UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO FIN DE MÁSTER

Implementación de un sistema de estimaciones del tiempo de recogida para taxis en un entorno de baja latencia y alta disponibilidad

Máster Universitario en Ingeniería Informática

Autor: Otero Rodríguez, Adrián

Tutor: Bellogín Kouki, Alejandro
Departamento de Ingeniería Informática

FECHA: Junio, 2021

Resumen

En los últimos tiempos, el crecimiento de las grandes ciudades y el empeoramiento de ciertos problemas derivados de ello, como la calidad del aire, han acelerado la aparición de una nueva oferta en servicios de movilidad que se ha posicionado como una alternativa real a la tradicional dualidad entre el transporte público y el taxi. Estos servicios han venido acompañados de importantes avances tecnológicos que han supuesto un cambio de paradigma en el sector y un gran avance para sus usuarios. Ante el nivel de innovación de estos nuevos competidores, el tradicional sector del taxi se ha visto en la obligación de evolucionar para adaptarse a estos nuevos tiempos.

En este Trabajo de Fin de Máster se plantea el desarrollo de un nuevo componente dentro de este ecosistema tecnológico, la implementación de un sistema capaz de ofrecer al usuario una estimación del tiempo de espera hasta su recogida, presentando esta información actualizada en tiempo real en la pantalla principal de una aplicación móvil.

Para ello, se ha diseñado una arquitectura donde implementar la solución tecnológica, definida sobre la plataforma *cloud* de Amazon Web Services. Además, se han desarrollado diferentes procesos automáticos de extracción y transformación de datos, incluidas ciertas estructuras especiales como los datos geoespaciales. Además, se ha realizado un minucioso análisis de los datos y se han desarrollado varios modelos predictivos aplicando diferentes algoritmos de aprendizaje automático. Finalmente, se ha detallado la forma en que este sistema pasará a un entorno real de producción, utilizando una base de datos en memoria que garantice unos tiempos de respuesta mínimos.

Por último, este proyecto ha permitido introducir una funcionalidad innovadora en el sector de las aplicaciones de movilidad que permite mejorar la experiencia del usuario. Además, se ha comprobado que la aproximación planteada es totalmente válida y permite ofrecer información precisa al usuario, con un tiempo de respuesta mínimo y sin la necesidad de contactar con los sistemas de los vehículos para conocer su ubicación real.

Palabras clave

Movilidad como servicio, aprendizaje automático, algoritmos, *random forest*, *gradient boosting*, *dataset*, datos geoespaciales, plataforma *cloud*, Amazon Web Services, base de datos en memoria.

Abstract

Lately, the growth of large cities and the worsening of certain problems, such as air quality, have accelerated the development of a new range of mobility services which has emerged as a real alternative to the traditional duality of public transport and taxis. Such services have been brought along great technological advances which have implied a change of paradigm in the sector and a major advance for users. Considering the level of innovation of these new competitors, the traditional taxi sector has been forced to evolve in order to adapt to these new times.

In this Master's Thesis, the development of a new component within this technological ecosystem is proposed. Also, a system capable of offering the user an estimation of the waiting time until pickup has been developed by presenting updated information on the main screen of a mobile application in real-time.

For this purpose, the architecture for developing the technological solution has been defined and designed using the Amazon Web Services cloud platform. Likewise, different automatic data extraction and transformation processes have been developed, including certain special structures such as geospatial data. Furthermore, a meticulous analysis of the data has been carried out and several analytical models have been developed by applying different machine learning algorithms. Finally, it has been detailed the way in which this system will be transferred to a real production environment by using an in-memory database guaranteeing minimum response times.

Lastly, this project has enabled the introduction of an innovative functionality in the field of mobility applications, improving the user experience. It has also been proven that the approach developed is totally valid and allows providing accurate information to the user, with a minimum response time, and without the need to connect with the vehicle systems to determine their real location.

Keywords

Mobility as a service, machine learning, algorithms, random forest, gradient boosting, dataset, geospatial data, cloud platform, Amazon Web Services, in-memory database

Índice de contenidos

Índice de tablas

Índice de figuras

1. Introducción

En este proyecto se ha planteado y desarrollado un sistema analítico capaz de ofrecer a los usuarios estimaciones del tiempo de espera necesario para que un taxi llegue a su ubicación. Esto ofrecerá los usuarios la posibilidad de conocer la información de antemano, antes de realizar una búsqueda de forma activa (es decir, introduciendo origen y destino), mejorando la experiencia del usuario y permitiéndoles valorar el taxi frente a otras posibles alternativas como el transporte público o los vehículos compartidos.

Para ello, como se detalla en los siguientes apartados, se han recolectado datos de diferentes fuentes, tanto internas como externas, se han procesado y analizado estos datos, se han desarrollado varios modelos predictivos para predecir el posible origen del taxi y estimar el tiempo de espera y se ha desarrollado un sistema capaz de ofrecer dichas estimaciones en un tiempo de respuesta mínimo, garantizando una experiencia de usuario óptima.

En este capítulo, en primer lugar, se detalla la motivación del proyecto. A continuación, se enumeran los principales objetivos que se busca alcanzar en el mismo y se describe la estructura de la memoria, de forma que el lector pueda comprender de antemano el contenido que va a encontrar en las diferentes secciones. Por último, se propone un plan de trabajo con las tareas y subtareas necesarias para completar el proyecto con éxito.

1.1. Motivación

En las últimas décadas, se ha experimentado un fenómeno conocido como éxodo rural, lo que ha supuesto un rápido crecimiento poblacional de las ciudades y una acelerada expansión territorial de estas urbes. En el caso de España, la población de los municipios con menos de 2.000 habitantes ha pasado de representar un 39% del total en los años cincuenta, a un 18% en la actualidad [1]. Como consecuencia, se han creado grandes núcleos de población con altos niveles de contaminación y una baja calidad del aire [2]. Esto ha generado grandes preocupaciones que han acaparado numerosos debates políticos. La situación ha ganado especial relevancia en los últimos años, donde se han acelerado los esfuerzos para tratar de revertir esta situación.

En Europa, el transporte es responsable de más del 30% de las emisiones de CO2 y el 72% proviene del transporte por carretera. En este sentido, los coches son la principal fuente de contaminación, generando un 60% del total de estas emisiones y son el segundo

mayor emisor de partículas nocivas de toda Europa [3]. En el caso concreto de Madrid, el tráfico rodado constituye aproximadamente un 70% del NO_2 que respira cada ciudadano [4]. Como consecuencia, numerosas ciudades europeas se han propuesto buscar soluciones a este problema, implantando diferentes medidas y creando nuevas regulaciones enfocadas a expulsar el vehículo privado de sus áreas centrales en favor de otras alternativas como transporte público, vehículos compartidos, empresas de VTC y taxis [5][6].

Por otro lado, la demanda de movilidad ha crecido exponencialmente en los últimos años, lo que ha propiciado la aparición de nuevas alternativas como empresas de VTC (Uber y Cabify) o de vehículos eléctricos compartidos (Zity, Emov, Wibble, etc.). La aparición de dichas empresas ha venido acompañada de innovadoras plataformas tecnológicas que permiten al usuario reservar un vehículo de forma sencilla en cuestión de segundos desde su propia aplicación móvil. Estas plataformas cuentan con una gran ventaja competitiva [7][8] frente a servicios más tradicionales como el taxi. En este contexto, surge la necesidad de dotar de nuevas mejoras e innovaciones al sector del taxi mediante servicios y soluciones novedosas que permitan optimizar y mejorar el servicio para alcanzar el nivel tecnológico de estos nuevos competidores.

La motivación de este proyecto será desarrollar una parte innovadora dentro de este ecosistema tecnológico. Se buscará mejorar la experiencia del usuario dentro de una aplicación móvil mediante un sistema que permita ofrecer al usuario estimaciones del tiempo de espera hasta que un taxi pueda llegar a su ubicación, de forma proactiva, es decir, sin que el usuario necesite realizar una búsqueda para obtener esta información.

1.2. Objetivos

El principal objetivo por alcanzar en el desarrollo de este proyecto es dotar al sector del taxi de nuevas capacidades tecnológicas en un contexto empresarial. Concretamente enfocado en el servicio al usuario, ofreciendo una nueva funcionalidad que le permita conocer el tiempo estimado de recogida antes de solicitar e incluso buscar un taxi. Este objetivo general se puede dividir en otros más específicos, descritos a continuación:

- **Objetivo 1**. Búsqueda y recolección de datos, tanto desde fuentes internas como externas. En estas fuentes se recogerá información relacionada con la movilidad entre distintos puntos de la ciudad, los datos de diferentes desplazamientos, información acerca de la disponibilidad de los taxis y datos geográficos.

- **Objetivo 2**. Tratamiento y análisis de datos, en busca de patrones relacionados con la movilidad en taxi dentro de Madrid.
- **Objetivo 3**. Creación de un modelo analítico, capaz de predecir el grado de disponibilidad de los taxis en las diferentes áreas de Madrid.
- **Objetivo 4**. Desarrollo de un algoritmo, capaz de estimar el tiempo entre un área de origen (ubicación del taxi) y un área de destino (punto de recogida del usuario).
- **Objetivo 5**. Análisis de los resultados en un entorno de pruebas, aplicando diferentes métricas de evaluación y optimización de los algoritmos.
- **Objetivo 6**. Puesta en producción del sistema completo en una aplicación real. Se deben tener en cuenta las restricciones impuestas respecto a tiempos de respuesta muy reducidos que garanticen la calidad óptima en la experiencia del usuario.
- **Objetivo 7**. Análisis de los resultados en un entorno real, medido en base a las diferencias entre el tiempo estimado y el observado.

1.3. Organización de la memoria

En el capítulo 2, "Estado del arte y de las tecnologías", se estudiará la situación actual del sector de aplicaciones móviles orientadas a servicios de movilidad, incluyendo sus evoluciones tecnológicas. Se revisarán las últimas novedades en el ámbito de los sistemas geográficos y los datos geoespaciales. Se analizarán los principales algoritmos de aprendizaje automático, profundizando en algunos de ellos. Se investigarán las diferentes plataformas *cloud* y sus características. Por último, se revisarán algunas de las librerías de Python que podrían ser necesarias en este proyecto.

En el capítulo 3, "Diseño y desarrollo", se detallará la arquitectura propuesta para la ingesta de datos, el procesamiento de la información y el desarrollo de los modelos predictivos, profundizando en cada una de las fases del desarrollo.

En el capítulo 4, "Integración y resultados", se describirá la forma en que el sistema se integrará con una aplicación móvil para dar respuesta a las peticiones. Por último, se analizarán las pruebas realizadas y sus resultados.

En el capítulo 5, "Conclusiones y trabajo futuro", se revisarán las conclusiones extraídas tras el desarrollo del proyecto y se valorará su impacto en el servicio ofrecido. Finalmente, se propondrán futuras líneas de trabajo, incluyendo posibles mejoras y alternativas a la solución propuesta.

1.4. Plan de trabajo

En esta sección se detallará el plan de trabajo en que se ha organizado el proyecto, incluyendo las diferentes tareas y subtareas que lo componen, una estimación del tiempo requerido para completar cada una de ellas y la distribución de estas a lo largo del tiempo.

Este plan de trabajo se ha estructurado en seis grandes bloques de tareas: en primer lugar, se llevará a cabo la recogida de información y se investigará acerca de los estudios realizados previamente en el ámbito del aprendizaje automático y sobre las diferentes tecnologías, técnicas y librerías existentes (T1. Estado del arte e investigación); a continuación, se llevará a cabo la toma de requisitos y el diseño e implantación de un sistema que cubra las necesidades planteadas (T2. Diseño de la solución y despliegue de la infraestructura); en tercer lugar, se llevará a cabo el desarrollo de la solución analítica, parte central de este proyecto. Esta parte abarca desde la recolección de datos hasta la creación de un conjunto de modelos capaces de estimar los tiempos de recogida (T3. Desarrollo de la solución analítica); en cuarto lugar, se validarán los resultados de los modelos (T4. Realización de pruebas y evaluación de los resultados); a continuación, se implantará la solución en un entorno real (T5. Puesta en producción en un entorno real); por último, se redactará un informe detallado de todo el proceso y las conclusiones obtenidas (T6. Elaboración del informe final).

En la siguiente imagen (Ilustración 1) se muestra la evolución temporal de este plan de trabajo (en días) y de las tareas y subtareas en que este se divide cada una de las fases del mismo.

