor ruido y la ruta más corta, basada únicamente en la longitud de los tramos, también suelen coincidir de manera recurrente, como hemos podido ver en las rutas descritas y adjuntas en el apartado *Resultados*. Así, finalmente, obteníamos un total de tres rutas en todos los supuestos de desplazamiento que se planteaban:

- Una primera ruta, con la zona cultural coincidente con la social o la de verdor.
- Una segunda ruta, individual, del camino según el índice de verdor o social.
- Una tercera ruta, con el camino de menor ruido y el basado en la longitud coincidentes en todo el recorrido.

Quizá pueda resultar conveniente unificar los datos con fuentes extra, o ampliar el rango de etiquetas elegidas para poder obtener más elementos, aunque los que vemos en las ilustraciones 86 Y 87 deberían estar en nuestros datos ya que uno es *tourism:artwork* y el otro pertenece a *leisure:garden*.

[113] Lo mismo ocurre en la intersección entre esta misma vía y el *Passeig de la Petxina*, donde encontramos el Jardín de las Hespérides, que no aparece como punto de paso en ninguna ruta que hayamos comprobado, aunque hemos rescatado todos aquellos elementos que se engloban dentro de la etiqueta *garden*.

Es también importante el área de análisis, y hemos de tener en cuenta que estamos basándonos en una ciudad española, donde habitualmente las zonas de restauración, sean de comida o bebida, siempre abundan en cualquier área elegida, motivo por el cual quizá las rutas generadas con el indicador social sean diferentes en mayor proporción a las culturales o de zonas verdes.

También se ha visto que el criterio de la distancia total de la ruta trazada, es decir, la distancia que hay entre el nodo de origen y el nodo de destino establecido por el usuario, no resulta determinante a la hora de que las rutas difieran entre sí. Tanto en rutas cortas, con origen y destino muy cercanos entre sí, como en rutas largas, que cruzan prácticamente media ciudad, los resultados obtenidos son los mismos.

BIBLIOGRAFÍA

Abdishakur, A. (2019, 9 diciembre). *Retrieving OpenStreetMap data in Python - Towards Data Science. Towards Data Science*. https://towardsdatascience.com/retrieving-openstreetmap-data-in-python-1777a4be45bb

Abdishakur, A. (2020, 4 junio). *Routing street networks: Find your way with Python*. Shakasom. https://shakasom.medium.com/routing-street-networks-find-your-way-with-python-9ba498147342

Algoritmo de Dijkstra. (2016, 25 mayo). Universidad INCCA de Colombia. https://jcrd0730.wixsite.com/estr/single-post/2016/05/25/algoritmo-de-dijkstra-1

Anaconda | Individual Edition. (2012, 6 septiembre). Anaconda. https://www.anaconda.com/products/individual

Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart, "*Exploring network structure, dynamics, and function using NetworkX", in Proceedings of the 7th Python in Science Conference* (SciPy2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11–15, Aug 2008

ArcGIS Web Application. (2019, 18 diciembre). Mapa de Soroll/Mapa de Ruido. https://geoportal.valencia.es/portal/apps/webappviewer/index.html?id=056f38a5f7e8414b81f03617900b4e67&mobileBreakPoint=300

Barton, J.; Hine, R.; Pretty, J. *The health benefits of walking in greenspaces of high natural and heritage value*. J. Integr. Environ. Sci. 2009, 6, 261–278.

Bienvenido al proyecto QGIS! (s. f.). QGIS. Recuperado 31 de agosto de 2021, de https://www.qgis.org/es/site/

Boeing, G. (2017, 8 enero). *OSMnx*: *New Methods for Acquiring, Constructing, Analyzing, and Visualizing Complex Street Networks*. Geoff Boeing. Computers, Environment and Urban Systems 65, 126-139. https://geoffboeing.com/publications/osmnx-complex-street-networks/

Carbassot. (2020, 11 mayo). *Find name of a street between 2 nodes. OSMnx*. Stack Overflow. https://stackoverflow.com/questions/61735149/find-name-of-a-street-between-2-nodes-osmnx

Colaboradores de Wikipedia. (2016, 28 agosto). *QGIS*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/QGIS

colaboradores de Wikipedia. (2017, 28 agosto). *Python*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Python

Colaboradores de Wikipedia. (2019, 2 agosto). *GDAL*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/GDAL

colaboradores de Wikipedia. (2019, 18 septiembre). *ABC (lenguaje de programación)*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/ABC_(lenguaje_de_programaci%C3%B3n)

Colaboradores de Wikipedia. (2020, 10 septiembre). *ArcGIS*. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/ArcGIS

Connors, L. (2021, 15 junio). *Creating a Simple Map with Folium and Python - Towards Data Science*. Towards Data Science. https://towardsdatascience.com/creating-a-simple-map-with-folium-and-python-4c083abfff94

Conversor de GEOJSON a Shapefile. (2018, 1 marzo). Gis&Beers. http://www.gisandbeers.com/conversor-de-geojson-a-shapefile/

Las coordenadas geográficas de València. La latitud, la longitud y la altitud sobre el nivel del mar de València, EspaÃ±a. (s. f.). Dateandtime. Recuperado 1 de septiembre de 2021, de https://dateandtime.info/es/citycoordinates.php?id=2509954

Creando hermosos mapas con Python. (2021, 1 febrero). ICHI.PRO. https://ichi.pro/es/creando-hermosos-mapas-con-python-121060872996188

Dataset: Mapa Ruido Lden 24h. (2019, 18 diciembre). Mapa Ruido Lden 24h. https://www.valencia.es/dadesobertes/es/dataset/?id=mapa-soroll-lden-24-h

De JSON a entidades (Conversión)—ArcGIS Pro | Documentación. (s. f.). ArcGIS Pro. Recuperado 1 de septiembre de 2021, de https://pro.arcgis.com/es/pro-app/latest/tool-reference/conversion/json-to-features.htm

Develop - OpenStreetMap Wiki. (2007). Develop OSM. https://wiki.openstreetmap.org/wiki/Develop

Drawing — NetworkX 2.6.2 documentation. (s. f.). NetworkX. Recuperado 2 de septiembre de 2021, de https://networkx.org/documentation/stable/reference/drawing.html

Fiona 1.8.20. (2021, 1 junio). PyPI. https://pypi.org/project/Fiona/

GeoJson Test Page. (s. f.). GeoJSON Test. Recuperado 31 de agosto de 2021, de https://utahemre.github.io/geojsontest.html

Grafos | Qué son, tipos, orden y herramientas de visualización. (2021, 11 febrero). GraphEverywhere. https://www.grapheverywhere.com/grafos-que-son-tipos-orden-y-herramientas-de-visualizacion/

Gwerder, S. (s. f.). *OSM TagFinder*. Tag Finder. Recuperado 31 de agosto de 2021, de http://tagfinder.herokuapp.com/

Holtz, Y. (2018). *Map with markers with Python and Folium*. The Python Graph Gallery. https://www.python-graph-gallery.com/312-add-markers-on-folium-map/

Home — Spyder IDE. (s. f.). © 2020 Spyder Website Contributors. Recuperado 31 de agosto de 2021, de https://www.spyder-ide.org/

How to create a simple map (with a marker) using Leaflet? (s. f.). OpenStreetMap Belgium. Recuperado 2 de septiembre de 2021, de https://openstreetmap.be/en/projects/howto/leaflet.html

Janakiev, N. (2018, 17 agosto). *Loading Data from OpenStreetMap with Python and the Overpass API*. Towards Data Science. https://towardsdatascience.com/loading-data-from-openstreetmap-with-python-and-the-overpass-api-513882a27fd0

Key:building - OpenStreetMap Wiki. (s. f.). OpenStreetMap - Wiki. Recuperado 2 de septiembre de 2021, de https://wiki.openstreetmap.org/wiki/Key:building

Key:landuse - OpenStreetMap Wiki. (s. f.). OpenStreetMap - Wiki. Recuperado 2 de septiembre de 2021, de https://wiki.openstreetmap.org/wiki/Key:landuse

Key:leisure - OpenStreetMap Wiki. (s. f.). OpenStreetMap - Wiki. Recuperado 2 de septiembre de 2021, de https://wiki.openstreetmap.org/wiki/Key:leisure

Leaflet - OpenStreetMap Wiki. (2007). Leaflet OSM. https://wiki.openstreetmap.org/wiki/Leaflet

Map features - OpenStreetMap Wiki. (s. f.). Map Features - Open Street Map. Recuperado 31 de agosto de 2021, de https://wiki.openstreetmap.org/wiki/Map_features

Mapshaper. (s. f.). Mapshaper. Recuperado 8 de septiembre de 2021, de https://mapshaper.org/

Martinez, J. C. (2021, 23 abril). *How to plot your data on maps using Python and Folium*. Live Code Stream. https://livecodestream.dev/post/how-to-plot-your-data-on-maps-using-python-and-folium/

Martínez, S. (2020b, octubre 7). *Qué es Open Street Maps y cómo descargar sus datos. Cursos GIS* | TYC GIS Formación. https://www.cursosgis.com/que-es-open-street-maps-y-como-descargar-sus-datos/

El módulo Python networkx encuentra la ruta más corta de dijkstra - programador clic. (s. f.). Programador Clic. Recuperado 2 de septiembre de 2021, de https://programmerclick.com/article/7683597054/

Morales, A. (2019, 28 enero). *Primeros pasos con QGIS Server*. MappingGIS. https://mappinggis.com/2014/03/primeros-pasos-con-qgis-server/

Moelter, A.; Lindley, S. *Influence of walking route choice on primary school children's exposure to air pollution—A proof of concept study using simulation. Sci. Total Environ*. 2015, 530–531, 257–262.

Muhammad, A. (2020, 7 marzo). *Is it possible to draw paths in Folium?* Stack Overflow. https://stackoverflow.com/questions/60578408/is-it-possible-to-draw-paths-in-folium

Navegador independiente de QGIS. (s. f.). QGIS Browser. Recuperado 1 de septiembre de 2021, de https://docs.qgis.org/2.18/es/docs/user_manual/qgis_browser/qgis_browser.html

Networkx.algorithms.shortest_paths.generic.shortest_path — NetworkX 2.6.2 documentation. (s. f.). NetworkX. Recuperado 2 de septiembre de 2021, de https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.shortest_paths.generic.shortest_path.html#networkx.algorithms.shortest_paths.generic.shortest_path

Nicolson, N. (2020, 25 marzo). *Display a networkx graph on a leaflet map*. GitHub Gist. https://gist.github.com/nickynicolson/d972c5fb9d1972692342448d48ce029c

Novack, T. (2018, 6 noviembre). *A System for Generating Customized Pleasant Pedestrian Routes Based on OpenStreetMap Data*. Sensors. https://www.mdpi.com/1424-8220/18/11/3794/htm

Oakley, M. (2020, 20 abril). *Earth Data Analytics Online Certificate*. Earth Data Science - Earth Lab. https://www.earthdatascience.org/tutorials/introduction-to-leaflet-animated-maps/

Ogre - ogr2ogr web client. (s. f.). Ogre. Recuperado 2 de septiembre de 2021, de https://ogre.adc4gis.com/

OpenStreetMap. (2006, abril). OpenStreetMap. https://www.openstreetmap.org/

OpenStreetMap Taginfo. (2006). Open Street Map TagInfo. https://taginfo.openstreetmap.org/

OpenStreetMap Wiki. (2006). OpenStreetMap Wiki. https://wiki.openstreetmap.org/wiki/Main_Page

Overpass Turbo. (2019, 18 junio). Overpass Turbo. https://overpass-turbo.eu/

Pham, K. (2017, 29 agosto). *Web Mapping with Python and Leaflet*. Programming Historian. https://programminghistorian.org/en/lessons/mapping-with-python-leaflet

Poom, A., Helle, J., & Toivonen, T. (2020, junio). *Journey Planners can promote active, healthy and sustainable urban travel*. URBARIA, Helsinki Institute of Urban and Regional Studies. https://www2.helsinki.fi/sites/default/files/atoms/files/age_poom_joose_helle_tuuli_toivonen_journey_planners_can_promote_active_healthy_and_sustainable_urban_travel.pdf

Python osmnx - How to download a city district map from OpenStreetMap based on the boundary ID? (2020, 29 noviembre). Stack Overflow. https://stackoverflow.com/questions/65064351/python-osmnx-how-to-download-a-city-district-map-from-openstreetmap-based-on-t