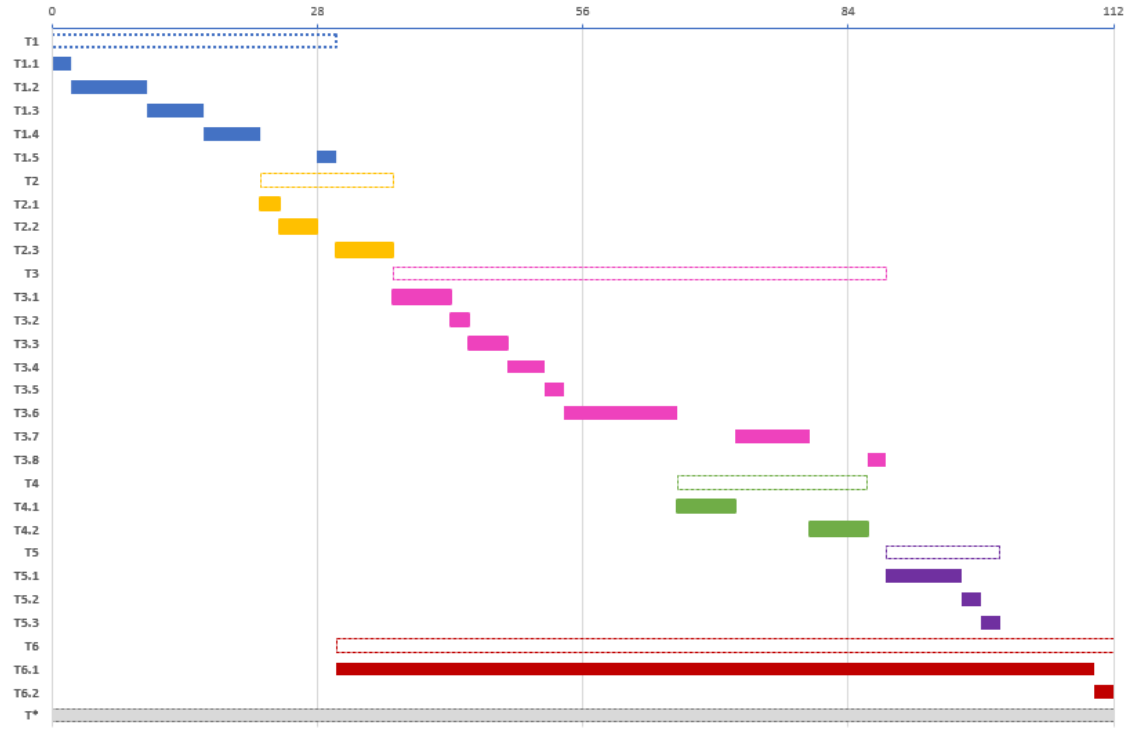

Ilustración 1. Planificación temporal dividida en tareas y subtareas (días)

En la siguiente tabla, se incluye el detalle de cada una de las subtareas que componen los seis grandes bloques de tareas mencionados anteriormente y se incluye una estimación del tiempo requerido, en horas, para completar cada una de ellas.

Tabla 1. Desglose de subtareas y estimación (horas)

#	Tarea / Subtarea	Estimación (horas)
T1	**Estado del arte e investigación**	**60**
T1.1	Revisión de librerías y frameworks para trabajar con datos geoespaciales	5
T1.2	Investigación sobre los diferentes tipos algoritmos de aprendizaje automático	20
T1.3	Estudio sobre las diferentes métricas de evaluación empleadas en los algoritmos de aprendizaje automático	15
T1.4	Análisis sobre los diferentes tipos de plataformas *cloud,* sus características y su funcionamiento	10
T1.5	Análisis de las diferentes alternativas existentes en bases de datos con baja latencia y su uso en un entorno productivo con estrictos requerimientos en los tiempos de respuesta	10
T2	**Diseño de la solución y despliegue de la infraestructura**	**30**
T2.1	Análisis de necesidades y requisitos	5
T2.2	Creación de una plataforma de análisis de datos donde recolectar, procesar y analizar la información	10

T2.3	Diseño de una arquitectura capaz de procesar las peticiones y darles respuesta cumpliendo con los requisitos fijados (milisegundos)	15
T3	**Desarrollo de la solución analítica**	**100**
T3.1	Recolección de datos desde fuentes internas (servicio de taxi) y externas (datos abiertos que incluyen información geoespacial, movilidad, etc.)	15
T3.2	Procesamiento, tratamiento, limpieza de datos y carga de la información en el *data lake*, en formato parquet	5
T3.3	División del área metropolitana de Madrid en polígonos que permitan agrupar los datos geoespaciales (latitud y longitud) en conjuntos más amplios para su análisis	10
T3.4	Análisis exploratorio de los datos disponibles (estadística descriptiva, volúmenes, gráficos univariantes, análisis de correlaciones, etc.)	10
T3.5	Preparación de los conjuntos de entrada para el entrenamiento de los modelos	5
T3.6	Elaboración de un modelo predictivo que permita identificar las áreas con más taxis disponibles en un momento determinado	30
T3.7	Desarrollo de un modelo predictivo que permita estimar el tiempo necesario para viajar entre dos áreas en un momento concreto	20
T3.8	Combinación de ambos desarrollos en una solución final que ofrezca estimaciones del tiempo de espera con unos parámetros de entrada establecidos (tiempo y ubicación)	5
T4	**Realización de pruebas y evaluación de los resultados**	**30**
T4.1	Pruebas sobre los modelos predictivos desarrollados	15
T4.2	Análisis de los resultados obtenidos y optimización de los modelos	15
T5	**Puesta en producción en un entorno real**	**30**
T5.1	Implementación de la arquitectura necesaria para dar respuesta a las peticiones, cumpliendo las restricciones temporales	20
T5.2	Puesta en producción de la solución analítica desarrollada	5
T5.3	Análisis de los resultados en un entorno real y valoración del impacto en el producto	5
T6	**Elaboración del informe final**	**30**
T6.1	Redacción completa de la memoria final del TFM	25
T6.2	Preparación y defensa oral del TFM	5
T*	**Reuniones y seguimiento**	**20**
TOTAL		**300**

2. Estado del arte y de las tecnologías

En este capítulo se revisarán las soluciones existentes en el mercado dentro del ámbito de la movilidad, analizando las funcionalidades ofrecidas por las diferentes aplicaciones móviles relacionadas con la temática de este proyecto. Por otro lado, se investigarán los principales sistemas de información geográfica y el uso de datos geoespaciales. Además, se analizarán algunos de los principales algoritmos de aprendizaje automático, especialmente aquellos que son utilizados frecuentemente en la resolución de problemas de predicción, enfocados a calcular la disponibilidad de taxis de un área y los utilizados en la estimación de tiempos. A continuación, se analizarán las diferentes plataformas *cloud* existentes y sus principales características. Por último, se revisarán las diversas librerías de Python que pueden aplicarse en la resolución de este problema.

2.1. Aplicaciones de movilidad

En la actualidad existe una gran variedad de aplicaciones móviles que facilitan el uso de servicios VTC y de vehículos compartidos. La aparición de estas empresas innovadoras ha obligado al sector del taxi a aumentar los esfuerzos por desarrollar su nivel tecnológico y alcanzar a sus nuevos competidores. Estas nuevas aplicaciones han sido desarrolladas tanto por empresas globales como nacionales y locales. En este proyecto nos centraremos en el área metropolitana de Madrid, donde algunas de las aplicaciones más frecuentes son:

- **Aplicaciones móviles asociadas a empresas de VTC:** Uber [9], Cabify [10] y Auro [11].
- **Aplicaciones móviles relacionadas con el sector del taxi:** Freenow [12], Joinup [13] y Pidetaxi [14].

Entre la funcionalidad ofrecida por estas aplicaciones, cabe destacar que la mayoría de ellas muestran al usuario los vehículos VTC o taxi cercanos en un mapa en tiempo real. Sin embargo, ninguna de ellas ofrece una estimación del tiempo de recogida, únicamente se muestra esta información al usuario cuando este realiza una búsqueda o reserva un viaje. A continuación, se detallan algunos ejemplos de la funcionalidad ofrecida por estas aplicaciones y su funcionamiento a la hora de ofrecer al usuario el tiempo estimado de recogida.

En el caso de Uber, en la pantalla principal de la aplicación se muestran los vehículos disponibles cerca del usuario, en dicha pantalla no se ofrece una estimación del tiempo de recogida. En el momento en que el usuario inicia la búsqueda e introduce un destino se calcula una estimación del precio del trayecto, la hora de llegada y el tiempo de recogida (recuadro rojo). La aplicación necesita conocer la posición del vehículo más cercano y calcular el tiempo de viaje hasta la ubicación del usuario. En la siguiente ilustración (Ilustración 2) se muestran ambas pantallas en la aplicación de Uber.

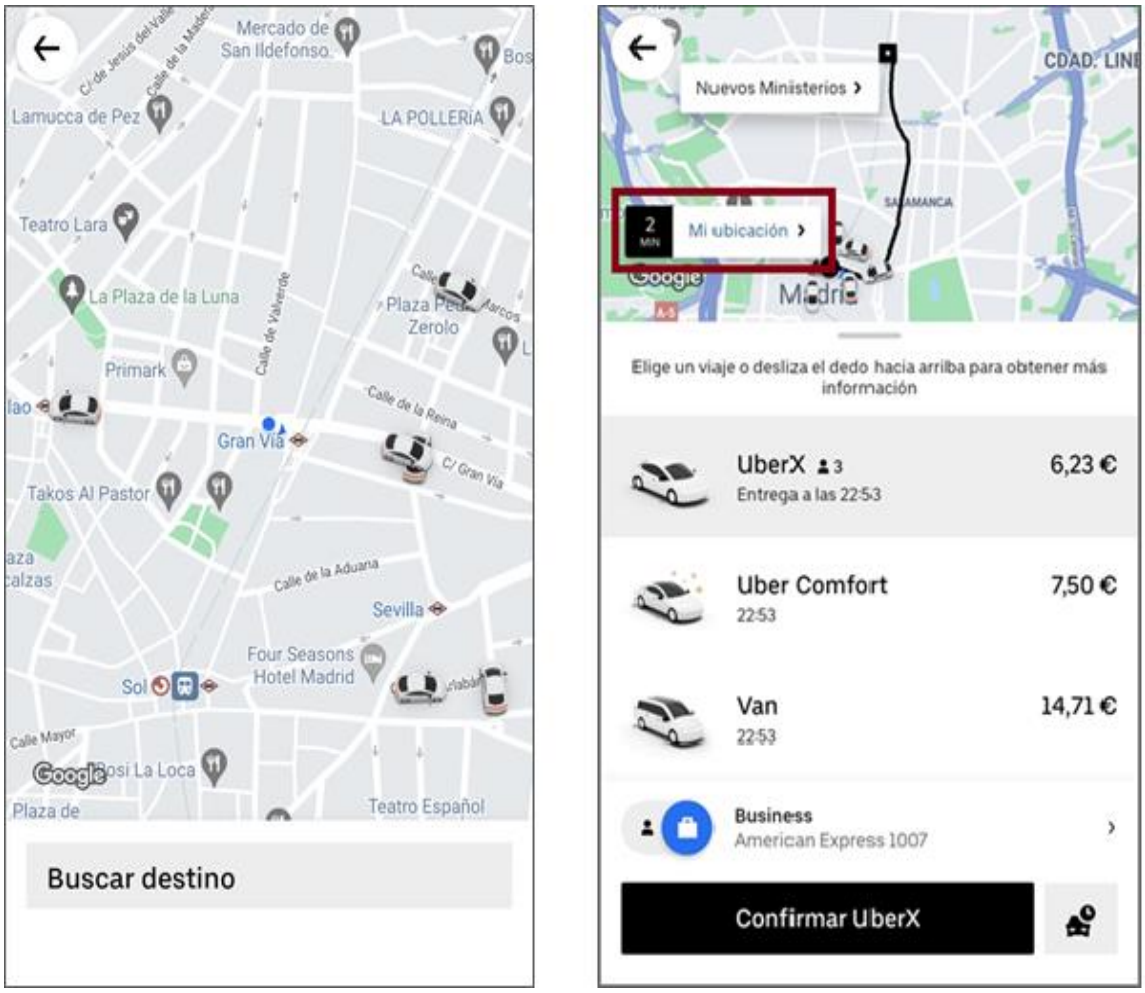

Ilustración 2. Aplicación móvil de Uber [9].

Por otro lado, en el caso de Cabify, en su pantalla principal muestran los vehículos cercanos disponibles en un mapa centrado sobre la ubicación actual del usuario. Unas características muy similares a las observadas en la aplicación de Uber. En este caso, tampoco se ofrece ninguna información al usuario acerca del tiempo estimado de recogida en esta pantalla.

Cuando el usuario introduce un destino, se calcula la ruta y el coste del trayecto, además, se muestra al usuario una estimación del tiempo de recogida. Es decir, el sistema necesita una búsqueda activa por parte del usuario para identificar los vehículos cercanos que pueden aceptar el servicio y calcular en función de ello el tiempo de recogida hasta la ubicación del usuario. En la siguiente ilustración (Ilustración 3) se muestra la pantalla principal y la pantalla de búsqueda en la aplicación de Cabify.