Qué son los grafos. (2020, 10 marzo). GraphEverywhere. https://www.grapheverywhere.com/que-son-los-grafos/

Retrieving OpenStreetMap data — AutoGIS site documentation. (s. f.). AutoGIS. Recuperado 2 de septiembre de 2021, de https://automating-gis-processes.github.io/site/notebooks/L6/retrieve_osm_data.html

Schaefd. (2018, 3 octubre). *Python: Visualizing social network with Networkx and Basemap*. GitHub. https://tuangauss.github.io/projects/networkx_basemap/networkx_basemap.html

Siriaraya, P., et al. (2020, 27 julio). *Beyond the shortest route: a survey on quality-aware route navigation for pedestrians*. in IEEE Transactions on Intelligent Transportation Systems. https://ieeexplore.ieee.org/ielx7/6287639/8948470/09149593.pdf

Software for Complex Networks — NetworkX 2.6.2 documentation. (s. f.). NetworkX Network Analysis in Python. Recuperado 27 de agosto de 2021, de https://networkx.org/documentation/stable/index.html

Solanki, S. (2020, 16 septiembre). *Ipyleaflet [Python] - Interactive Maps in Python based on leafletjs by Sunny Solanki*. CoderzColumn. https://coderzcolumn.com/tutorials/data-science/ipyleaflet-interactive-maps-in-python-based-on-leafletjs

Tags - OpenStreetMap Wiki. (s. f.). Tags - Open Street Map. Recuperado 31 de agosto de 2021, de https://wiki.openstreetmap.org/wiki/Tags

Tenkanen, H. (2017, 18 diciembre). *New feature: Retrieving OSM POIs with osmnx? · Issue #116 · gboeing/osmnx*. GitHub. https://github.com/gboeing/osmnx/issues/116

Universidad De Colombia. (2016, 25 mayo). *Algoritmo de Dijkstra*. Universidad INCCA de Colombia. https://jcrd0730.wixsite.com/estr/single-post/2016/05/25/algoritmo-de-dijkstra-1

User reference — OSMnx 1.1.1 documentation. (s. f.). OSMnx. Recuperado 2 de septiembre de 2021, de https://osmnx.readthedocs.io/en/stable/osmnx.html

Wang, Z.,Novack, T., Yan, Y., & Zipf, A., *Quiet Route Planning for Pedestrians in Traffic Noise Polluted Environments*, in IEEE Transactions on Intelligent Transportation Systems, doi: 10.1109/TITS.2020.3004660

Weighted Graph — NetworkX 2.7rc1.dev0 documentation. (s. f.). NetworkX. Recuperado 2 de septiembre de 2021, de https://networkx.org/documentation/latest/auto_examples/drawing/plot_weighted_graph.html#sphx-glr-auto-examples-drawing-plot-weighted-graph-py

Welcome to Python. (s. f.). Python. Recuperado 31 de agosto de 2021, de https://www.python.org/

Welcome to Python Overpass API's documentation! — Python Overpass API 0.6 documentation. (2012). Python Overpass API. https://python-overpy.readthedocs.io/en/latest/

Welcome to Spyder's Documentation — Spyder 5 documentation. (s. f.). Spyder's Documentation. Recuperado 31 de agosto de 2021, de https://docs.spyder-ide.org/current/index.html

Wu, Q. (2021, 3 septiembre). *Leafmap - A Python package for geospatial analysis and interactive mapping in a Jupyter environment. | PythonRepo*. PythonRepo. https://pythonrepo.com/repo/giswqs-leafmap-python-geolocation

ANEXOS

I. Anexo de búsqueda de etiquetas.

La búsqueda de las etiquetas de este proyecto ha sido superficial, es decir, nos hemos intentado basar en zonas que, de manera neutra y objetiva, cualquier persona pudiese encontrar de interés. Asimismo, es bien sabido que cada individuo tendrá unas preferencias u otras, por ese motivo no se ha entrado en materia más profunda en base a edificios religiosos, aunque hay numerosas maneras de que estos sean identificados. En el caso de España, que, pese a ser un país aconfesional, acoge en su seno -dada su historia, su evolución y su religión principal hasta años muy recientes- numerosos edificios religiosos[114] dedicados a la religión católica, más concretamente, cristiana, tendríamos, por ejemplo, las siguientes opciones:

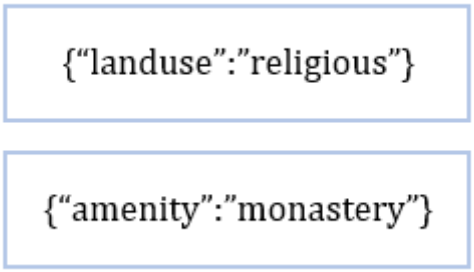


Ilustración 88. Otras opciones de búsqueda. Fuente propia.

En cuanto a las zonas verdes, la ciudad de València es conocida por saberlas alternar de manera urbana, introduciéndolas así en la evolución urbanística de la ciudad misma. Aparte de parques o jardines, podríamos encontrar numerosas etiquetas destinadas a la búsqueda de información sobre estas áreas, tanto dedicadas al deporte[115], como pueden ser campos de golf o de hípica, a la pesca o a la agricultura. Un ejemplo de esto podrían ser las siguientes etiquetas indicativas:

[114] Toda la información está disponible en la Wiki de OpenStreetMap: https://wiki.openstreetmap.org/wiki/Key:building en el apartado *Religious*, y en https://wiki.openstreetmap.org/wiki/Key:leisure, en aquellas zonas cuyo uso de suelo queramos conocer, que puede ser *religious* o, por ejemplo, *cemetery*.

[115] Todo ello se encuentra accesible en la página de Wiki OpenStreetMap: https://wiki.openstreetmap.org/wiki/Key:landuse, con aquellas zonas cuyos usos de suelo se dedican a objetivos determinados, como plantaciones específicas (*vineyard)*, o en https://wiki.openstreetmap.org/wiki/Key:leisure, donde podemos encontrar la actividad en concreto o el conjunto de actividades que queramos practicar (*Stadium*, *swimming_area*).

Ilustración 89. Otros tipos de zonas verdes. Fuente propia.

También en cuanto a zonas culturales podríamos haber extendido más la búsqueda[116], pues podríamos encontrar también teatros, que no aparecen como atracción turística, o edificios que por su fachada puedan ser identificados como zona de interés turístico, o incluso zonas de baile o gimnasios en cuanto a zonas de interés social, como podemos ver en los breves ejemplos adjuntos a continuación:

[116] Toda esta información procede de una búsqueda realizada in facto, pero que no ha sido llevada a cabo en el desarrollo final del programa ya que este proyecto solamente pretende dar una metodología aplicativa para una posterior, y posible, creación de un proyecto más práctico; así, esto solo pretende ser un trabajo de investigación, como ha sido indicado en el inicio.

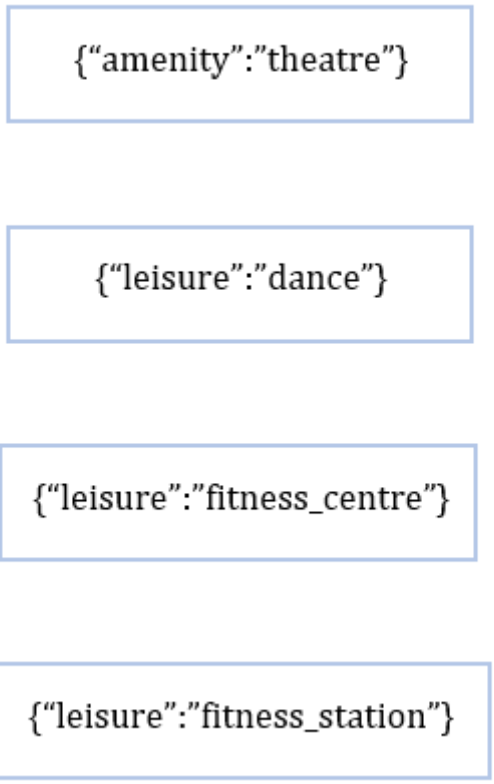


Ilustración 90. Otros tipos de zonas sociales

Incluso podríamos basar nuestro análisis en algo tan sencillo como una ruta que nos indique también las fuentes de agua que podemos encontrar, ya que València es una ciudad perfectamente diseñada para ser recorrida en vehículos menores y no motores, tales como bicicletas o patinetes, lo cual podría ser de ayuda para los descansos en la visita a esta ciudad:

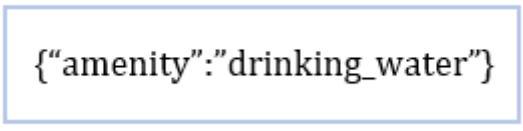


Ilustración 91. Otro ejemplo de la búsqueda. Fuente propia.

Además de otras etiquetas pertenecientes a las categorías que hemos seleccionado, tendríamos otras categorías disponibles, particularmente para la localización de zonas verdes, como por ejemplo la opción *natural*, que es utilizado para describir elementos físicos de la tierra, incluyendo aquellos que han tenido alguna intervención antrópica. Dentro de

esta categoría podríamos encontrar árboles -individuales[117] o en conjuntos de colectivos-, agua, tierra de regadío, líneas de costa, tierras no cultivadas, líneas de árboles, y elementos más propios de zonas rurales o, al menos, no urbanas.

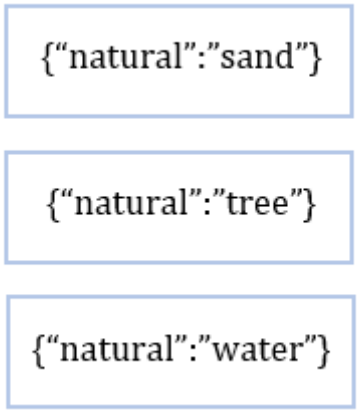


Ilustración 92. Ejemplos de tipos de suelo que podemos encontrar en OSM. Fuente propia.

Por todo lo explicado, y, en resumen, esto se basa en un trabajo de investigación orientativo, claro que lo ideal podría ser que cada individuo, sea cual sea su procedencia, su interés en la ciudad, la hora del día en que se ejecute la búsqueda de información y de ruta y, sobre todo, la preferencia personal del momento. Simplemente, con este anexo, se pretende indicar la cantidad de funcionalidades aplicativas reales que podemos encontrar con este método, que serían adaptables, si no a prácticamente toda la población[118], sí al menos al grueso poblacional mundial.

[117] Esta opción se ha descartado del análisis porque los árboles bordean habitualmente las avenidas más grandes, y entraría en conflicto con la variable de *Ruido* que hemos seleccionado.
[118] Al menos, toda aquella población que posea un dispositivo con conexión a internet.

II. Anexo de indicaciones de la metodología de análisis espacial aplicada en ArcGIS.

Con la intención de no sobrecargar de información que se pueda alejar del contenido principal y sin querer dejar de detallar cómo se ha conseguido el resultado final, este anexo explicativo seguirá paso a paso los métodos analíticos espaciales realizados con el software ArcGIS para llegar al objetivo, que era generar una capa que contuviese toda la información interrelacionada de las tres fuentes de datos que tenemos de partida.

I. Creación de un nuevo campo

Una vez hemos importado las capas de origen de datos a nuestro proyecto en el SIG de escritorio, que en este caso es ArcGIS[119], procederemos a las diversas modificaciones necesarias para llegar al resultado unificado. Como se explica durante el desarrollo del trabajo, en su apartado correspondiente, en primer lugar, necesitaremos tomar la capa obtenida del Ayuntamiento de València para añadir un nuevo campo de valores numéricos enteros, corto, pues solamente habrá una posición, y trasladar los datos que se encuentran en el campo *gridcode* al nuevo campo al que habremos nombrado *Ruido*. Con esta nomenclatura más clara se busca simplificar el posterior acceso a la información.

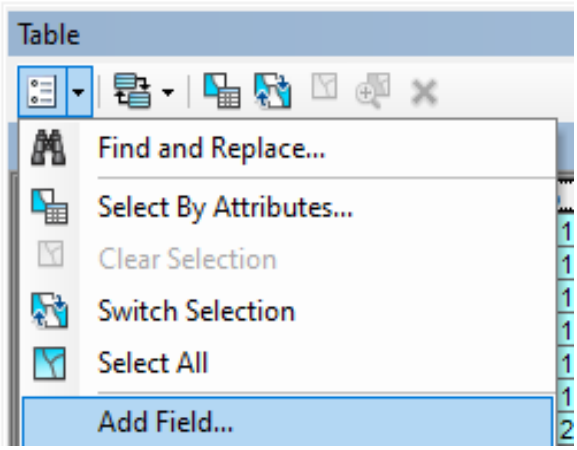


Ilustración 93. Añadir un nuevo campo a una tabla de atributos. Fuente propia.

[119] Se comenzó el desarrollo del trabajo con QGIS, como se puede comprobar, pero se ha trabajado más con ArcGIS durante el máster y por ello sus funcionalidades resultan más conocidas, además de ser más prácticas y eficientes.

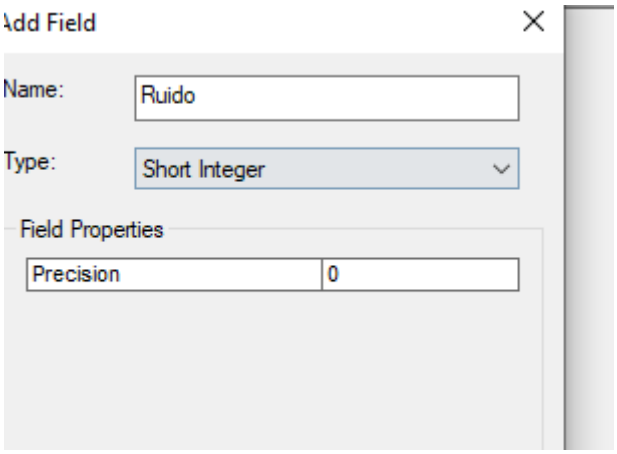


Ilustración 94. Particularidades del nuevo campo Ruido añadido. Fuente propia.

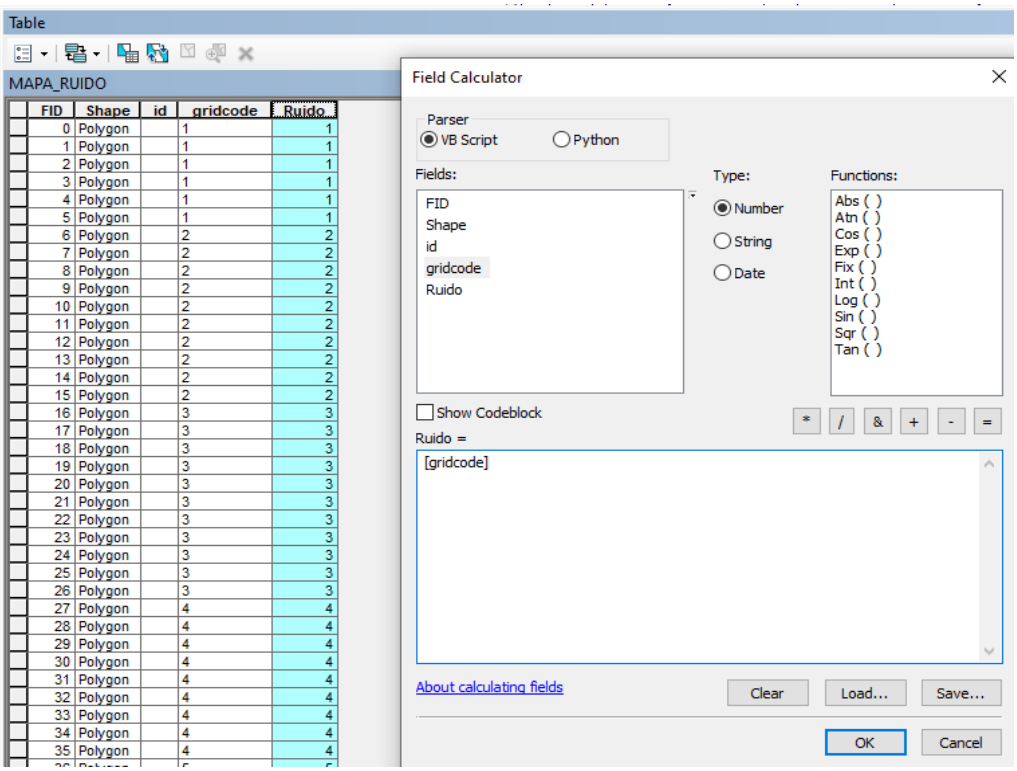


Ilustración 95. Valores del nuevo campo Ruido. Fuente propia.

Una vez tengamos esta información, ya podremos realizar la unión pertinente[120] entre las capas que necesitamos, que serán la de ruido y aquella que contiene los datos lineales de los segmentos o tramos de calles rescatados de OSM.

[120] En un primer lugar se realizó la función *Intersects (Analysis)*, pero esta nos dividía los tramos cada vez que intersecaban con un polígono diferente con distinto nivel de ruido, y dejaba esos tramos sin conexiones entre sí, pues los nodos descargados aludían al inicio y al final del tramo principal, no de las subdivisiones realizadas con este método. Por ese motivo, se descarta intersecar las capas y se elige unificarlas con la media del ruido de todo el tramo.

II. Join (Tool)

Los datos derivados del ruido por tramos (*MAPA_RUIDO*) y los tramos en sí descargados de OSM: esa será nuestra información de entrada. En la capa de salida, sin embargo, el resultado, será *Ruido_tramos*, una capa que se quedará virtual para posteriores análisis. Unificando esta información, obtendremos como resultado la media del ruido[121] de todo el segmento de la calle, para cada uno de los tramos, con el nodo de inicio y el de fin.

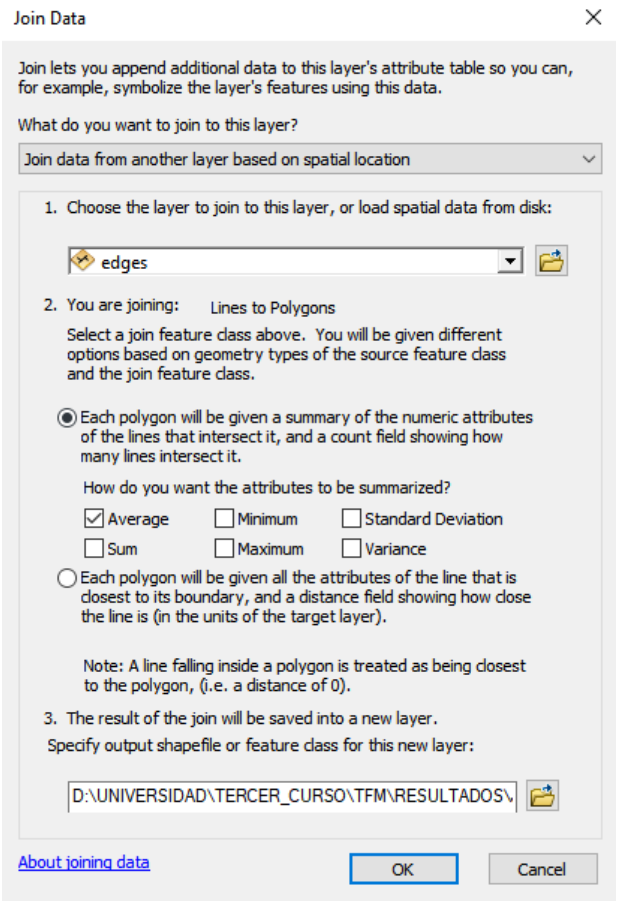


Ilustración 96. Unión entre los datos de tramos de calle y la media de ruido por tramo. Fuente propia.

Ahora que hemos obtenido una capa unificando esta información, tendremos el valor del ruido para cada tramo de una misma calle, como podemos ver en la ilustración adjunta, donde la media para la Plaza del Carme de ruido será de 1, esto es, uno de los valores más bajos en los niveles de ruido que podemos obtener, por lo cual, para el trazado de la ruta, será predominante este tipo de zonas si lo que buscamos es la tranquilidad y alejarnos del tráfico.

[121] Porque lo seleccionamos nosotros así ya que para este análisis es el mejor valor, *AVG*, pero hay múltiples opciones de cálculo en las uniones, como se puede ver en la ilustración 96.

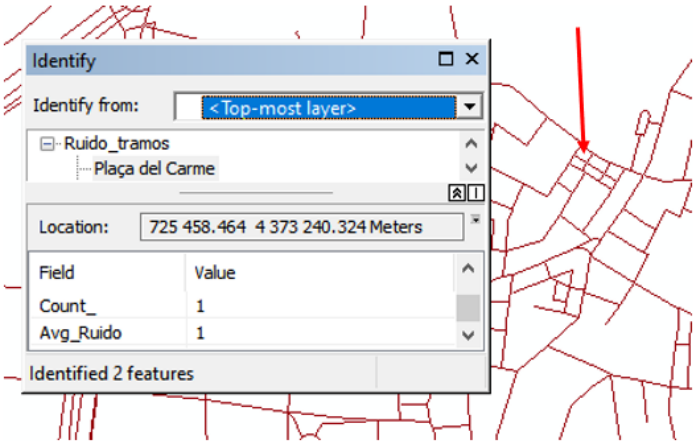


Ilustración 97. Resultado de la unión y la media de ruido por tramo. Fuente propia.

III. Buffer (Analysis)

Para poder ver qué elementos entran dentro del radio de nuestra calle, tendremos que realizar un buffer. Tras varias pruebas se determina que con un buffer de cinco metros es suficiente, y para poderlo realizar necesitaremos la capa de entrada que es la misma que el resultado de la intersección[122] y tendremos que determinar también los metros que queremos amplificar el radio[123].

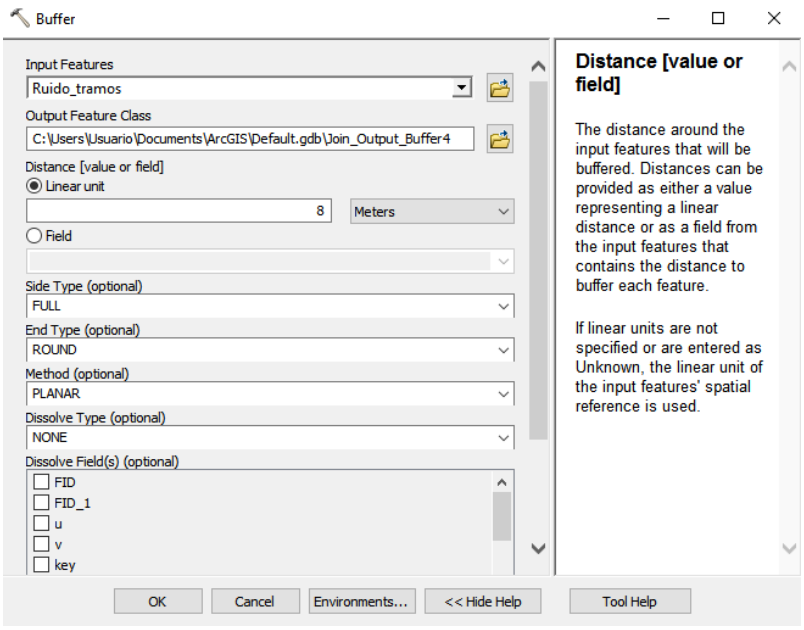


Ilustración 98. Realización del Buffer de 8 metros de la capa que contiene la media de ruido por tramos. Fuente propia.

[122] Es decir, la obtenida en el apartado a de este anexo.
[123] En la ilustración tomamos el buffer final, de 8 metros, pero también hemos realizado pruebas con 20, 10 y 5 metros, como puede verse en el Anexo 3.

En estas opciones también podemos elegir el tipo de buffer que queremos, y también si preferimos o no conservar todos los campos; nosotros, como hemos indicado previamente, no eliminaremos ninguno.

IV. Añadir los índices

Para añadir los índices seguiremos la misma estructura que en el apartado *a* de este mismo Anexo, cuando añadimos el campo *Ruido* para simplificar posteriores consultas[124]. Cabe destacar que la nomenclatura de estos campos ha de ser explicativa, ya que luego se modificará y es mejor poner la característica en primer lugar: *VerdorIndic* es mejor que *IndiceVerdor*, ya que luego al gestionar el siguiente apartado y realizar el *Join*, nos modificará el nombre por *Sum_Indice* y tendremos tres campos nombrados iguales, sin poder diferenciarlos.