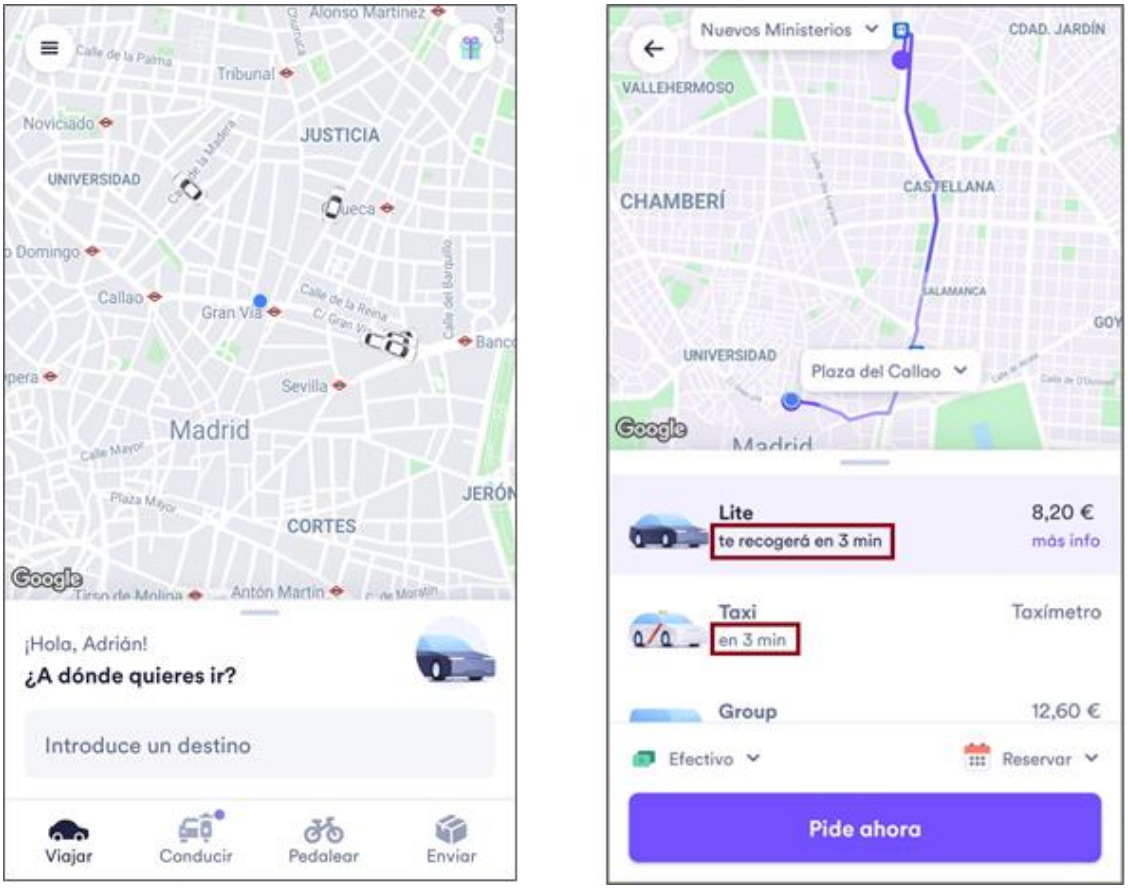

Ilustración 3. Aplicación móvil de Cabify [10].

Por último, Auro no ofrece ningún tipo de información acerca de los vehículos cercanos en ninguna de las pantallas. No se informa al usuario acerca del tiempo estimado de recogida ni en la pantalla principal ni cuando el usuario realiza una búsqueda de forma activa. En la siguiente ilustración (Ilustración 4) se puede observar una muestra de ambas pantallas.

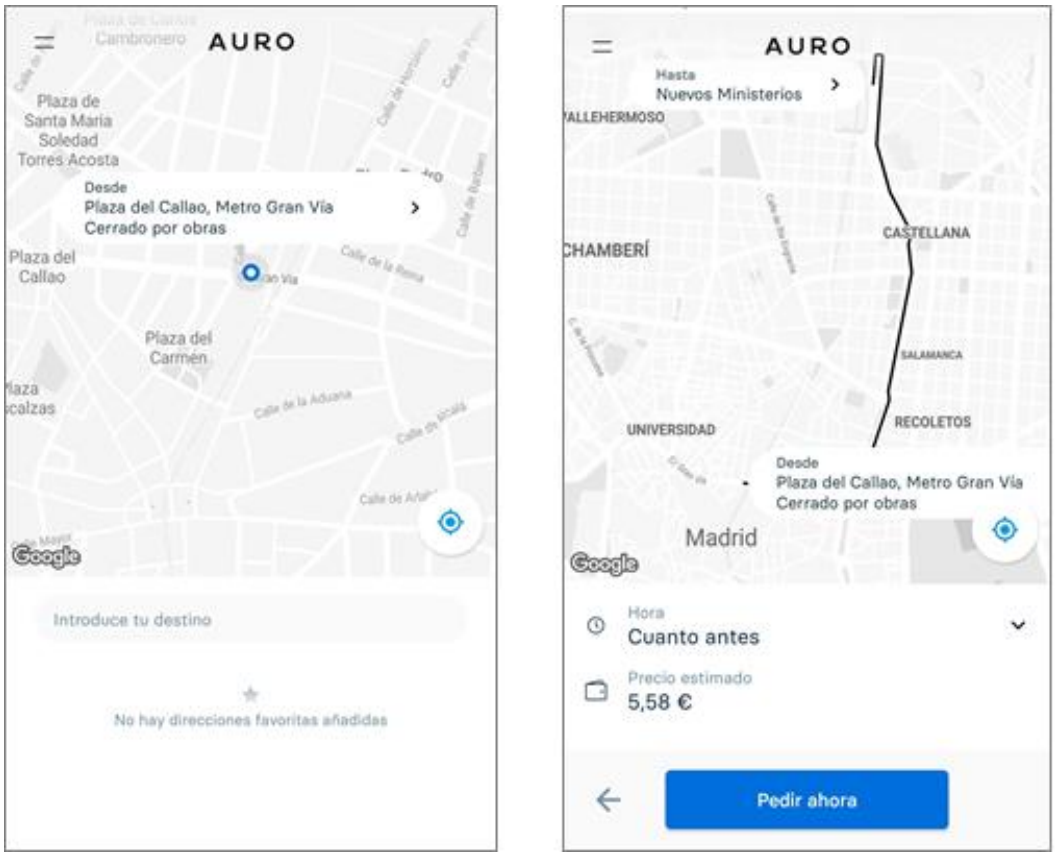

Ilustración 4. Aplicación móvil de Auro [11].

Teniendo en cuenta el análisis realizado sobre las características ofrecidas por las compañías más importantes en el sector de movilidad a la hora de estimar el tiempo de recogida, el planteamiento de este proyecto parece una propuesta innovadora en el sector: diseñar un sistema capaz de ofrecer estimaciones válidas sin la necesidad de que el usuario realice una búsqueda activa ni de contactar con los vehículos cercanos para calcular esta estimación. Por un lado, se aporta al usuario información extra, que puede ayudarle en su proceso de decisión y, por otro lado, la empresa puede ofrecer una mejor calidad del servicio aumentando su reputación y el grado de satisfacción de los usuarios.

2.2. Datos geoespaciales

Los datos geográficos o geoespaciales hacen referencia a la información asociada, implícita o explícitamente, con localizaciones relativas a la superficie de la Tierra. Estos conjuntos de datos poseen un componente geométrico o espacial, que permite ubicar la localización de los elementos y establecer relaciones espaciales entre diferentes objetos. Dentro de este ámbito, uno de los sistemas más extendidos es GIS (*geographic information system*, o sistema de información geográfico) [15]. Este sistema proporciona

la tecnología necesaria para organizar mapas y datos en capas que pueden ser combinadas, analizadas y visualizadas.

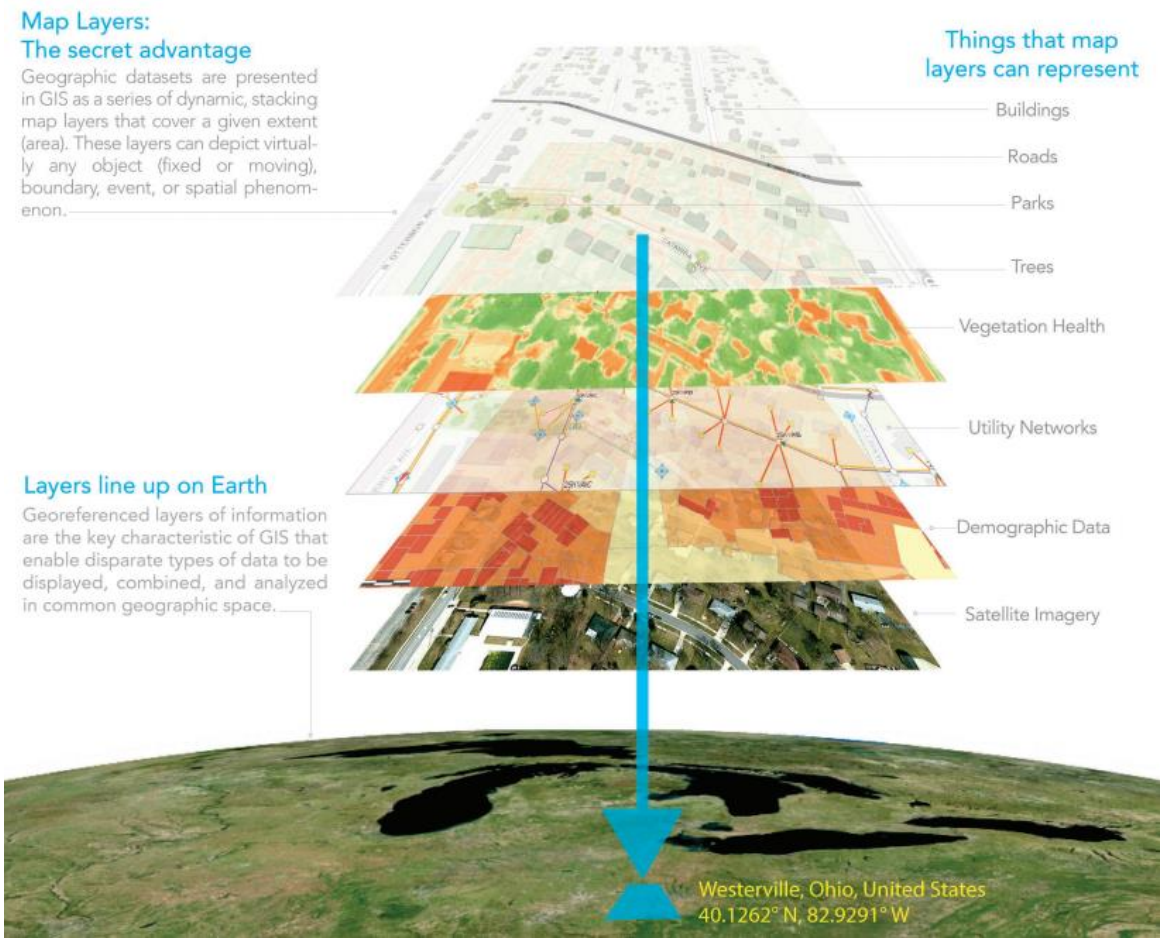

Ilustración 5. Capas en el sistema GIS [15].

El uso de GIS está implantado en todo de tipo de sectores, incluidas organizaciones gubernamentales, que ofrecen conjuntos de datos públicos acerca de sus territorios. Por ejemplo, la comunidad [16] y el ayuntamiento [17] de Madrid cuentan con sus propios geoportales donde proporcionan todo tipo de datos geográficos en formatos como KML o SHP.

Dentro de este *framework*, uno de los formatos más comunes es Shapefile (SHP) [18]. Este formato está ampliamente extendido y ha llegado a convertirse en un estándar en la industria para el intercambio de información geográfica entre sistemas GIS. Es un formato vectorial de almacenamiento que permite guardar información acerca de entidades geográficas, incluyendo su ubicación geométrica y diferentes atributos. Estos elementos

geográficos se pueden representar mediante puntos, líneas o polígonos. El formato se compone de tres tipos de archivos obligatorios y algunos adicionales:

1. **.shp**: Este fichero incluye las entidades geométricas de los objetos.
2. **.shx**: Este fichero almacena el índice de las entidades geométricas.
3. **.dbf**: Es la base de datos donde se almacenan los atributos de los objetos.