El resultado de añadir estos campos sería el siguiente en la tabla de atributos del archivo con la información de origen obtenida mediante el script, la capa *shp_datos*:

VerdorIndi	CulturalIn	SocialInd
1	1	3
1	3	1
1	3	1

Ilustración 99. Ejemplo de los índices añadidos en la tabla. Fuente propia.

V. Join (Tool)

Por último, solo nos queda unificar el resto de información, para ver qué elementos tenemos dentro de qué segmentos de la calle y así obtener todos los datos de un mismo tramo en una única tabla de atributos[125]. Para realizar esta unión[126] necesitaremos la capa del buffer que hayamos determinado como óptima, y la capa donde están almacenados los datos extra de los elementos de OSM que queramos unificar.

[124] No se añadirán las imágenes otra vez para no sobrecargar el documento; para ver los pasos acudir a las ilustraciones 93, 94 Y 95.

[125] O, cuando accedamos al archivo desde Python, en un mismo DataFrame.

[126] La función *Join* se realiza al pulsar con el botón derecho sobre la capa que se quiere unificar, en nuestro caso, *Buffer_8_ruido*, la capa resultada del buffer creado en el apartado b de este mismo Anexo.

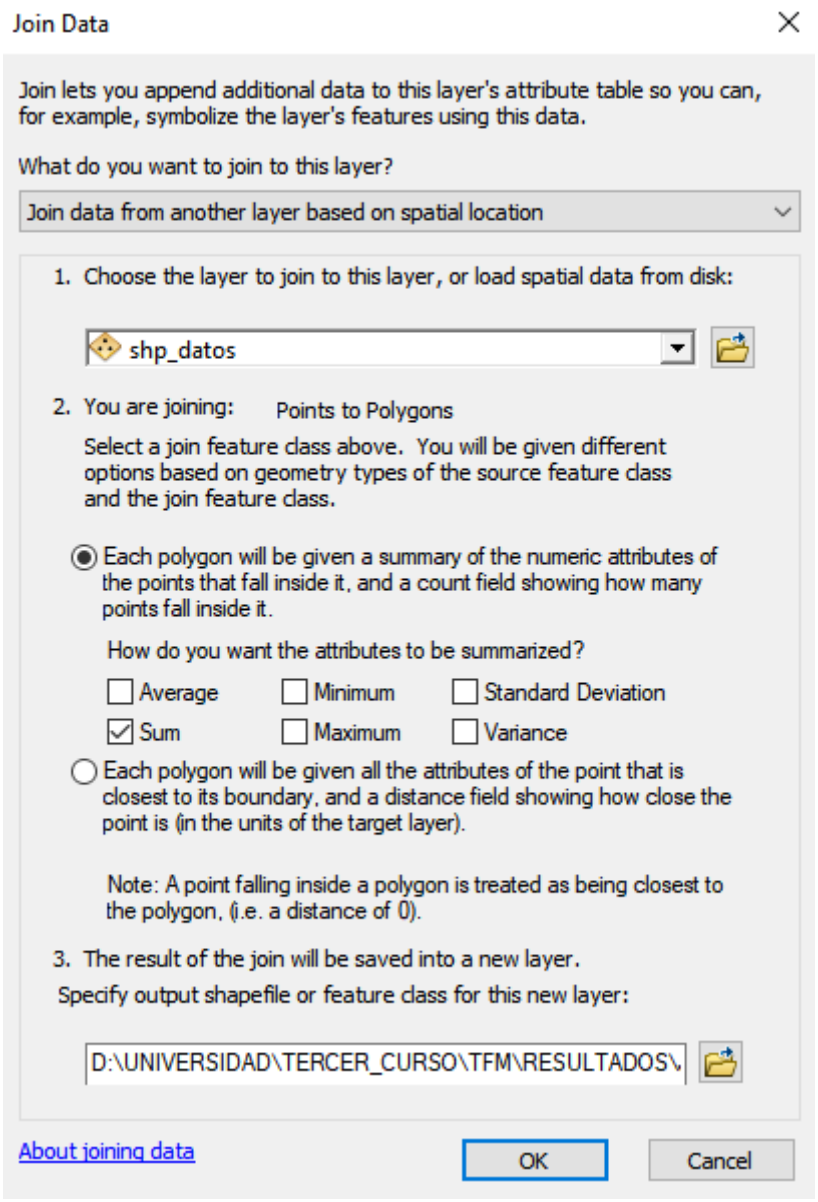


Ilustración 100. Unión de la información de índices con la capa buffer_8m, sumatorio. Fuente propia.

Seleccionaremos la opción *Sum*, para que nos sume todos los elementos que se presenten en la zona del buffer, en base al índice que hayamos determinado sean pertenecientes a un tipo u otro de sector.

Habremos obtenido ahora toda la información de los índices, pero nos falta poner en común nuestros tramos con los nodos que hemos descargado en el script inicial[127], para lo cual será necesario duplicar la capa de los nodos y crear, en cada una, con los pasos que vimos en el apartado inicial de este anexo, un campo nuevo con los valores de los FID de los nodos, que se llamarán FIDfrom y FIDto, respectivamente[128].

[127] Para más información, acudir al Anexo 2.

[128] Es necesario que sean dos capas ya que la unión que vamos a utilizar posteriormente solo nos permite realizar una unión por cada par de capas, por lo que solo podríamos obtener una parte de la información.

Cuando hayamos creado estos campos nuevos, necesitaremos realizar una unión entre los campos denominados *osmid* en estas tablas y los campos *from_* y *to* de nuestra capa final con la información, que es *Buffer_8_ruido_datos.shp*.

Esta vez la unión no será espacial, sino que se realizará en base a los atributos, para así poder seleccionar qué columna y qué valores son los que queremos interrelacionar y con qué campos de la capa base queremos hacerlo, tal y como se muestra a continuación:

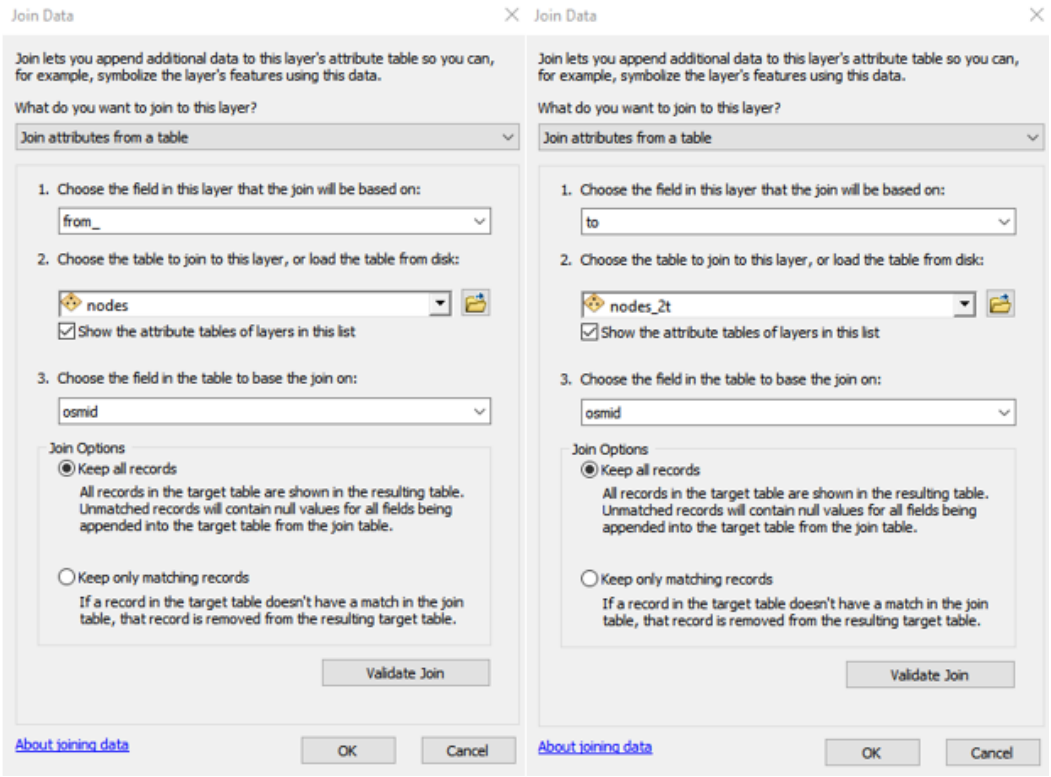


Ilustración 101. Unión de las capas duplicadas de nodos con la tabla de Buffer_8_ruido_datos. Fuente propia

Ahora ya podremos obtener la informacion que necesitamos, es decir, que se nos añaden a nuestra tabla de atributos de la capa que vamos a manejar dos campos nuevos -entre muchos otros que se agregan con más información complementaria-, con el nodo de origen y el de destino por cada tramo, como podemos ver a continuación:

FIDfrom	FID_12_13	osmid_12	y_1	x_1	highway_12	street_c_1	lon_1	lat_1	ref_12	FIDto
117	13	25767445	4372033.10335	725736.76269		4	-0.375883	39.468296		13

Ilustración 102. Ejemplo de la tabla de atributos final. Fuente propia

Ya solo nos quedará exportar la capa[129] con el nombre que queramos para identificarla y analizarla con Python, en nuestro caso se denominará *Datos_completos_buffer_8m_nodos*.

[129] Clic derecho sobre la capa *Buffer_8_ruido_datos* y seleccionamos "Data → Export Data…".

III. Anexo de pruebas alternativas: buffers.

Como hemos indicado en el desarrollo del proyecto, se ha elegido finalmente un buffer de cinco metros para la realización de la demostración de nuestra hipótesis. Sin embargo, el método *Buffer (Analysis)* permite introducir en la creación del buffer cualquier valor, desde 0 hasta infinito. Con respecto a esta decisión, ha sido tomada tras descartar otras pruebas complementarias y testeos alternativos con otros valores, para comprobar que, si hablamos de cinco metros, no hay pérdida sustancial de información, la ruta sigue siendo directa y sin desvíos considerables y, además, no incurrimos en el solape informativo como nos puede pasar en los siguientes casos.

Se ha probado con un buffer de tres metros, que resultaba insuficiente, otro de diez metros, otro de quince y otro de veinte. Aquí se adjuntan únicamente el desarrollo de la prueba del buffer de diez metros y el de veinte metros, para no sobrecargar con información repetitiva y porque han sido las muertas más visuales y explicativas que se han encontrado.

I. Con buffer de 20 metros.

Para comenzar con las pruebas se ha querido establecer un buffer de valor elevado, ya que las calles no cuentan con la misma anchura que las avenidas grandes, buscando así que toda la información sea contenida en los tramos respectivos a los que pertenecen. De esta manera, hemos tomado como valor inicial veinte metros, para lo cual se ha obtenido un resultado como el que se adjunta a continuación:

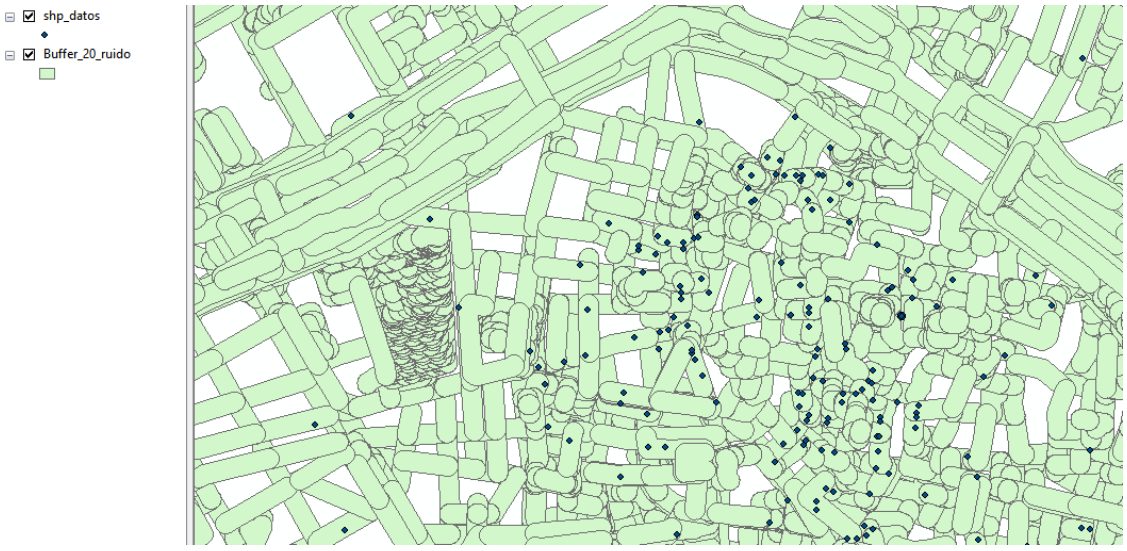


Ilustración 103. Tramos con un buffer de 20 metros, zona Ciutat Vella. Fuente propia.

Donde, como se puede apreciar, los solapes son habituales y en algunos casos casi completos, con tramos que tapan parcial o incluso totalmente otros segmentos de la misma calle o avenida. En contraposición, podemos decir que estamos seguros de que aquí estamos obteniendo toda la información posible, pero también de que podemos estar duplicándola

en algunos de estos segmentos, al tomar elementos que pueden pertenecer a otras zonas o tramos y, con ello, no solo no agilizamos el recorrido, sino que tendríamos que tomar algunos desvíos, retrasándonos así en la llegada al destino, algo que no se encuentra dentro de nuestra intención.

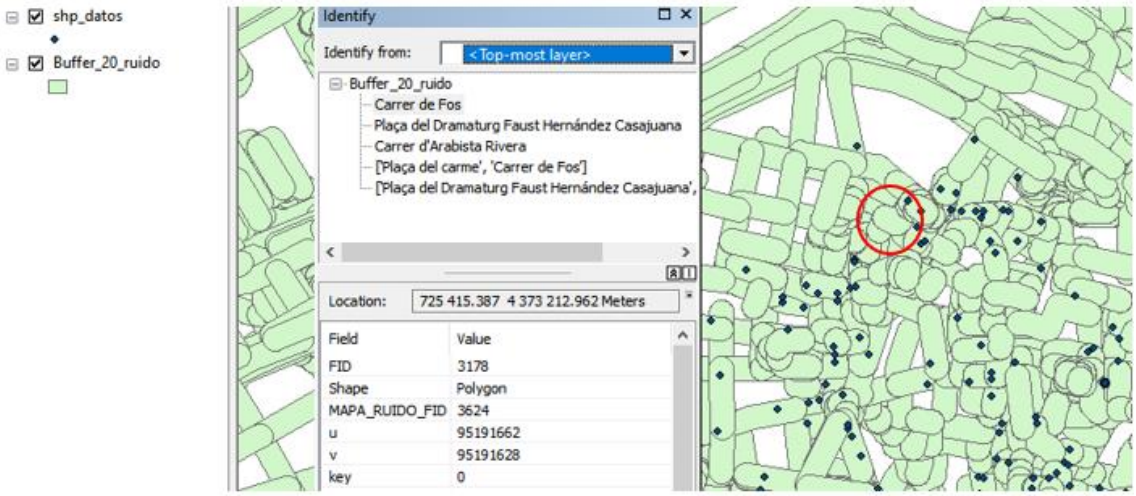


Ilustración 104. Zona de solape múltiple en buffer de 20 metros, de varias calles con diferentes valores. Fuente propia.

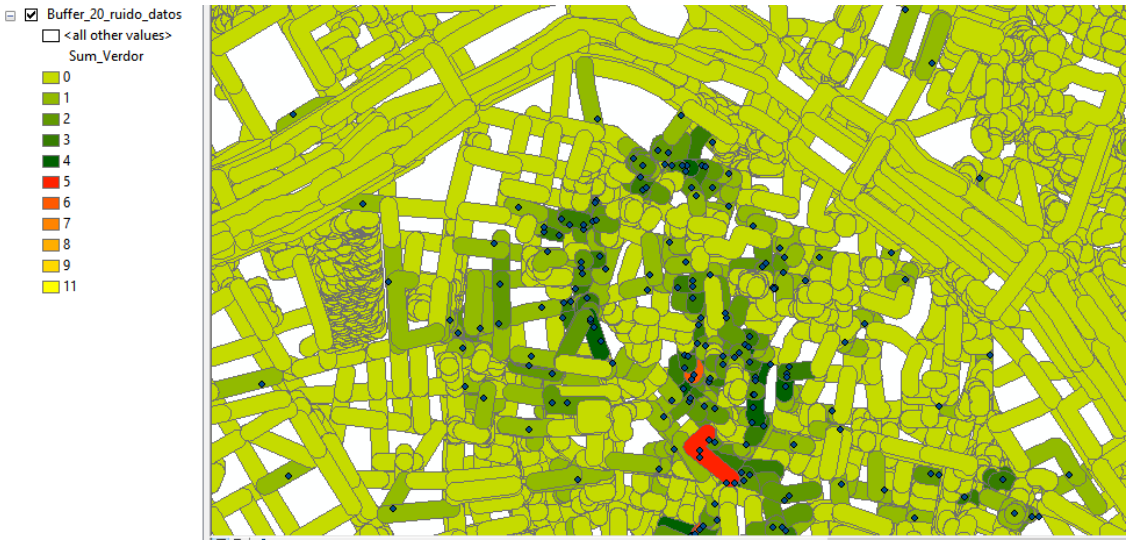


Ilustración 105. Valores de Suma de Verdor con buffer de 20 metros en la zona Ciutat Vella. Fuente propia.

Como vemos, también la leyenda se ha ampliado, puesto que ahora contempla todos los valores en el tramo comprendido entre 0 elementos y 9 elementos, y es normal teniendo en cuenta que contamos con cierta duplicidad en la información. Al pulsar sobre la calle *Dels Cavallers*, que es el tramo horizontal naranja que contiene un elemento, nos toma todos aquellos segmentos que entran en contacto con éste, todos aquellos que se encuentran en el interior del círculo rojo; así, nos da una información que no es real ni verídica.

Por este motivo necesitaremos reducir el radio del buffer, y realizaremos pruebas con los valores de 10 metros, 5 metros y 8 metros.

II. Con buffer de 10 metros.

Tras las pruebas realizadas con respecto a los veinte metros y a los quince se ha establecido, en la información que toma el método de análisis, un buffer de diez metros. Esto dará como resultado un ensanchamiento de este valor de la calle completa, de manera que podrá abarcar más zona que si solamente tomamos un buffer de cinco metros, llegando a algunos elementos cuya localización se pueda encontrar un poco más alejada, pero, al mismo tiempo, asequible dentro de una ruta; es decir, que no nos haga desplazarnos demasiado del camino que nos marca el aplicativo.

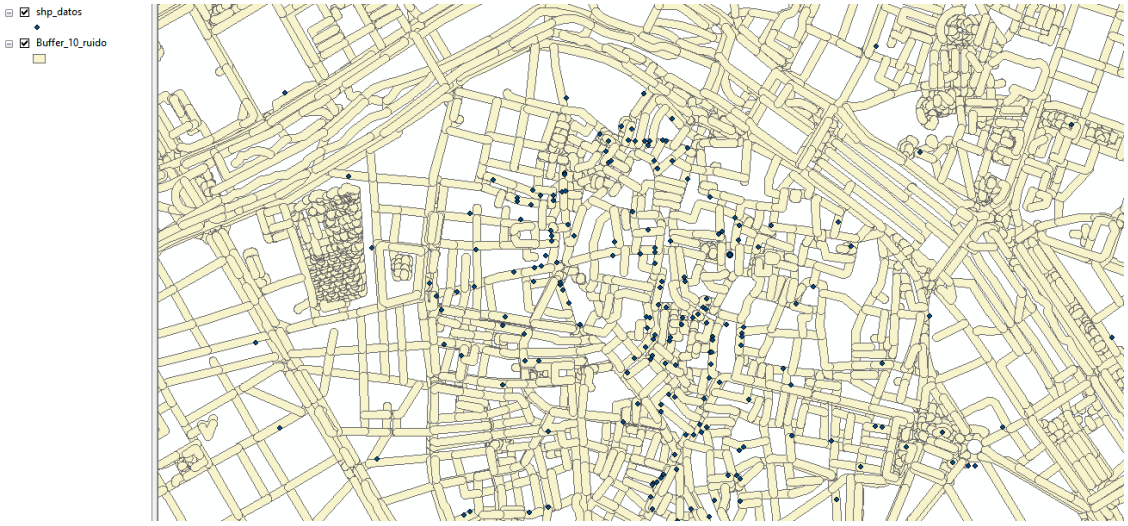


Ilustración 106. Buffer de diez metros, zona Ciutat Vella. Fuente propia.

Como vemos en la ilustración superior, cuya zona es coincidente con la imagen adjunta en el proyecto con respecto al buffer de cinco metros[130], las líneas que simbolizan los segmentos aparecen más gruesas, aunque también se pueden percibir ciertos solapes en las zonas de intersección de tramos, siendo así que, si seleccionamos uno de estos solapes, nos dará la información de su tramo y también de aquel con el que solapa[131]:

[130] Disponible en la ilustración 109.
[131] Todos aquellos que se encuentran dentro del óvalo trazado sobre la imagen.

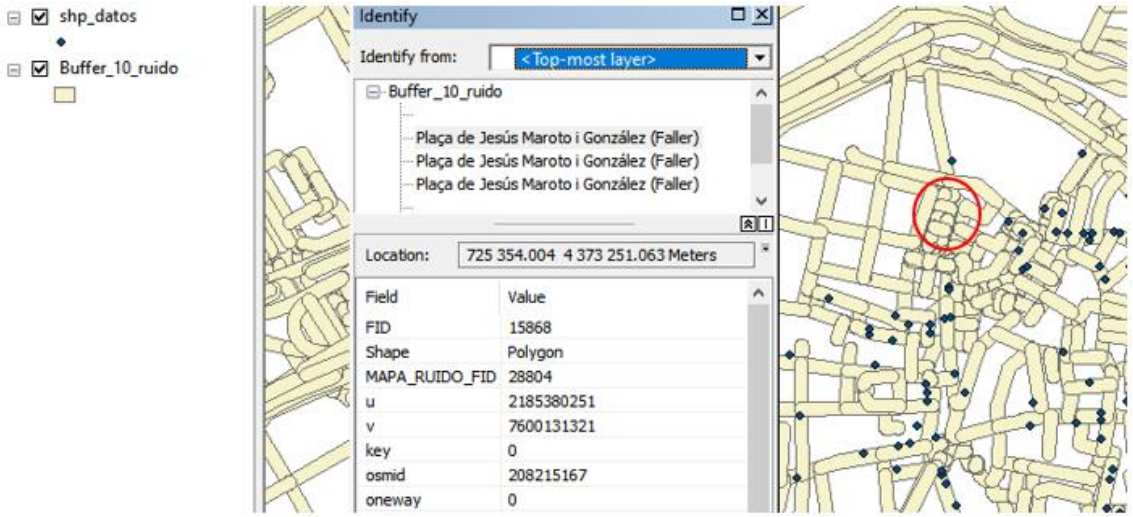


Ilustración 107. Zona de solape con buffer de 10 metros. Fuente propia.

Con respecto a la información que añadiremos posteriormente cuando sean unificadas las capas de información del ruido en cada tramo y aquellos elementos que se encuentren dentro de nuestro buffer de diez metros, tendremos la siguiente imagen de acuerdo a, por ejemplo, la suma de elementos correspondientes a zonas verdes en un segmento de calle:

Ilustración 108. Zona de solape con diferentes valores de verdor. Fuente propia.

Como esto no nos resulta tan útil, se ha decidido proceder con un buffer menor, pasando así a estipular las posibilidades en 5 metros y 8 metros.