Aunque este tipo de formato continúa siendo el más extendido dentro del sector, en los últimos años han surgido algunas alternativas que buscan solucionar ciertas limitaciones de este:

- Está limitado a un máximo entre 2 – 4 GB por archivo.
- Los tipos de datos están limitados a *integer*, *float*, *date* y *text*, con un máximo de 254 caracteres. Es decir, no hay soporte para tipos más avanzados.
- El fichero .dbf tiene un límite de 255 campos de atributos. Además, el nombre de estos atributos está limitado a 10 caracteres.
- Por defecto, no incluye la definición del sistema de referencia de coordenadas.
- Se requieren varios tipos de archivos, que GIS interpreta como uno único.
- No es posible definir el conjunto de caracteres utilizado (ISO, UTF-8, etc.).

Entre los formatos que han surgido para hacer frente a las desventajas mencionadas anteriormente y que buscan consolidarse como la principal alternativa al formato Shapefile, se encuentran formatos como KML, GML, GeoJSON o GPKG, algunos de los cuales se utilizan en populares plataformas como Google Maps. Sus principales características y diferencias son:

- **KML (*Keyhole Markup Language*)**: Este tipo de formato combina cartografía y geometría en un único archivo y utiliza el sistema de referencia de coordenadas WGS-84. Su principal inconveniente es que está basado en XML y no es eficiente en grandes conjuntos de datos [19].
- **GeoJSON**: Es muy sencillo y está basado en el formato JSON. Sin embargo, mediante este formato no es posible representar todos los objetos geométricos ni algunos sistemas de referencia de coordenadas más avanzados [20].
- **GML (*Geography Markup Language*)**: Este formato también está basado en XML y permite implementar representaciones de datos más avanzadas, sin embargo, se trata de un estándar extremadamente complejo. Importantes iniciativas se basan en este formato, por ejemplo, “Inspire” (*Infraestructure for*

Spatial Information in Europe), una iniciativa de la Unión Europea cuyo objetivo es la creación de una infraestructura de datos espaciales en Europa.

- **GPKG (GeoPackage)**: Es un formato definido por la OGC (*Open Geospatial Consortium*) como un "*estándar abierto, multiplataforma y compacto, para la transferencia de información geoespacial*" [21].

Entre todos estos formatos, GeoPackage es uno de los más prometedores y cuenta con el apoyo de múltiples gobiernos, instituciones y organizaciones de código abierto. Este formato está construido sobre una base de datos SQLite y admite grandes volúmenes de datos. Además, es capaz de almacenar datos complejos y múltiples tipos geométricos incluyendo: *Point*, *Line*, *Polygon*, *MultiPoint*, *MultiLine*, *MultiPolygon*, *CompoundCurve*, *CurvedPolygon*, *MultiCurve* y *MultiSurface*.

En el desarrollo de este proyecto se han utilizado tanto datos en formato Shapefile como en formato GeoPackage.

2.3. Modelos de aprendizaje automático

En la actualidad, existen numerosos algoritmos de aprendizaje automático que permiten resolver todo tipo de cuestiones. Los problemas de clasificación, el reconocimiento de imágenes, la detección de anomalías o los motores de recomendación, son solo algunos ejemplos entre un amplio espectro de posibilidades. Estos algoritmos tradicionalmente se han clasificado en dos tipos [22]:

- **Aprendizaje supervisado o *supervised learning*,** donde los datos usados en el entrenamiento están etiquetados, por ejemplo, en el caso de problemas binarios con un 0 o un 1. Algunos modelos muy comunes en esta categoría son: regresiones lineales y logísticas (*linear and logistic regresions*), árboles de decisión (*decision trees*), *random forest*, *gradient boosting*, entre otros.
- **Aprendizaje no supervisado o *unsupervised learning***, donde los datos usados en el entrenamiento del modelo no están etiquetados y por tanto el propio algoritmo deberá ser capaz de clasificar la información por sí solo. Algunos ejemplos de este tipo de algoritmos son: k-medias (*k-means*), PCA (*Principal Components Analysis*), SVC (*Support Vector Clustering*), entre otros.

Sin embargo, en los últimos tiempos han surgido nuevos tipos de clasificaciones que incluyen categorías propias como: **aprendizaje por refuerzo** (***reinforcement learning***), algoritmos que mejoran su comportamiento a través de ciclos de aprendizaje donde

buscan obtener la máxima recompensa; **redes neuronales y *deep learning***, algoritmos basados en los principios de los perceptrones multicapa; ***ensemble learning***, algoritmos que se basan en combinar numerosos modelos sencillos, entrenados independientemente, para construir un modelo más robusto que mejore el rendimiento global.

En este proyecto, se necesita resolver dos tipos de problemas: por un lado, será necesario predecir el número de taxis disponibles en las distintas áreas en un momento temporal determinado, por otro lado, es necesario estimar el tiempo de viaje en taxi entre un origen (la ubicación del taxi) y un destino (el punto de recogida del usuario) en un momento dado.

En ambos problemas, los datos históricos cuentan con un objetivo etiquetado. Por un lado, conocemos el número exacto de taxis que, históricamente, han estado disponibles en ciertas áreas y momentos temporales, por otro lado, contamos con un histórico de viajes entre diferentes áreas con el tiempo de duración del trayecto.

Por ello, ambos problemas podrían resolverse utilizando un enfoque tradicional basado en algoritmos clásicos de aprendizaje supervisado. Sin embargo, en los últimos años, ha crecido la popularidad de los algoritmos de *ensemble learning*, en general, este tipo de algoritmos ofrecen unos resultados más precisos y reducen la tasa de error en numerosos escenarios [23], aunque son muy intensivos en cuanto a potencia de cálculo se refiere. Una de las grandes barreras de este tipo de algoritmos han sido las limitaciones computacionales, sin embargo, la evolución tecnológica ha ido reduciendo este problema, facilitando la adopción y el desarrollo de este tipo de algoritmos en los últimos años.

Como se ha mencionado anteriormente, los algoritmos *ensemble* se basan en la combinación de múltiples modelos para obtener mejores resultados, combinando el conocimiento adquirido por todos ellos [24].

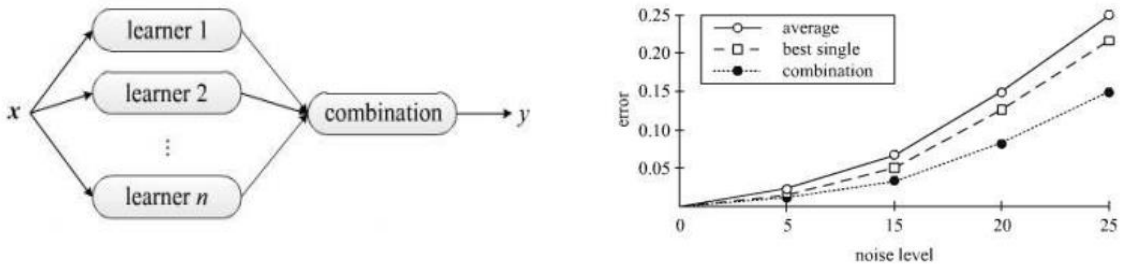

Ilustración 6. Ensemble learning algorithms [24].

Este tipo de modelos permiten resolver problemas tanto de clasificación (*classification*) como de estimación (*regression*). Dentro de esta categoría existen principalmente tres técnicas [25]:

- ***Bagging***: Mediante esta técnica, los modelos se entrenan en paralelo y la combinación de las predicciones de todos ellos se utiliza para calcular la predicción final. En esta categoría tenemos modelos como el *random forest*.
- ***Boosting***: Con esta técnica, los modelos se entrenan secuencialmente, de forma que cada uno de los diferentes modelos trata de corregir los errores del anterior. En este grupo se incluyen modelos como *XGBoost*.
- ***Stacking***: Con esta técnica, se entrena un conjunto de modelos y se utiliza su salida como entrada para entrenar otro modelo que combine todas las predicciones.

En este proyecto, hemos aplicado dos de estos algoritmos para resolver los problemas definidos. Por un lado, se ha utilizado un modelo de *bagging*, *random forest,* para predecir el número de taxis disponibles en un área. Por otro lado, se ha utilizado un algoritmo de *boosting*, XGBoost, para entrenar un modelo capaz de estimar el tiempo de viaje entre un origen y un destino.

Respecto a ***random forest*** *[26]*, es un tipo de algoritmo de *bagging* que crea numerosos árboles de decisión en paralelo y combina sus resultados para obtener predicciones más precisas y estables. A la hora de entrenar estos árboles de decisión, cada árbol depende de los valores de un vector aleatorio muestreado independientemente. Este algoritmo se puede aplicar para resolver tanto problemas de regresión como de clasificación.

Por otro lado, ***XGBoost*** *[27]*, es un algoritmo de *boosting* que combina secuencialmente una serie de árboles de decisión, de forma que cada árbol intentará corregir el error del árbol anterior. Para ello, los árboles se construyen utilizando los residuos en lugar de las etiquetas iniciales, esto es la diferencia entre el valor observado frente al predicho. *XGBoost* es una implementación específica de *gradient boosting*, que busca mejorar tanto la velocidad de ejecución como el rendimiento del modelo. Al igual que en el caso de *random forest*, se puede aplicar para resolver problemas tanto de clasificación como de regresión.

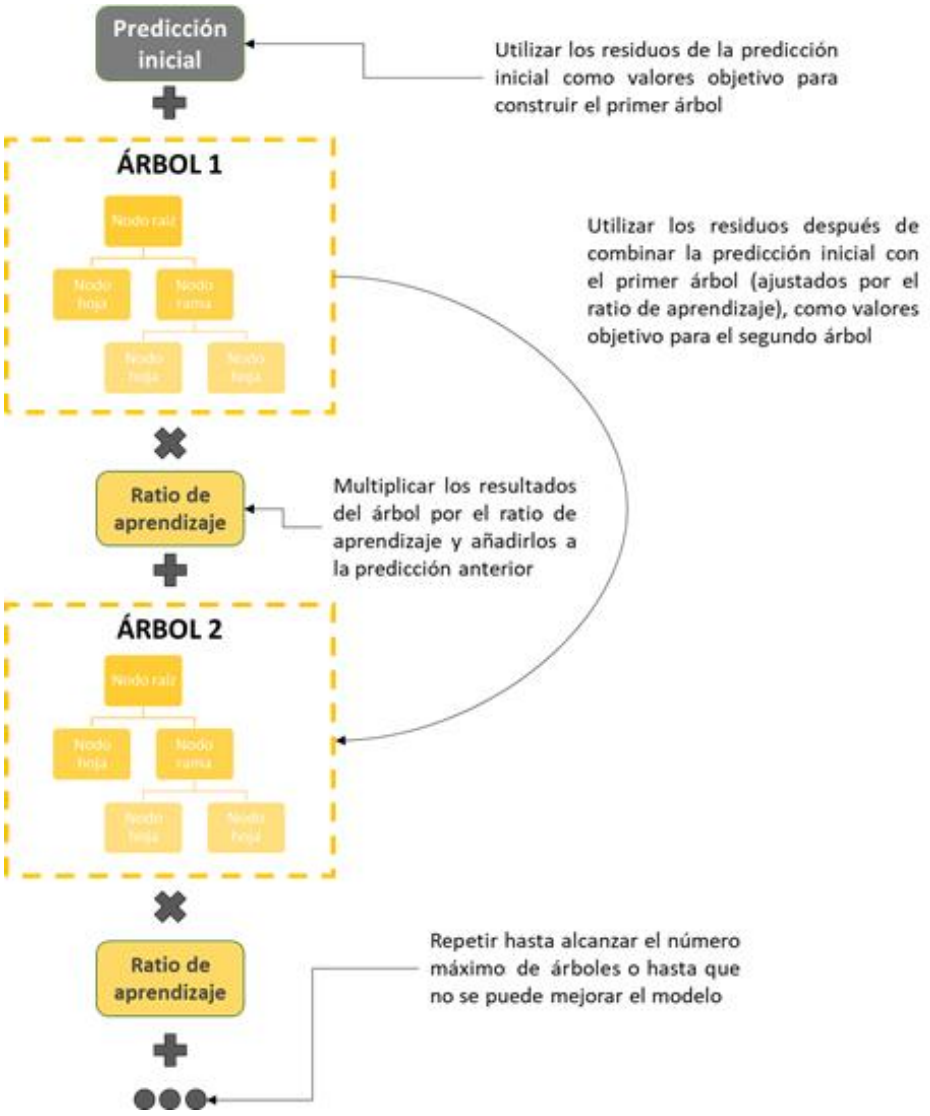

Ilustración 7. Fases de aprendizaje en algoritmos de gradient boosting [27].