III. Buffer de 5 metros.

Para poder ver por qué hemos elegido el buffer de 8 metros como el más adecuado para nuestro análisis, procederemos a ver por qué se descarta el de cinco en primer lugar. Como se puede ver en la imagen a continuación, muchos de los elementos rescatados de OSM que forman parte indispensable del análisis se encontraban fuera del recorrido del buffer:

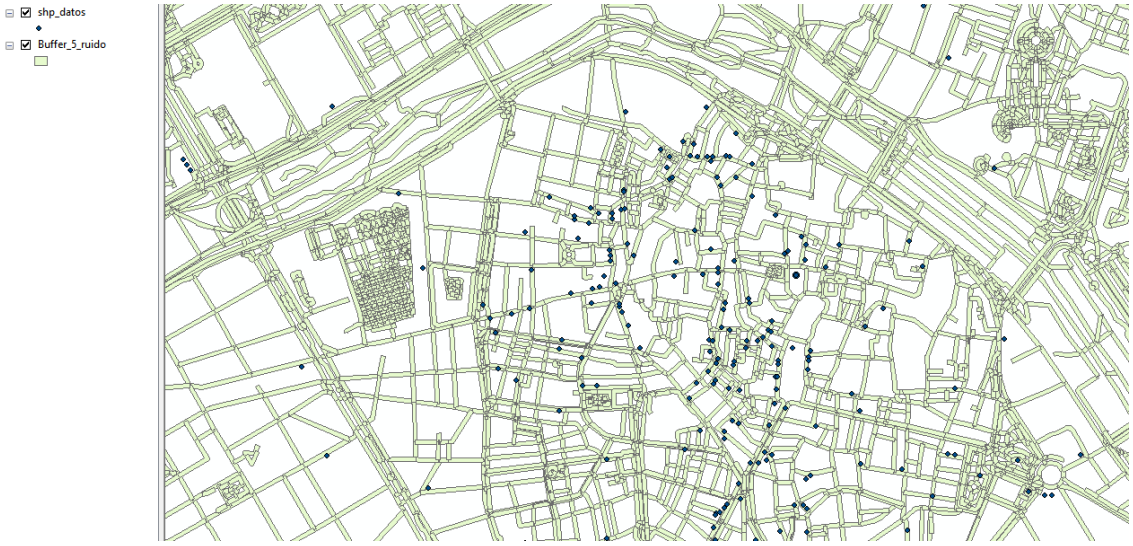


Ilustración 109. Buffer de 5 metros en la zona de Ciutat Vella. Fuente propia.

En cuanto añadimos la información, vemos que muchos de los elementos que queremos incluir en nuestro análisis quedan fuera de nuestras líneas de calle, por lo que no se tomarán, no hay más que ver la leyenda y compararla con los dos apartados anteriores para poder ver las diferentes carencias. No hay solapes, pero sí falta de información:

Ilustración 110. Datos de los elementos en buffer de 5 metros. Fuente propia.

IV. Buffer de 8 metros.

Pese a que estas han sido las imágenes adjuntas a lo largo del proyecto, las añadiremos aquí más ampliadas para que se puedan contrastar correctamente. Se ha elegido este valor para el buffer porque, como hemos visto, un buffer de diez metros era demasiado, ya que en numerosas intersecciones se producían solapes y la información podría llegar a resultar confusa a futuro, y el de cinco, sin embargo, contaba con falta de información, y eso es lo último que necesitamos en este proyecto.

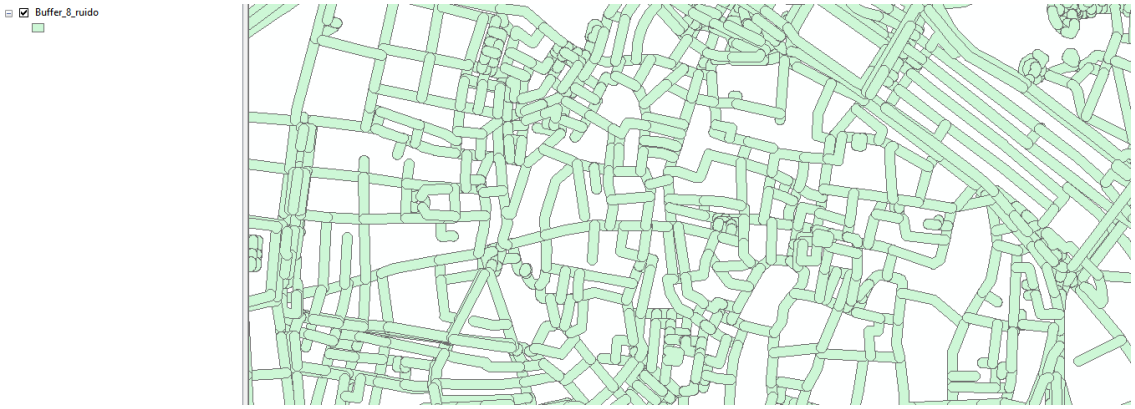


Ilustración 111. Buffer de 8 metros, zona Ciutat Vella. Fuente propia.

Donde, como podemos ver, también se producen solapes, pero se ha preferido contar con este inconveniente a saber que estamos cayendo en la falta o pérdida de información.

Ilustración 112. Visualización de la Suma Cultural, zona Ciutat Vella, buffer de 8 metros. Fuente propia.

IV. Anexo de datos del resto de índices

Como hemos visto en el desarrollo de la metodología aplicativa, la unión entre nuestros tramos de calles con un buffer de ocho metros y los elementos que se pueden encontrar en estos tramos, es decir, la unificación de las capas *Buffer_8_ruido* y *shp_datos*, nos dará como resultado unas columnas sumatorias de todos los índices de manera individual, que contendrá el valor total de los elementos de los diferentes apartados que hemos seleccionado en su totalidad. Así, hemos visto los ejemplos con *Sum_Verdor*, es decir, con el índice que engloba aquellos elementos pertenecientes a lo que hemos denominado *Zonas Verdes*.

Es interesante destacar, sin embargo, que los índices no se mueven en un tramo de valores equitativo: mientras que contamos con numerosos bares, restaurantes y pubs en cada tramo, las zonas verdes o culturales son menores. Conociendo esto, sabremos entonces que no es lo mismo que en una zona haya cuatro lugares de restauración o cuatro lugares culturales[132], ya que, como vemos en la siguiente ilustración, los máximos de estas características son diferentes:

	FID	name	length	Avg_Ruido	Sum_Verdor	Sum_Cultur	Sum_Social
	17597	Carrer del Convent de Santa Clara	50.306	2.5	7	7	21

Ilustración 113. Ejemplo ilustrativo de máximos en Sum_Social y Sum_Cultural. Fuente propia.

Los mapas de color y niveles, acorde a los valores de los distintos índices hallados, quedarán de la siguiente manera[133]:

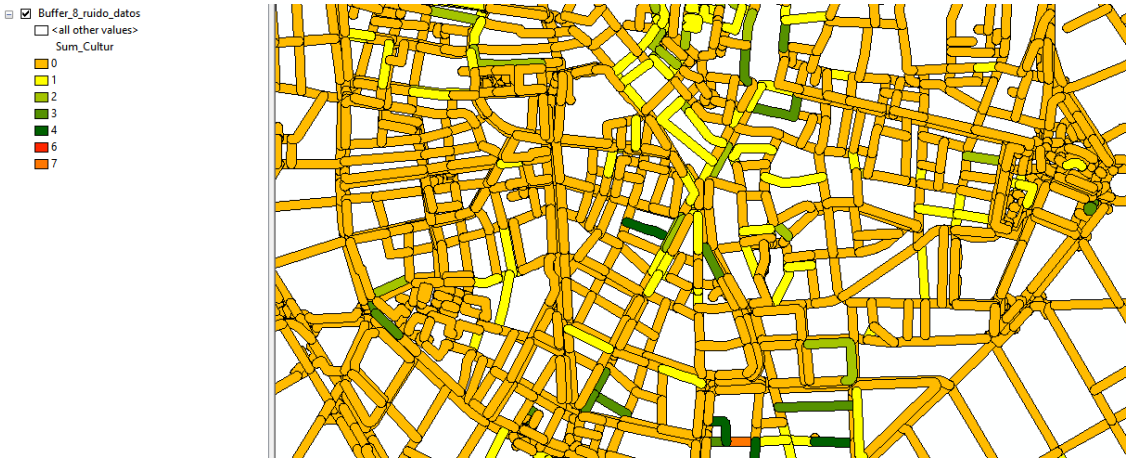


Ilustración 114. Mapa acorde al índice Cultural de cada tramo, zona Ciutat Vella. Fuente propia.

[132] No es un dato determinante en este proyecto por el método de análisis que ha sido seleccionado, donde el usuario da predominancia a uno u otro índice según su preferencia. Sin embargo, en otros análisis, podría ser interesante normalizar estos datos para manejarlos indistintamente.
[133] En la zona sur de Ciutat Vella.

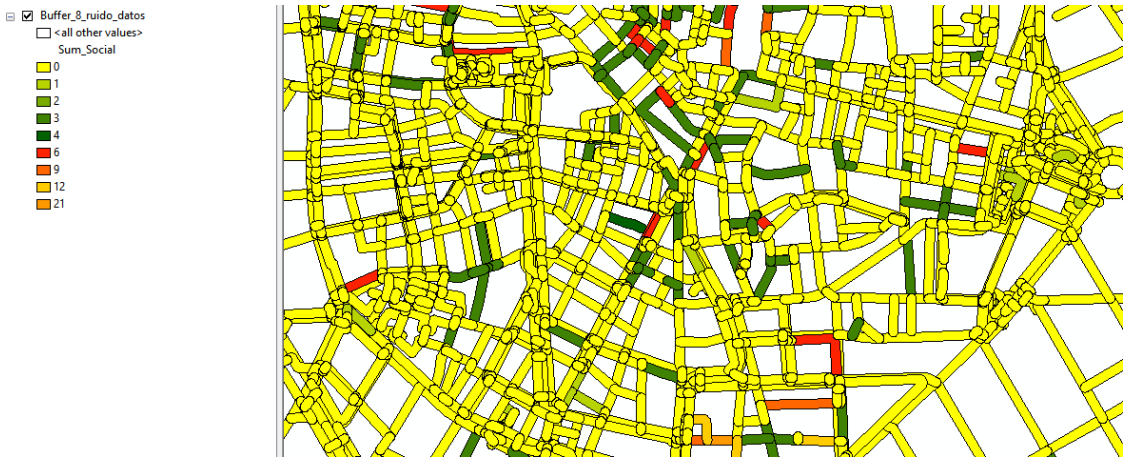


Ilustración 115. Mapa acorde al índice Social de cada tramo, zona Ciutat Vella. Fuente propia.

Donde también podremos ver los máximos, al igual que en la ilustración adjunta 115. Como vemos, la diferencia comparativa es elevada para los mismos tramos, por ejemplo, el siguiente:

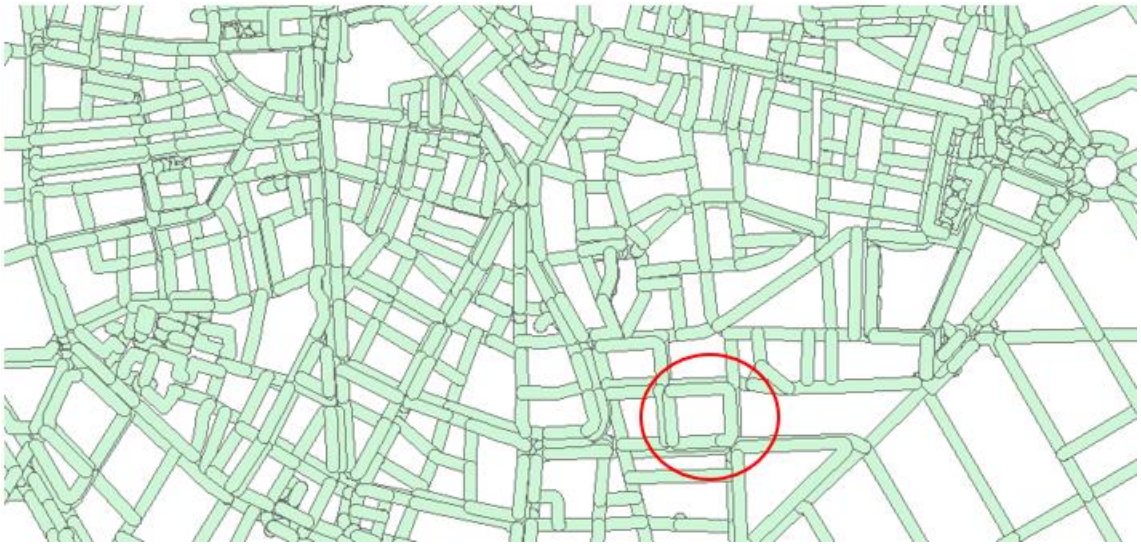


Ilustración 116. Ejemplo de tramo para comparar índices. Fuente propia.