La principal diferencia de *XGBoost* frente al *gradient boosting* tradicional es que mientras que este último utiliza métricas tradicionales para decidir las divisiones óptimas de los árboles, por ejemplo, tratando de minimizar el MSE (*Mean Squared Error* o error cuadrático medio) en cada nodo. Por otro lado, *XGBoost* utiliza sus propias métricas [27]:

$$Gain = Left\ leaf_{similarity} + Right\ leaf_{similarity} - Root_{similarity}$$

Donde:

$$Similarity = \frac{(\sum_{i=1}^{n} Residual_i)^2}{\sum_{i=1}^{n}[Prev.Probaility_i \times (1 - Prev.Probability_i)] + \lambda}$$

De forma que se busca encontrar el nodo donde la función de ganancia es mayor.

2.4. Plataformas *cloud*

Los requisitos y características del proyecto hacen necesario disponer de un sistema con alta disponibilidad, capaz de recolectar datos en tiempo real y almacenarlos eficientemente. Además, este sistema debe ofrecer la capacidad de procesar esta información y realizar diferentes análisis sobre los datos. Debido a estos requisitos y al alto nivel de complejidad de algunas de las operaciones requeridas para transformar y analizar la información, será necesaria una alta capacidad de computación en ciertos periodos del proceso. Además, este sistema debe ser fácilmente escalable para adaptarse al crecimiento constante en el volumen de los datos capturados. Por estos motivos, se ha optado por desarrollar el proyecto sobre una plataforma *cloud*.

Este tipo de plataformas ofrecen la flexibilidad de escalar fácilmente la capacidad cuando es necesario, además, cuentan con una serie de servicios que permiten al usuario abstraerse de ciertas labores de instalación, configuración y gestión de la infraestructura. Actualmente, las grandes compañías tecnológicas son las líderes de este sector, Amazon con Amazon Web Services (AWS) [28]; Microsoft con Microsoft Azure [29] y Google con Google Cloud Platform (GCP) [30]. Estas compañías cuentan con productos maduros y ofrecen funcionalidades más avanzadas que sus competidores. Uno de los estudios con mayor reconocimiento en este ámbito es el elaborado periodicamente por Gartner, en su última publicación se destaca el liderazgo de estas tres plataformas [31].

Ilustración 8. Cuadrante de Gartner para infraestructuras y plataformas cloud [31].

En el "Anexo A. Análisis de plataformas *cloud*" se ha incluido un detallado análisis comparando las diferentes alternativas del sector. Como conclusión del estudio realizado, cabe destacar que estas plataformas basadas en *cloud* pública destacan frente al resto de competidores tanto en sus capacidades, como en la madurez de los servicios ofrecidos. Dentro de estas plataformas, Amazon Web Services está un paso por delante de Microsoft y Google y es la que cuenta con una mayor variedad de servicios y tiene un mayor nivel de madurez en sus productos [31], además, ofrece un gran número de componentes relevantes para este proyecto de forma gratuita. Por ello, Amazon Web Services (en adelante AWS) ha sido la plataforma elegida para desarrollar el proyecto.

2.5. Librerías de Python

Python es uno de los lenguajes de programación más populares en el análisis, tratamiento y procesamiento de datos. Es de código abierto y cuenta con numerosas librerías que lo convierten en una herramienta muy poderosa. Entre todas las librerías, en las siguientes páginas se profundiza en algunas de las más relevantes para el desarrollo del proyecto.

2.5.1. Pandas y Numpy

Pandas y Numpy son dos de las librerías más populares en Python y su uso está ampliamente extendido en proyectos relacionados con el análisis de datos debido a las potentes funcionalidades que ofrecen en el tratamiento de estos [32][33].

Por su parte, Pandas facilita el tratamiento y análisis de datos mediante series (una dimensión) y estructuras de datos conocidas como *dataframes* (dos dimensiones). Además, proporciona funciones que permiten cargar información en estos *dataframes* desde diferentes tipos de ficheros y cuenta con operaciones que permiten manipular y analizar estos datos [34][35]. Algunas de las funcionalidades clave de Pandas que se han utilizado en este proyecto son:

- El tamaño de los *dataframes* puede alterarse, es decir, se pueden insertar y/o eliminar columnas.
- Cuenta con funciones que permiten transformar fácilmente otros tipos de estructuras Python en *dataframes*.
- Permite realizar operaciones de agregación sobre los datos y aplicar diferentes cálculos sobre los distintos campos.

- Es posible combinar los datos de diferentes *dataframes* o dividir un único *dataframe* en varios.
- Facilita el tratamiento de datos vacíos (*missing data*).
- Dispone de herramientas para cargar datos desde diferentes tipos de ficheros como CSV, JSON o Parquet o desde diferentes bases de datos.
- Permite filtrar los datos en base a uno o más campos.

Por otro lado, Numpy [36] es una librería especializada en el cálculo matemático que incluye soporte para tratar de forma eficiente grandes volúmenes de datos. Cuenta con diferentes operaciones para manipular datos en *arrays* y matrices multidimensionales [35]. Esta es una librería fundamental y es la base de muchas otras, incluida Pandas.

Es decir, Pandas ofrece las herramientas necesarias para manipular datos a alto nivel con operaciones matemáticas creadas sobre Numpy.

2.5.2. GeoPandas

Geopandas es una librería que permite interpretar, transformar y almacenar datos geoespaciales. Esta librería extiende los *dataframes* de Pandas, mencionados en el apartado anterior (2.5.1), y facilita la manipulación datos geométricos mediante operaciones espaciales.

Cuenta con un tipo de datos especial llamado *geodataframes* que incluye las características propias de los *dataframes* de Pandas y le añade un nuevo elemento, los objetos geométricos de Shapely (una librería especializada en la manipulación y el análisis de objetos geométricos en el plano cartesiano) mediante un campo adicional llamado *geometry*. Es decir, con las funciones de Geopandas es posible manipular objetos geométricos como puntos, líneas o polígonos [41].

2.5.3. H3 (Uber)

H3 es una librería basada en un sistema desarrollado por Uber conocido como "*Uber's Hexagonal Hierarchical Spatial Index*". Este sistema permite dividir el mundo en una cuadrícula de celdas hexagonales [42].

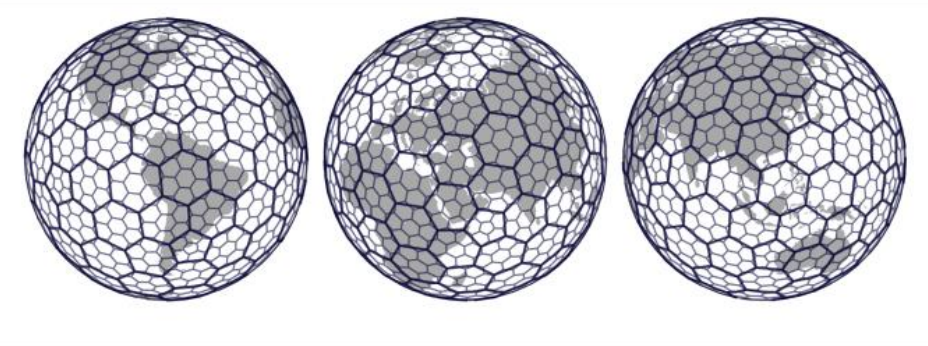

Ilustración 9. Visión del mundo dividido en áreas [42].

A la hora de analizar grandes conjuntos de datos asociados a localizaciones, es necesario agrupar la información. Se pueden realizar varios tipos de agrupaciones. Por un lado, aplicando criterios como los límites administrativos o políticos. Esto genera importantes problemas desde una perspectiva matemática. Por otro lado, es posible generar estas áreas mediante cuadrículas regulares que forman mosaicos, generalmente compuestos por celdas con formas cuadradas o rectangulares. El problema con este tipo de celdas es que una misma celda tiene varios vecinos con diferentes distancias. En el caso de las triangulares, cada celda tiene vecinos con tres distancias diferentes. Por su parte, las celdas cuadrangulares cuentan con vecinos con dos tipos de distancias. Sin embargo, en el caso de las celdas hexagonales se soluciona este problema, puesto que todos sus vecinos son equidistantes.

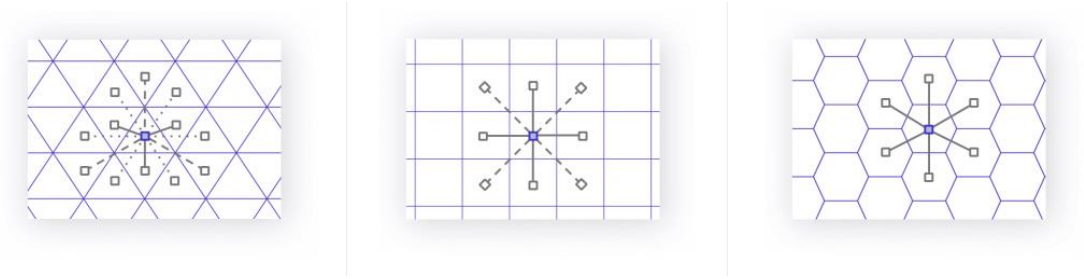

Ilustración 10. Distancia entre una celda y sus vecinos [42].

Por otro lado, el sistema propuesto por Uber está jerarquizado en 16 niveles, donde cada celda se puede subdividir en 7 hexágonos menores. Lo que permite realizar operaciones aumentando o reduciendo la precisión del sistema mediante celdas padres e hijas.

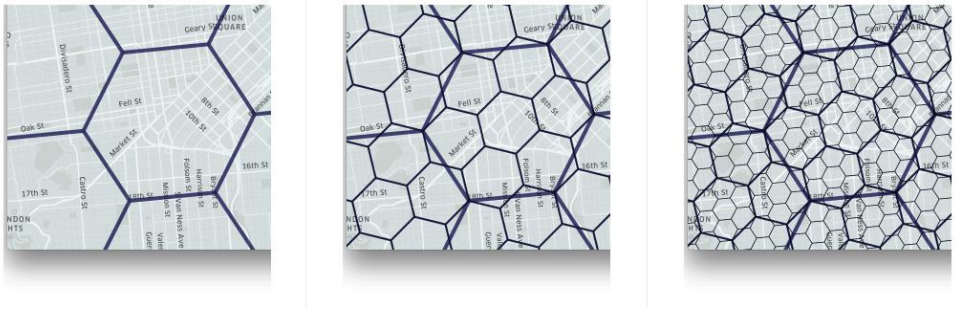

Ilustración 11. Jerarquía en niveles de las celdas [42].

En la siguiente tabla (Tabla 2) se muestran los niveles de precisión definidos en la librería, desde el mínimo, el nivel 0, formado por 122 celdas y donde cada hexágono comprende una región de más de 4 millones de kilómetros cuadrados. Hasta llegar al máximo de precisión, el nivel 15, formado por más de 500 billones de celdas y donde cada hexágono abarca un área inferior a 1 metro cuadrado.

En este proyecto se ha tomado como punto de referencia la resolución 7, es decir, cada uno de los hexágonos definidos tendrá un área de aproximadamente 5 kilómetros cuadrados.

Tabla 2. Áreas para cada uno de los tamaños de celda en H3 [43].

H3 Resolution	Average Hexagon Area (km^2)	Average Hexagon Edge Length (km)	Number of unique indexes
0	4,250,546.8477000	1,107.712591000	122
1	607,220.9782429	418.676005500	842
2	86,745.8540347	158.244655800	5,882
3	12,392.2648621	59.810857940	41,162
4	1,770.3235517	22.606379400	288,122
5	252.9033645	8.544408276	2,016,842
6	36.1290521	3.229482772	14,117,882
7	5.1612932	1.220629759	98,825,162
8	0.7373276	0.461354684	691,776,122
9	0.1053325	0.174375668	4,842,432,842
10	0.0150475	0.065907807	33,897,029,882
11	0.0021496	0.024910561	237,279,209,162
12	0.0003071	0.009415526	1,660,954,464,122
13	0.0000439	0.003559893	11,626,681,248,842
14	0.0000063	0.001348575	81,386,768,741,882
15	0.0000009	0.000509713	569,707,381,193,162

Por otro lado, cada celda está definida mediante un identificador único, en formato hexadecimal, es decir, cada identificador consta de 16 bits. Estos bits se corresponden con las 16 resoluciones presentes en el sistema.

En la librería h3 de Python, que implementa este sistema, se han identificado las funciones con mayor utilidad para el proyecto de entre todas las disponibles [44]. A continuación, se mencionan algunas de las más utilizadas en el desarrollo del proyecto:

- **geoToH3**: Convertir coordenadas geográficas (latitud y longitud) a la celda H3 que contiene dicha posición.
- **h3ToGeoBoundary**: Encontrar los límites geográficos de una celda concreta.
- **h3ToGeo:** Calcular las coordenadas (latitud y longitud) del centroide de cada celda.
- **h3IndexesAreNeighbors**: Identificar si dos celdas son vecinas, es decir, adyacentes.
- **h3ToParent** / **h3ToChildren**: Encontrar tanto la celda padre como las celdas hijas (en base a la resolución).
- **Polyfill;** Definir el conjunto de todas las celdas contenidas dentro de los límites geométricos de un polígono.
- **h3Distance**: Medir la distancia entre dos áreas, en número de celdas intermedias.
- **h3Line:** Identificar las celdas intermedias entre dos de ellas, en línea recta.
- **pointDistKm**: Proporciona la distancia entre dos puntos, en kilómetros. Este cálculo se realiza mediante la fórmula del semiverseno (o fórmula de Haversine) [45], una ecuación para la navegación astronómica que permite calcular la distancia más corta entre dos puntos sobre la superficie terrestre (mediante su latitud y longitud) teniendo en cuenta la curvatura terrestre. Se calcula de la siguiente forma [45]:

$$d = 2R \arcsin\left(\sqrt{sin^2\left(\frac{Lat_1 - Lat_2}{2}\right) + \cos(Lat_1)\cos(Lat_2)\, sin^2\left(\frac{Long_1 - Long_2}{2}\right)}\right)$$

Donde Lat_1 es la latitud en el punto 1, $Long_1$ es la longitud en el punto 1, Lat_2 es la latitud en el punto 2, $Long_2$ es la longitud en el punto 2 y R es el radio de tierra.

En este proyecto, el sistema de celdas hexagonales desarrollado por Uber nos ha permitido solucionar diversos problemas de forma eficiente. Por un lado, ha facilitado la

creación de áreas óptimas dentro del municipio de Madrid, permitiendo agregar los datos asociados a los taxi disponibles en estas celdas hexagonales y con ello, contribuyendo en el proceso de creación de un modelo capaz de predecir la disponibilidad en una zona y momento concretos.

Por otro lado, gracias a esta propuesta de áreas hexagonales, donde todas las celdas adyacentes son equidistantes respecto al origen, ha sido posible enriquecer la información disponible con nuevas variables, como la distancia entre un origen y un destino medido en base al número de celdas ubicadas entre estas áreas. Esto ha permitido simplificar el análisis y mejorar los resultados obtenidos en el modelo desarrollado para estimar los tiempos de viaje entre diferentes zonas en un momento temporal determinado.

2.5.4. Google Maps

Google ofrece una amplia variedad de APIs a través de sus servicios en Google Cloud Platform (GCP). Dentro de este amplio catálogo de APIs, en este proyecto se han utilizado dos de ellas, "*Directions API*" y "*Distance Matrix API*". Estas APIs utilizan la base de conocimiento de Google para calcular la ruta óptima entre dos puntos y obtener estimaciones del tiempo y la distancia del viaje, teniendo en cuenta los datos del tráfico en tiempo real y modelos que permiten predecir la saturación del tráfico en un momento dado en el futuro. Ambas funciones forman parte de la misma librería "googlemaps" de Python [46].

En el caso de "*Directions API*", los principales parámetros de entrada que se deben configurar son: las coordenadas de origen, las coordenadas de destino, la fecha de inicio del viaje, el modo de viaje (en este caso será "*driving*"), la unidad de medida (en este caso será el sistema métrico, "*metric*") y el modelo de tráfico a aplicar ("mejor estimación", "optimista" o "pesimista"), entre otros. Como resultado, esta función devuelve una o varias rutas entre el punto de origen y el punto de destino, incluyendo la distancia y el tiempo del viaje.

Por otro lado, "*Distance Matrix API*", necesita unos parámetros similares, la principal diferencia es que, admite una lista de orígenes y otra de destinos, en vez de un único valor. Como resultado devuelve una matriz de distancias donde cada registro se corresponde con un origen y un destino e incluye variables similares a la API anterior (distancia, tiempo de viaje con y sin tener en cuenta los datos de tráfico, entre otros), pero sin incluir los detalles del trayecto, es decir, cada uno de los pasos intermedios.

Estas APIs se han utilizado en el proyecto para construir un *dataset* con los tiempos de viaje y la distancia entre dos puntos situados en diferentes áreas y en distintos momentos temporales. Dicha información ampliará el volumen de datos disponibles acerca de viajes entre varios puntos y será utilizado para entrenar el modelo con el que estimar tiempos.

Cada llamada a las APIs de Google tiene un coste asociado, por lo que se dispone de un presupuesto limitado a la hora de crear el *dataset* mencionado. Estos costes varían entre 5 USD (tiempos sin incluir datos del tráfico) y 10 USD (tiempos considerando el tráfico) por cada 1.000 llamadas a la API (una por cada cálculo del tiempo entre origen y destino).

Tabla 3. Precio de las llamadas a la API de Google

API	200 USD de crédito mensual (uso gratuito equivalente)	Intervalo de volumen mensual (Precio por mil)	
		De 0 a 100.000	**De 100.001 a 500.000**
Directions	Hasta 40.000 llamadas	5,00 USD	4,00 USD
Directions Advanced	Hasta 20.000 llamadas	10,00 USD	8,00 USD
Distance Matrix	Hasta 40.000 elementos	5,00 USD	4,00 USD
Distance Matrix Advanced	Hasta 20.000 elementos	10,00 USD	8,00 USD
Roads - Route Traveled	Hasta 20.000 llamadas	10,00 USD	8,00 USD
Roads – Nearest Road	Hasta 20.000 llamadas	10,00 USD	8,00 USD
Roads – Speed Limits	Hasta 2.000 elementos	20,00 USD	16,00 USD

Teniendo en cuenta los créditos de prueba disponibles (300 USD) y la capa gratuita de Google para la API (200 USD mensuales), se cuenta con un presupuesto de 500 USD, lo que supone aproximadamente 50.000 llamadas a la API, es decir, es posible construir un *dataset* con 50.000 registros. Por ello, se ha definido una estrategia, detallada en la sección 3.6.2, para elegir los pares origen-destino en función del tipo de área y la fecha.

2.5.5. Matplotlib y Folium

A la hora de representar datos geográficos en Python existen diferentes alternativas, algunas de las más populares son Matplotlib y Folium. Estas librerías ofrecen potentes herramientas y funciones para facilitar la creación de todo tipo de representaciones visuales de excelente calidad.

Matplotlib [38] se considera casi un estándar en el análisis de datos con Python a la hora de generar gráficos en dos dimensiones a partir de un conjunto de datos. La base de esta librería es Matlab e implementa gran parte de la funcionalidad de esta herramienta simplificando su uso.

El módulo más destacado en Matplotlib es "pyplot". Este módulo proporciona una interfaz para crear figuras y ejes de forma implícita y automática. Además, permite configurar numerosos detalles en el aspecto del gráfico, incluyendo la leyenda, los colores, el estilo de las líneas, los títulos, los ejes, etc.

En este proyecto se ha usado esta librería tanto para representar los gráficos sobre mapas geográficos, donde las coordenadas de latitud y longitud se representan sobre los ejes vertical y horizontal del gráfico, como para realizar diferentes análisis sobre el conjunto de datos mediante diagramas de líneas y barras o histogramas, entre otros.

Además, esta librería se integra perfectamente con los paquetes de Pandas y GeoPandas, lo que nos permite generar las representaciones gráficas directamente a partir de los datos incluidos en los diferentes *dataframes*.

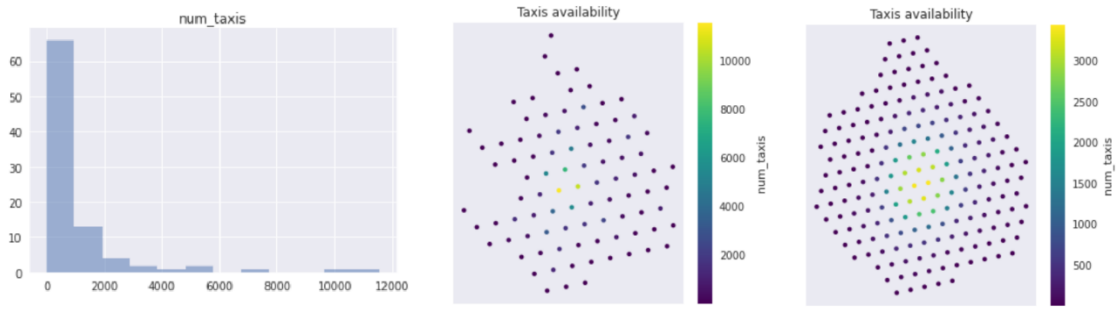

Ilustración 12. Representación de datos mediante Matplotlib

Por último, es posible añadir un mapa geográfico como base de las representaciones creadas con Matplotlib mediante la librería Contextily. Para ello, es necesario establecer el sistema de referencia de coordenadas (CRS) que relaciona la proyección de estos puntos en dos dimensiones con ubicaciones reales de la tierra. Se admiten dos sistemas de referencia EPSG (*European Petroleum Survey Group*) diferentes: WGS84 (EPSG:4326, sistema mundial para dispositivos GPS) y Web Mercator (EPSG:3857, desarrollado por Google para Google Maps y utilizado por los principales sistemas de cartografía en internet). Finalmente, aunque el proveedor estándar para los mapas es OpenStreetMap, es posible utilizar numerosos proveedores diferentes.

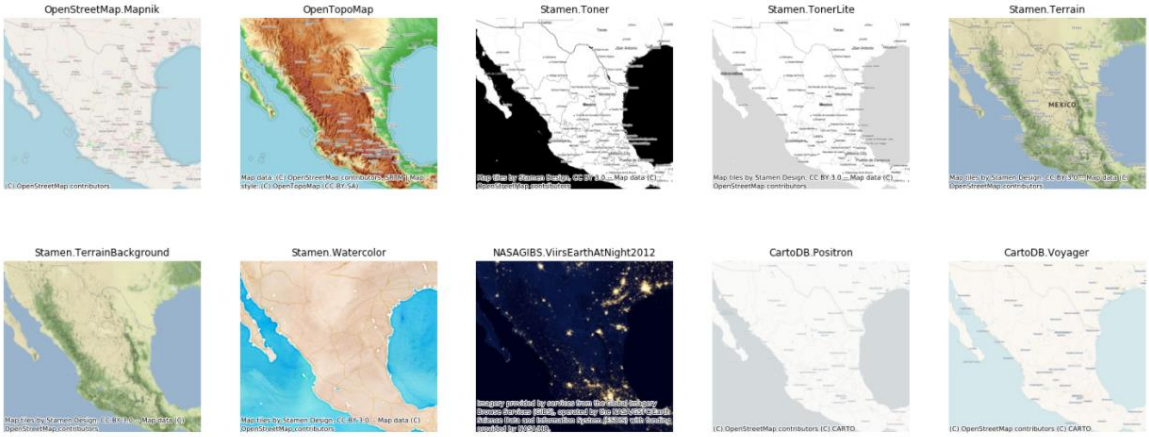

Ilustración 13. Proveedores de mapas en Contextily [39].

Por otro lado, Folium [40] es otra librería de Python que permite generar representaciones visuales a partir de datos geográficos. La gran ventaja de esta librería es que permite crear mapas interactivos, frente a las representaciones estáticas de Matplotlib. Para ello utiliza una potente librería de JavaScript, llamada Leaflet.js, que permite publicar mapas interactivos en la web.

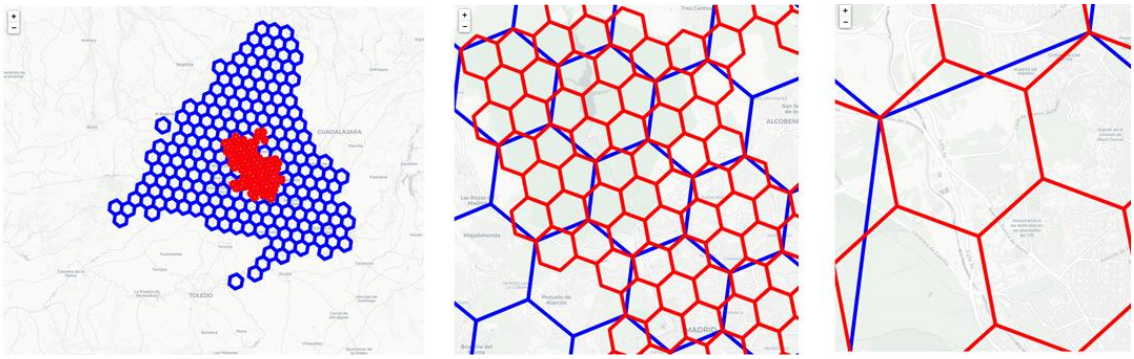

Ilustración 14. Representación de datos mediante Folium.

2.5.6. Sklearn y XGBoost

Python cuenta con un amplio repertorio de librerías de aprendizaje automático, que facilitan el desarrollo de modelos predictivos, algunas de las más utilizadas son Sklearn y XGBoost. Por un lado, Scikit-learn (Sklearn) es una de las librerías más populares de *machine learning*, combina la sencillez de su uso con un elevado rendimiento y está construida sobre Matplotlib, NumPy, SciPy. Incluye mecanismos para el pre-procesado de los datos y admite técnicas de aprendizaje supervisado y no supervisado. Esta librería implementa diferentes algoritmos de clasificación, regresión y *clustering*, que permiten

entre otras funcionalidades: entrenar árboles de decisión, realizar análisis y seleccionar características, desarrollar clasificaciones y agrupaciones en clústeres no supervisados o detectar valores atípicos.