Es un tramo que contiene, como hemos visto, información de los tres índices, pero los cuales no tendrán el mismo baremo ya que, si bien coinciden en el mínimo, no lo hacen en su máximo, siendo preponderantes aquellos elementos de interés social por encima del resto:

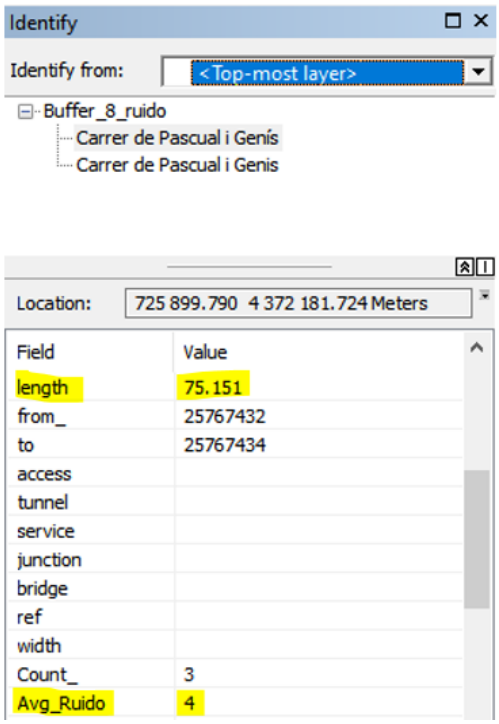


Ilustración 117. Información del tramo elegido con respecto a longitud y ruido. Fuente propia.

Con una media de ruido de 4, que resulta bastante elevada, y una longitud de calle de poco más de 75 metros. En cuanto a las instalaciones que podemos encontrar en este tramo, tenemos el ejemplo que podemos ver a continuación:

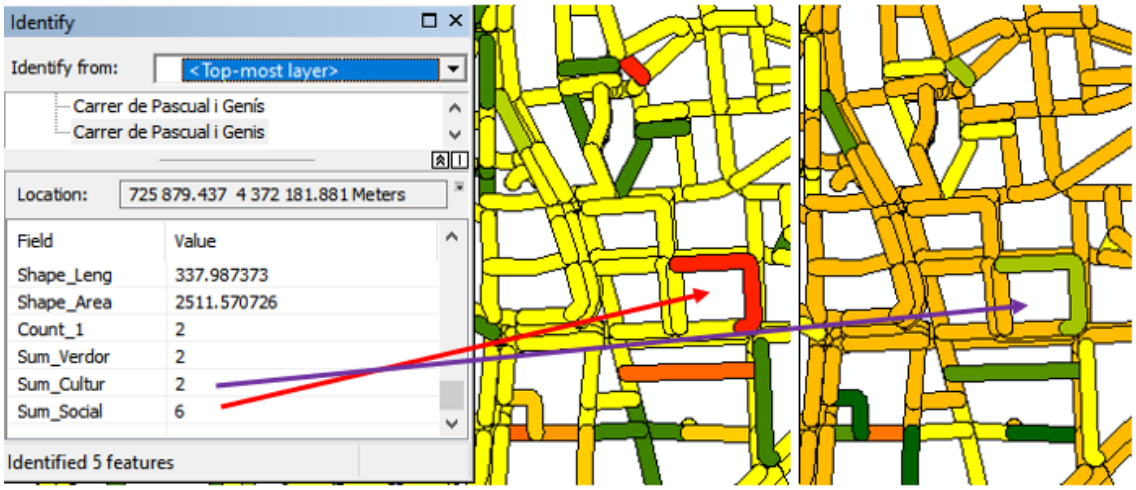


Ilustración 118. Información detallada de los índices en el mismo tramo. Ilustración comparativa. Fuente propia.

Donde, en cuanto a lugares culturales, dos puede parecer bajo, pero el máximo hallado de lugares culturales en los tramos es de 7; mientras tanto, el número de lugares sociales máximo es de 21, por lo que seis es, con esta perspectiva, mucho más bajo con respecto a 21 que 2 con respecto a 7.

V. Anexo de scripts.

En este anexo se adjuntarán los scripts completos y se detallarán los códigos que se estimen necesarios para su comprensión.

I. Script para la obtención de datos

```
import osmnx as ox
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import folium
import geopandas
import fiona


import osmnx as ox
i=0
query = open("query_completa.geojson", "w") #Nombre del geojson a crear


place = "Valencia, Spain" #Lugar de busqueda
ruta="D:\\UNIVERSIDAD\\TERCER_CURSO\\TFM\\RESULTADOS" #Ruta donde guardar los datos
datos = ox.graph_from_place(place, network_type='walk') # query para obtener Datos de calles
proyectar=ox.project_graph(datos) #Proyectar las calles para poder guardarlas
ox.save_graph_shapefile(proyectar,ruta)
fig, ax = ox.plot_graph(datos)


buscar={"tourism":"museum", "tourism":"attraction", "tourism":"artwork", "amenity":"place_of_worship",
        "leisure":"park", "leisure":"garden", "amenity":"bar", "amenity":"pub",
        "amenity":"cafe", "amenity":"restaurant"}
df_todos=ox.geometries_from_place(place,buscar)
resultado=df_todos.to_json()
n = query.write(resultado)
query.close()
#-*- coding: utf-8 -*-
```

Ilustración 119. Script para la obtención de datos de OSM. Fuente propia.

En este primer código vamos a acceder a la información de OpenStreetMap para poder obtenerla y trabajar con ella desde nuestro equipo. Para ello, como se ha detallado en el desarrollo del programa, primero se obtendrán, proyectados, aquellos tramos y nodos necesarios para recrear la red que cubre toda la localización que hayamos elegido. En nuestro caso, será la ciudad de València, por lo que seleccionaremos ese lugar, una ruta donde poder guardar los dos archivos tipo *shapefile* que se generen[134].

Tras esto, será necesario proyectar esos datos para poderlos guardar, en la ruta que hayamos elegido. Además, podremos visualizar la imagen que obtenemos con esta localización, siempre teniendo en cuenta que el tipo de ruta que estamos generando es para los viandantes[135]. Esta imagen se puede ver en la ilustración 43.

Después, tendremos que buscar en la zona aquellos elementos que hayamos creído oportunos que queramos rescatar, que en nuestro caso son todos aquellos que se encuentran dentro de la variable *buscar*. Después, simplemente guardaremos los resultados

[134] Uno con los tramos (*edges.shp*) y otro con los nodos (*nodes.shp*).
[135] Esto es, que no tendrá en cuenta giros prohibidos, cambios de sentido ni la velocidad de las calles.

en un archivo JSON, en la ruta que hayamos establecido, y que obtendrá toda esta información del lugar concreto que hayamos indicado.

Así es como obtendremos los datos de los elementos de nuestra zona, que luego serán llevados a un conversor online[136] para poder trabajar con el mismo tipo de archivo en que obtendremos el resto de los datos y que sea así más sencillo su tratamiento conjunto y puesta en común a fin de obtener un único resultado que contenga toda la información de la que partimos.

II. Script para obtener la información y los índices de usuario

El script para la obtención de los datos ha sido dividido en dos funciones principales. En primer lugar, encontraremos la función *process_data_from_shapefile(file)*, que tomará el archivo, o *file*, que nosotros le queramos facilitar de entrada, y que será el resultado del tratamiento de los datos obtenidos en el script anterior, unificados con los datos de ruido y las capas de tramos y nodos de la red, es decir, el archivo que contenga el compendio unificado de toda la información completa. En nuestro caso ese archivo será *Datos_completos_buffer_8m_nodos.shp*.

En este script vamos a trabajar con diferentes librerías. *Fiona* es aquella que nos va a permitir trabajar sin problema con archivos *shapefile*, abriéndolos de forma sencilla y eficaz. Gracias a *Pandas* podremos crear y trabajar con nuestro DataFrame, que será la base para poder obtener toda la información que necesitaremos; *NetworkX* es la librería que hemos detallado en un inicio, y que será la que tomemos para poder obtener el algoritmo, o al menos su resultado numérico, que después, gracias a *folium*, podremos plasmar cómodamente en un mapa de la zona, obteniendo los resultados más visuales y óptimos para cualquier usuario. Este mapa se abrirá gracias a la librería *webbrowser*, que permite que una pestaña de nuestro navegador abra el mapa guardado de manera automática.

```
import fiona
import pandas as pd
import networkx as nx
import folium
import webbrowser
```

Ilustración 120. Importaciones del programa para sacar los índices. Fuente propia.

Lo primero que necesitaremos será cargar el archivo creado en el SIG de escritorio y analizarlo para poder rescatar de su tabla de atributos todos aquellos datos que nos resulten de mayor utilidad. En este caso y como veremos en la imagen a continuación, rescataremos la información del ruido, que si recordamos es la media del ruido de todo el tramo, los índices y las coordenadas de los nodos de origen y destino, que serán, respectivamente, *lon-lat* y *lon_1-lat_1* y que nos serán útiles después.

[136] Dependiendo del tamaño del archivo usaremos *Mapshaper* u *Ogre*, siendo este último para archivos de menor tamaño.

En base a los índices, con estas variables, vamos a ponderar según la tabla 3, donde establecíamos los criterios. Así, dependiendo del índice que queramos obtener, se ponderará con un peso u otro cada uno de los índices complementarios para, finalmente, obtener un índice de *Agradabilidad* que sea la suma de todos estos índices ponderados según la importancia que en cada tipo de ruta se le dé al índice en particular. Esto se ve claro en las imágenes 103 y 104.

```python
def process_data_from_shapefile(file):
    shape = fiona.open(file)

    output = []

    num_lineas = 100
    i = 0

    for line in shape:

        indice_verdor = 0
        indice_social = 0
        indice_cultural = 0
        indice_ruido = 0
        peso = 0

        ruido = float(line['properties']['Avg_Ruido'])
        verdor = float(line['properties']['Sum_Verdor'])
        cultural = float(line['properties']['Sum_Cultur'])
        social = float(line['properties']['Sum_Social'])
        lon_from = float(line['properties']['lon'])
        lat_from = float(line['properties']['lat'])
        lon_to = float(line['properties']['lon_1'])
        lat_to = float(line['properties']['lat_1'])
```

Ilustración 121. Script para procesar los datos del archivo shapefile, extracción de variables. Fuente propia

```python
pond_verde_v = 0.3
pond_cultural_v = 0.1
pond_social_v = 0.1
pond_ruido_v = 0.05

verdor_v = verdor * pond_verde_v
cultural_v = cultural * pond_cultural_v
social_v = social * pond_social_v
ruido_v = ruido * pond_ruido_v


indice_verdor = verdor_v + cultural_v + social_v + ruido_v



pond_verde_s = 0.1
pond_cultural_s = 0.1
pond_social_s = 0.3
pond_ruido_s = 0.1


verdor_s = verdor * pond_verde_s
cultural_s = cultural * pond_cultural_s
social_s = social * pond_social_s
ruido_s = ruido * pond_ruido_s


indice_social = verdor_s + cultural_s + social_s + ruido_s
```

```python
pond_verde = 0.1
pond_cultural = 0.3
pond_social = 0.0
pond_ruido = 0.1

verdor_c = verdor * pond_verde
cultural_c = cultural * pond_cultural
social_c = social * pond_social
ruido_c = ruido * pond_ruido


indice_cultural = verdor_c + cultural_c + social_c + ruido_c

if ruido <=2:
    indice_ruido = 3

elif ruido > 2 and ruido <= 4:
    indice_ruido = 2

elif ruido == 5:
    indice_ruido = 1

else:
    indice_ruido = 0
```

Ilustración 122. Script para procesar los datos del archivo shapefile, ponderación de los índices. Fuente propia.

```
        l = []
        l.append(line['properties']['FID_1'])
        l.append(line['properties']['FIDfrom'])
        l.append(line['properties']['FIDto'])
        l.append(line['properties']['length'])
        l.append(indice_ruido)
        l.append(indice_cultural)
        l.append(indice_social)
        l.append(indice_verdor)

        output.append(l)

    c = ['IdTramo','NodoFrom','NodoTo','Length','Ruido','Cultural','Social','Verdor']

    df = pd.DataFrame (output, columns = c)

    return df
```

Ilustración 123. Script para procesar los datos del archivo shapefile, generación del DataFrame. Fuente propia.