En este proyecto, se ha utilizado la librería Sklearn para: preparar los datos de entrada al modelo, por ejemplo, dividiendo los conjuntos de entrenamiento, validación y prueba; para entrenar el modelo basado en *random forest* (mediante la función "RandomForestRegressor") con el que predecir la disponibilidad de taxis en las diferentes áreas de Madrid y para validar los resultados de ambos modelos mediante métricas como R^2, MSE, RMSE, MAE, *cross-validation* o K-fold *cross-validation*.

Por otro lado, XGBoost (*Extreme Gradient Boosting*) es una librería diseñada para ser muy efectiva a la par que flexible, que emplea algoritmos de *gradient boosting*, es decir, que buscan optimizar una función objetivo para ponderar los modelos generados y mejorar los resultados. En los últimos tiempos ha crecido mucho su popularidad debido a los éxitos alcanzados resolviendo diversos problemas en competiciones de *machine learning*, además, ha conseguido demostrar grandes resultados en diferentes *benchmarks* donde se analizaron distintos algoritmos de aprendizaje automático. Esta librería únicamente toma valores numéricos como entrada, por lo que es necesario transformar los otros tipos de datos, por ejemplo, los categóricos. Trabajar únicamente con datos numéricos es algo que le ayuda a ser tan eficiente.

En este proyecto se ha utilizado esta librería para entrenar un modelo de *gradient boosting* mediante la función "XGBRegressor" con el que estimar los tiempos de duración de los viajes realizados entre un origen y un destino en un momento dado.

2.5.7. AwsWrangler

AWS Data Wrangler es una librería de Python que facilita su integración con diferentes servicios de AWS. En este proyecto se ha usado esta librería para recuperar los datos de las tablas de AWS Athena (un servicio de análisis de datos mediante consultas SQL) y los ficheros de datos almacenados en AWS S3 (un servicio de almacenamiento de objetos en la nube) en diferentes formatos (csv, parquet, shp y gpkg) y para cargar esta información en *dataframes* de Pandas o *geodataframes* de GeoPandas.

3. Diseño y desarrollo

En este capítulo, se detallará la forma en que se ha diseñado y ejecutado la solución al problema planteado: ser capaces de estimar el tiempo de recogida de un usuario en un tiempo de respuesta mínimo (ms). Para ello, se detallará la arquitectura propuesta y se profundizará en las diferentes fases llevadas a cabo en el desarrollo del proyecto.

En la siguiente ilustración (Ilustración 15) se detalla un esquema, a alto nivel, de cada una de las fases del proyecto: desde los procesos de extracción y preparación de los datos, hasta el análisis de la información, el desarrollo de los modelos predictivos y la puesta en producción. Además, en las siguientes páginas se profundizará en cada uno de estos componentes.

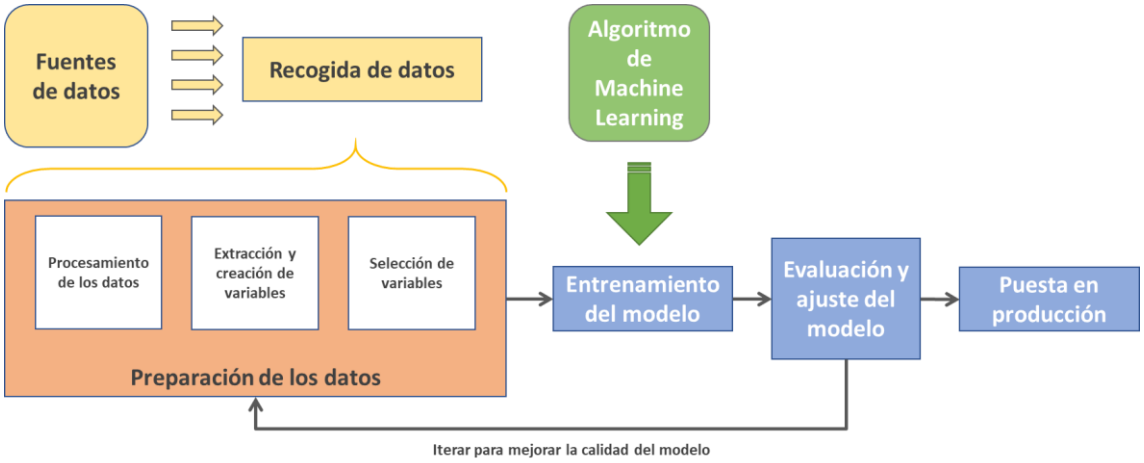

Ilustración 15. Fases en el desarrollo de los modelos.

3.1. Infraestructura y arquitectura

La infraestructura necesaria para la realización del proyecto se ha implementado en el *cloud* público de AWS, como ya se detalló en la sección 2.4.

En el siguiente diagrama (Ilustración 16) se pueden ver los diferentes componentes que forman parte de la arquitectura con la que se cubrirán todas las fases del desarrollo del proyecto: la ingesta de datos desde los orígenes, el procesamiento de los datos, el análisis de la información, la creación de los modelos predictivos y la integración y puesta en producción del sistema.

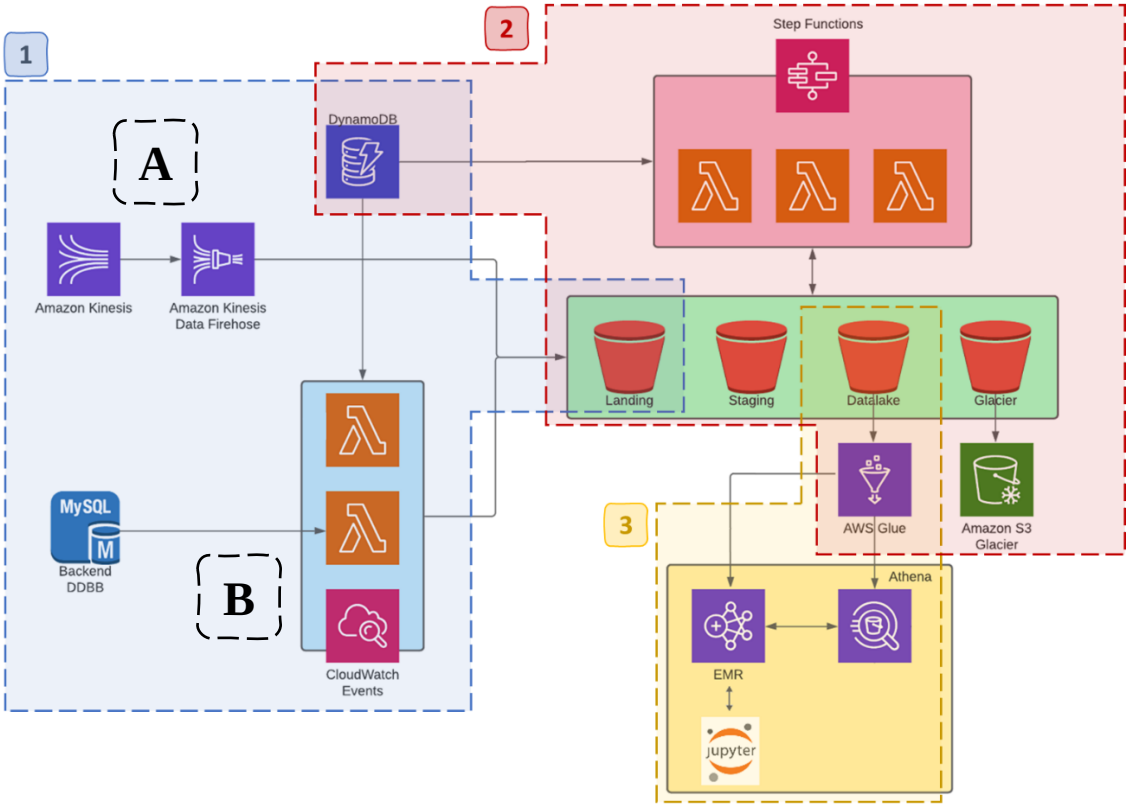

Ilustración 16. Diagrama de la infraestructura desarrollada en AWS.

Como se puede apreciar en la imagen anterior (Ilustración 16) es posible distinguir tres partes dentro de la arquitectura en función de su finalidad. Por un lado, la primera de ellas (resaltada en azul) se encarga de capturar los datos, en función del tipo de origen podemos distinguir a su vez dos partes (identificadas en la ilustración, Ilustración 16, como A y B):

A. Los datos que se envían constantemente en *streaming* desde un origen (el servicio de taxis) son capturados mediante AWS Kinesis Data Streams (un servicio de colas de mensajes). A continuación, se almacena la información utilizando AWS Kinesis Firehose (un servicio para procesar datos en *streaming*) realizando las mínimas transformaciones, mediante una función AWS Lambda (un servicio de AWS que permite ejecutar programas en código Python, entre otros lenguajes) que transforma los ítems recibidos en formato JSON a formato CSV. Finalmente, los datos se envían desde Firehose para ser almacenados en un *bucket* (contenedor de objetos) de AWS S3 (servicio de almacenamiento de objetos), identificado como "*Landing*".
B. Los datos se extraen desde una base de datos relacional (MySQL) mediante una función AWS Lambda, que ejecuta un programa en Python para conectarse a la

base de datos y acceder a la información actualizada. Esta función se ejecuta periódicamente mediante una regla configurada en AWS Cloudwatch (un sistema que permite programar eventos automáticos que se ejecutan con una periodicidad definida). Finalmente, los datos se almacenan en un *bucket* de AWS S3 identificado como "*Landing*", al igual que en el caso anterior

En la sección 3.3 se ha profundizado al detalle en cada uno de los elementos de la ingesta.

La segunda parte de la infraestructura (resaltada en rojo en la ilustración, Ilustración 16) se encarga de recoger la información almacenada en el *bucket* de "*Landing*", procesa esta información aplicando las transformaciones necesarias (por ejemplo, modificaciones en el tipo de datos en ciertos campos), convierte los datos a formato parquet, almacena estos datos organizados en particiones en el *bucket* identificado como "*Datalake*" y, finalmente, actualiza el catálogo de datos. En la sección 3.4 se analiza en detalle este proceso.

Por último, la tercera parte (resaltada en amarillo en la ilustración, Ilustración 16), consta de varios componentes enfocados al análisis de datos y la elaboración de modelos predictivos. Por un lado, tenemos AWS Athena que consiste en un servicio de consultas que permite analizar los datos almacenados en AWS S3 mediante lenguaje SQL estándar, utilizando los metadatos (tablas, campos, etc.) definidos en un catálogo de datos denominado AWS Glue. Por otro lado, existe un clúster de AWS EMR (una plataforma administrada de clústeres destinada al procesamiento de datos) con Jupyter Hub, a través de una aplicación web podemos crear diferentes Jupyter Notebooks donde analizar los datos y desarrollar modelos predictivos, utilizando lenguaje de programación Python y sus diferentes librerías.

Como se ha mencionado anteriormente, en las siguientes secciones de este capítulo se profundizará en cada una de estas partes, analizando todos los componentes que forman parte de la ingesta y el procesamiento de datos y detallando cada uno de los pasos realizados en la creación de los modelos predictivos.

3.2. Fuentes de datos

En la elaboración de este proyecto se han utilizado diversas fuentes de datos, tanto internas como externas (publicadas en portales de datos abiertos). En las siguientes páginas se analizará en detalle cada una de ellas.

3.2.1. Datos asociados al servicio de taxis

Las fuentes de datos asociadas al taxi provienen de una empresa privada dedicada a agregar diferentes servicios de movilidad en una única aplicación móvil, incluyendo transporte público, taxi y vehículos eléctricos compartidos. En este proyecto se han utilizado datos procedentes de este servicio del taxi, concretamente, (1) información sobre la ubicación de los taxis disponibles y (2) datos acerca de los viajes realizados en taxi.

Por un lado, el primer conjunto de datos contiene información acerca de los taxis disponibles en cualquier ubicación de la Comunidad de Madrid en un momento dado. El conjunto de datos se ha generado a partir de la información capturada en tiempo real durante un periodo de 3 meses. Esto supone un total de 276.380 registros asociados a un total de 125 taxis diferentes.

Los datos asociados a cada taxi disponible son enviados periódicamente en formato JSON. Entre los datos capturados se incluyen detalles como la posición del taxi, en latitud y longitud, su identificador único o la fecha de referencia.