Una vez se han obtenido los cuatro índices, se almacenan a modo de DataFrame con las columnas nombradas de manera que sea accesible y reconocible después, y el resultado obtenido de esta función, que devuelve el DataFrame completo, será el siguiente:

	IdTramo	NodoFrom	NodoTo	Length	Ruido	Cultural	Social	Verdor
0	0	1	0	3.304	0	0.600000	0.600000	0.300000
1	1	2	0	18.027	0	0.600000	0.600000	0.300000
2	2	3	0	22.729	0	0.550000	0.550000	0.275000
3	3	70	1	24.782	0	0.566667	0.566667	0.283333
4	4	12917	1	8.395	0	0.550000	0.550000	0.275000
..	...	...	...	...	...	...	...	...
95	166	64	61	7.315	0	0.450000	0.450000	0.225000
96	167	95	62	24.411	2	0.400000	0.400000	0.200000
97	171	20221	63	6.310	0	0.450000	0.450000	0.225000
98	172	12926	63	7.416	0	0.450000	0.450000	0.225000
99	173	12926	64	6.418	2	0.400000	0.400000	0.200000

[100 rows x 8 columns]

Ilustración 124. DataFrame resultado del procesamiento con la función "process_data_from_shapefile", 100 columnas. Fuente propia.

La segunda función del script será aquella que nos generará la ruta, tomando ese DataFrame obtenido en la función anterior que vemos en la ilustración superior. La función se llamará *generate_route(df)* y analizará el tipo de ruta que estamos solicitando obtener mediante el input creado, con las diferentes opciones. Así, teniendo en cuenta la zona seleccionada, primará un índice u otro, y tomará este índice de todos los tramos que recorramos, para luego facilitarle esa información como peso al grafo (*weigth*) y que nos pueda crear la ruta más corta en base a esas localizaciones con menor peso[137].

[137] El detalle paso a paso ha sido dado con anterioridad en el apartado de *Generación de rutas*, por lo que no se repetirá aquí.

```python
def generate_route(df, ponderaciones, source, target, color):

    peso = 0

    if ponderaciones == 'zona verde':
        print('elegido zona verde')
        maximo = max(df['Verdor'])
        peso = (maximo-df['Verdor'])
        df['Peso'] = peso

    elif ponderaciones == 'zona social':
        print('elegida zona social')
        maximo = max(df['Social'])
        peso = (maximo-df['Social'])
        df['Peso'] = peso

    elif ponderaciones == 'zona cultural':
        print('elegida zona cultural')
        maximo = max(df['Cultural'])
        peso = (maximo-df['Cultural'])
        df['Peso'] = peso

    elif ponderaciones == 'no ruta':
        print('no se ha elegido ruta particular')
        peso = df['Length']

    else:
        print('elegida ruta más tranquila')
        maximo = max(df['Ruido'])
        peso = (maximo-df['Ruido'])
        df['Peso'] = peso

    df['weight'] = df['Peso']*df['Length']
```

Ilustración 125. Script con la parte de generación de rutas en base a lo elegido por el usuario. Fuente propia.

```python
G = nx.from_pandas_edgelist(df, 'NodoFrom', 'NodoTo', 'weight')
route = nx.shortest_path(G, source=source, target=target, weight='weight')
print(route)

coords_lon = []
coords_lat = []
points = []

for i in route:
    l = df.loc[(df['NodoFrom'] == i)]
    if len(l)>0:
        coords_lon.append(l.iloc[0].loc['LonFrom'])
        coords_lat.append(l.iloc[0].loc['LatFrom'])
    else:
        l = df.loc[(df['NodoTo'] == i)]
        if len(l)>0:
            coords_lon.append(l.iloc[0].loc['LonTo'])
            coords_lat.append(l.iloc[0].loc['LatTo'])
        else:
            print('No se encuentra el nodo', i)


for n in range(len(coords_lat)):
    points.append([coords_lat[n], coords_lon[n]])

p = points[0]
m = points[-1]

map = folium.Map(location=p, zoom_start=15)
folium.map.Marker(p,icon=folium.Icon(color='black', icon='home'),popup="origen").add_to(map)
folium.map.Marker(m,icon=folium.Icon(color='blue', icon='flag'),popup="destino").add_to(map)



folium.PolyLine(points, color=color).add_to(map)
map.save("map.html")
webbrowser.open("map.html")
```

Ilustración 126. Script con la generación del algoritmo, obtención de las coordenadas y los puntos y creación del mapa, marcadores y generación de la polilínea. Fuente propia.

Donde finalmente encontraremos las llamadas a la función:

```
df = process_data_from_shapefile("D:\\UNIVERSIDAD\\TERCER_CURSO\\TFM\\RESULTADOS\\Datos_completos_buffer_8m_nodos.shp")
print(df[['IdTramo','NodoFrom','NodoTo']])

generate_route(df, 'zona verde', 14510, 18732, 'green')
generate_route(df, 'zona social', 14510, 18732, 'brown')
generate_route(df, 'zona cultural', 14510, 18732, 'yellow')
generate_route(df, 'menor ruido', 14510, 18732, 'orange')
generate_route(df, 'no ruta', 14510, 18732, 'red')
```

Ilustración 127. Ejemplo de llamada a las funciones. Fuente propia.

La otra opción indicada para la obtención de las ponderaciones es introducir un input, que le pregunte directamente al usuario qué tipo de ruta solicita[138]:

```
ponderaciones = input('introduzca su preferencia (entre menor ruido, zona social, zona cultural o zona verde):')
```

Ilustración 128. Alternativa a la llamada a las funciones: creación del input. Fuente propia.

Y con este programa habríamos obtenido los resultados que podemos ver en el apartado correspondiente, denominado *Resultados*.

[138] Aunque esto se ha visto más válido en el desarrollo del aplicativo que en la metodología.

VI. Anexo de ejemplo alternativo: Madrid.

Si quisiéramos probar otras ciudades, los mismos programas, procesos y pasos seguidos en esta memoria explicativa servirían para poder obtener resultados diferentes. Por ejemplo, para el caso de Madrid, solo tendríamos que modificar el nombre del archivo a obtener y la localización que se almacenará en la ruta *place* y, de esta manera, obtendremos igualmente la información de aristas y nodos del mapa de la ciudad como se muestra a continuación, importando el resultado obtenido[139] al SIG de escritorio con el que queramos trabajar:

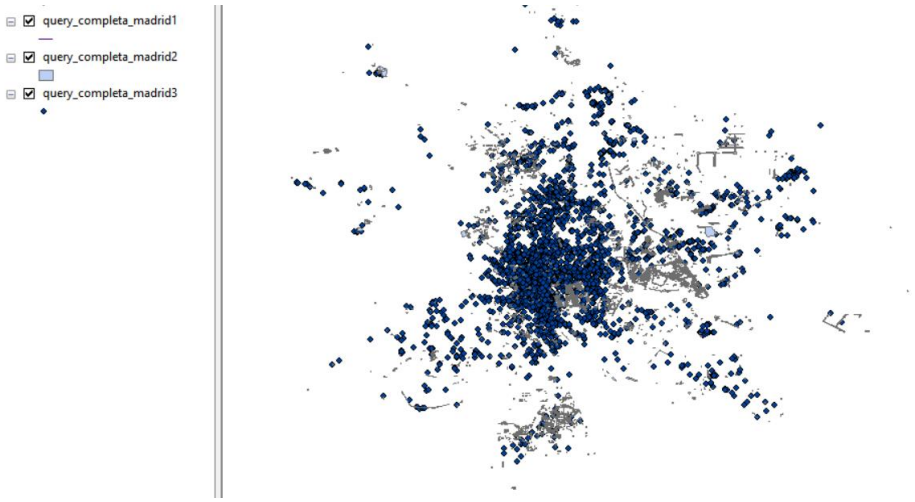


Ilustración 129. Visualización de los tramos y nodos en la red de Madrid. Fuente propia.

Al ser una ciudad más grande y, sobre todo, mucho más masificada y concurrida que València, los datos son mayores[140], llegando a triplicar el peso del archivo GeoJSON obtenido. La consulta que se realiza es prácticamente igual que en el caso de València, detallado en el desarrollo, solo modificando el lugar y el nombre del archivo:

```
import osmnx as ox
i=0
query = open("query_completa_madrid.geojson", "w") #Nombre del geojson a crear


place = "Madrid, Spain" #Lugar de busqueda
ruta="D:\\UNIVERSIDAD\\TERCER_CURSO\\TFM\\RESULTADOS" #Ruta donde guardar los datos
datos = ox.graph_from_place(place, network_type='walk') # query para obtener Datos de calles
proyectar=ox.project_graph(datos) #Proyectar las calles para poder guardarlas
ox.save_graph_shapefile(proyectar,ruta)
fig, ax = ox.plot_graph(datos)


buscar={"tourism":"museum", "tourism":"attraction", "tourism":"artwork", "amenity":"place_of_worship",
        "leisure":"park", "leisure":"garden", "amenity":"bar", "amenity":"pub",
        "amenity":"cafe", "amenity":"restaurant"}
df_todos=ox.geometries_from_place(place,buscar)
resultado=df_todos.to_json()
n = query.write(resultado)
query.close()
#-*- coding: utf-8 -*-
```

Ilustración 130. Cambio en el script para la obtención de datos de Madrid. Fuente propia.

[139] Después de transformarlo a formato *shapefile* previamente.

[140] El archivo JSON obtenido en la consulta ha sido dividido en tres *shapefiles* para poder realizar la visualización, lo que dificulta los procesos posteriores, pues tendrá que ser unificado mediante el SIG o la programación, estableciendo índices o filtros de selección para tomar las zonas que necesitemos.

Por lo demás, el desarrollo sería igual, con la salvedad de que, en el caso de Madrid, no hay datos abiertos relativos a los índices o niveles de ruido, el cual tendría que ser hallado como consecuencia de la velocidad estipulada para cada uno de los tramos, información que se puede encontrar en los datos obtenidos de OSM. La información que obtenemos es igual que la que hemos visto en el caso de València:

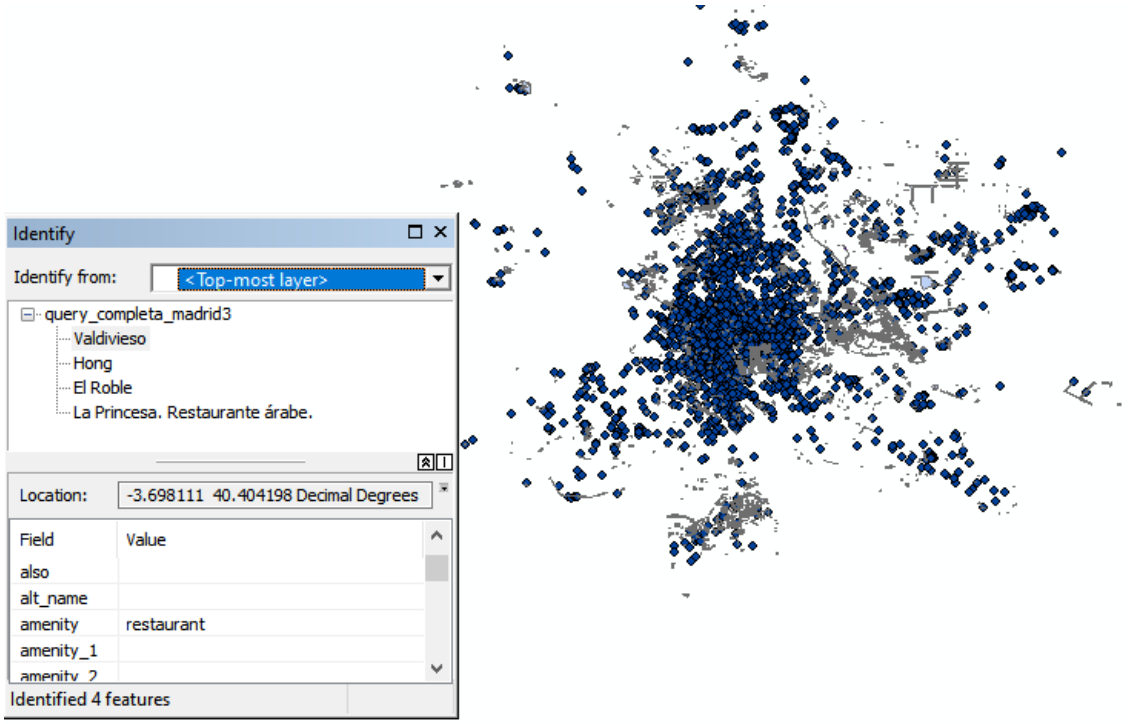


Ilustración 131. Ejemplo de selección de elemento en Madrid. Fuente propia.

Solo que la capacidad requerida para el procesamiento será mayor, así como el tiempo de ejecución que tardemos en obtener los resultados.