- **ID**: Identificador único asignado a cada taxi.
- **FECHA_POSICIÓN**: Fecha en formato *timestamp* en que se ha recogido el dato.
- **CIUDAD**: Nombre de la ciudad asociada al taxi.
- **LAT**: Coordenadas, latitud del taxi en el momento de capturar los datos.
- **LON**: Coordenadas, longitud del taxi en el momento de capturar los datos.

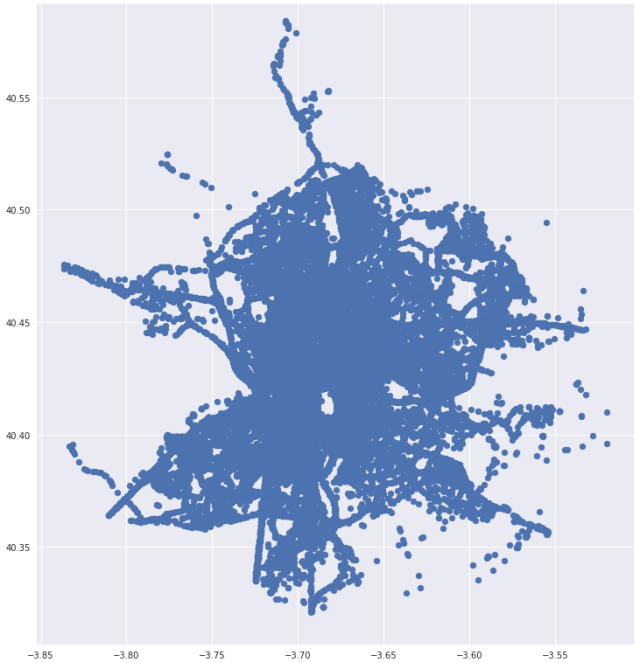

Ilustración 17. Coordenadas de los taxis disponibles

Este *dataset* será utilizado para construir el modelo con el que predecir la tasa de disponibilidad de los taxis en cada una de las áreas de Madrid en un momento determinado.

Por otro lado, el segundo conjunto de datos contiene información acerca de los viajes realizados en taxi, incluyendo origen, destino, fecha del trayecto, detalles del usuario, detalles del taxi, costes asociados al viaje e información del trayecto. Cuenta con un total de 159 registros, cada uno de los cuales corresponde con un viaje realizado en taxi. Entre los campos más importantes para este proyecto destacan los siguientes:

- **ID**: Identificar único del viaje.
- **USER_ID**: Identificador único del usuario.
- **TAXI_ID**: Identificador único del taxi.
- **STATUS**: Último estado conocido del viaje (cancelado, finalizado, en curso, esperando al conductor, esperando al usuario, etc.).
- **ORIGIN_LATITUDE**: Latitud asociada al punto de origen seleccionado.
- **ORIGIN_LONGITUDE**: Longitud asociada al punto de origen seleccionado.
- **DESTINATION_LATITUDE**: Latitud asociada al punto de destino indicado.
- **DESTINATION_LONGITUDE**: Longitud asociada al punto de destino indicado.
- **CREATED_AT**: Fecha y hora de solicitud del viaje.
- **PICKUP_TIME**: Fecha y hora de la recogida del usuario en el origen.
- **ARRIVAL_TIME**: Fecha y hora de llegada al destino.
- **ESTIMATED_ARRIVAL_TIME**: Fecha estimada de llegada al destino por el planificador.
- **PLAN_ESTIMATED_TIME**: Duración del trayecto estimada por el planificador.
- **PLAN_ESTIMATED_DISTANCE**: Distancia del trayecto estimada por el planificador.

Originalmente se había planteado utilizar esta fuente de datos para construir el modelo con las estimaciones de tiempos entre dos puntos, sin embargo, el volumen de datos actual de este *dataset* es demasiado reducido como para utilizarlo en la elaboración de un modelo predictivo. Por lo tanto, ha sido necesario encontrar información adicional a

través de otra fuente de datos. En este caso se ha utilizado la API de Google, como se detalla en el apartado 3.2.2.

Por otro lado, este *dataset* se utilizará para realizar las pruebas finales, midiendo la fiabilidad de las predicciones en un entorno real. Para ello, se han utilizado los puntos de origen y destino y se ha calculado el tiempo de recogida como la diferencia entre la fecha de solicitud del viaje y la fecha de recogida, para así poder comparar el resultado real con la estimación calculada mediante la combinación de los modelos desarrollados.

3.2.2. Datos de viajes entre dos puntos (API Google Maps)

A través de las APIs de Google ("*Directions API*" y "*Distance Matrix API*") es posible obtener el tiempo de viaje entre varios puntos en el modo de transporte indicado y considerando los datos del tráfico. Este tipo de API podría resolver el segundo de nuestros problemas, estimar el tiempo de viaje entre un origen (posición del taxi) y un destino (punto de recogida del usuario), sin embargo, las limitaciones en cuanto al tiempo de respuesta (ms) hacen inviable el uso de este servicio. Por ello, se ha construido una nueva fuente de datos con 49.803 registros, donde cada registro se corresponde con un trayecto entre un punto de origen y un punto de destino. Esta fuente de datos incluye los siguientes campos:

- **DEPARTURE_TIME**: Fecha del trayecto.
- **ORIGIN_LAT**: Coordenadas de origen, latitud.
- **ORIGIN_LON**: Coordenadas de origen, longitud.
- **ORIGIN_ADDRESS**: Dirección de origen (calle, CP, ciudad y país).
- **ORIGIN_ID**: Área de origen (identificador del hexágono H3).
- **DESTINATION_LAT**: Coordenadas de destino, latitud.
- **DESTINATION _LON**: Coordenadas de destino, longitud.
- **DESTINATION _ADDRESS**: Dirección de destino (calle, CP, ciudad y país).
- **DESTINATION _ID**: Área de destino (identificador del hexágono H3).
- **DISTANCE_METERS**: Distancia recorrida, en metros.
- **DURATION_SECONDS**: Tiempo del viaje sin considerar el tráfico, en segundos.
- **DURATION_IN_TRAFFIC_SECONDS**: Tiempo del viaje considerando el tráfico, en segundos.

En la sección 3.6.2 se detalla la forma en que se ha construido esta fuente de datos mediante la API de Google, así como la estrategia seguida para seleccionar cada uno de los pares origen-destino y la fecha del trayecto.

3.2.3. Datos geoespaciales, áreas administrativas (GADM)

GADM (*Database of Global Administrative Areas*) es una base de datos abiertos que incluye las áreas administrativas de la mayoría de los países del mundo. Estos datos se dividen en varios niveles y utilizan el sistema de referencia de coordenadas WGS-84 (también conocido como WGS 1984 o EPSG:4326).

Los diferentes niveles administrativos de España que ofrece esta fuente de datos abarcan desde los límites geográficos que delimitan el propio país hasta un nivel mínimo a escala municipal, incluyendo delimitaciones de comunidades autónomas, provincias y regiones.

Ilustración 18. Divisiones geográficas en España [51].

Este dataset [51], contiene 8MB de datos y se puede descargar tanto en formato Geopackage, como en formato Shapefile. Este conjunto de datos está formado por varias capas y cada una de ellas contiene diferentes campos. En este proyecto se ha utilizado la capa 2 (comunidad autónoma) y la capa 4 (municipio) con los siguientes campos:

- **NAME_0**: Descripción del nivel 0, el nombre del país.
- **NAME_1**: Descripción del nivel 1, el nombre de la comunidad autónoma.
- **NAME_2**: Descripción del nivel 2, el nombre de la provincia. No disponible en la capa 2 (comunidad autónoma).

- **NAME_3**: Descripción del nivel 3, el nombre de la región. No disponible en la capa 2 (comunidad autónoma).
- **NAME_4**: Descripción del nivel 4, el nombre del municipio. No disponible en la capa 2 (comunidad autónoma).
- **GEOMETRY**: Delimitación geográfica del área a nivel de país, comunidad autónoma, provincia, región o municipio, en función de la capa asociada.

A partir de este conjunto de datos, ha sido posible transformar la información mediante diferentes librerías como GeoPandas y generar visualizaciones mediante otras librerías gráficas como Matplotlib o Folium. Por ejemplo, podemos obtener fácilmente los polígonos que delimitan la comunidad de Madrid, como se puede observar en la *Ilustración 19*. Por otro lado, como se puede ver en la *Ilustración 20*, desde la capa 4, también es posible acceder a los datos asociados a los límites administrativos de los diferentes municipios, en este caso el término municipal de Madrid o calcular las delimitaciones asociadas al resto de municipios pertenecientes a la Comunidad de Madrid, como se puede apreciar en la *Ilustración 21*.

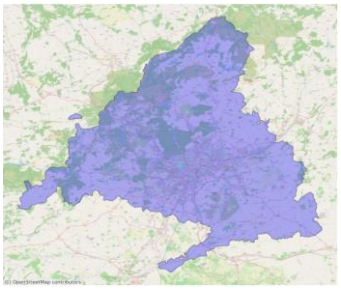

Ilustración 19. Área geográfica de la CCAA de Madrid,

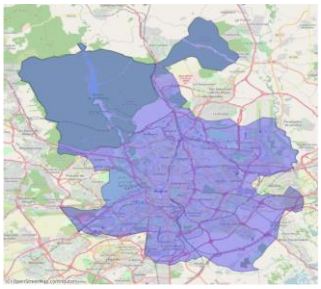

Ilustración 20. Área geográfica del municipio de Madrid,

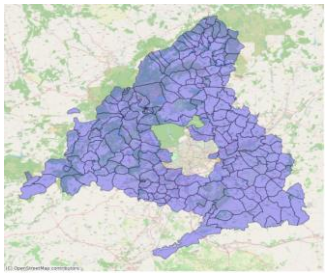

Ilustración 21. Otros municipios de la CCAA de Madrid

3.2.4. Datos geoespaciales, áreas de Madrid (geoportal del ayuntamiento)

El geoportal del ayuntamiento de Madrid [52], ofrece sus propios conjuntos de datos geográficos con información detallada a nivel municipal. Entre toda la información disponible, podemos encontrar datos acerca de los 131 barrios que forman parte del municipio de Madrid, incluyendo sus delimitaciones geográficas, los nombres y códigos de cada barrio o el distrito al que pertenecen. Esta información está disponible en diferentes formatos como KML, TXT, CSV o SHP (este último ha sido el utilizado en este proyecto). Este conjunto de datos incluye diferentes campos, entre los que se destacan por su utilización en la elaboración del proyecto los siguientes:

- **OBJECTID**: Identificador del objeto geométrico.
- **CODDIS**: Código asociado al distrito.
- **NOMDIS**: Nombre del distrito.
- **CODBAR**: Código asociado al barrio.
- **NOMBRE**: Nombre del barrio.
- **GEOMETRY**: Delimitación geográfica de los distintos barrios mediante polígonos.

En la siguiente ilustración (Ilustración 22) se muestra una representación visual, mediante la librería Matplotlib, de los datos obtenidos de esta fuente, los barrios de Madrid.

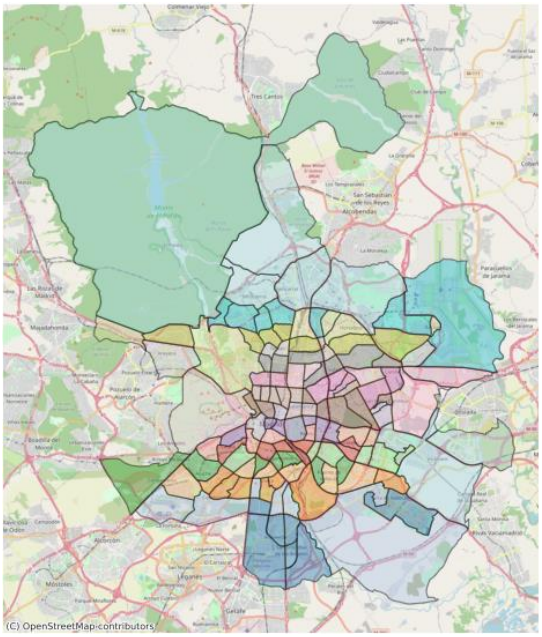

Ilustración 22. Delimitación de los barrios de Madrid.

3.3. Ingesta de datos

En la ingesta de los datos, el primer paso consiste en extraer la información desde las diferentes fuentes de origen y almacenarla dentro de la infraestructura, en un *bucket* de AWS S3. Para ello, es necesario tener en cuenta que existen dos tipos de fuentes de datos bien diferenciadas. Por un lado, tenemos una fuente tipo "*push*" (el servicio que envía la ubicación de los taxis disponibles), que requiere disponer de un sistema capaz de capturar y almacenar los datos cuando son recibidos. Por otro lado, existe otra fuente tipo "*pull*" (la base de datos con todos los viajes realizados en taxi), a la que es necesario conectarse para acceder a los datos y extraer la información necesaria.

Esta dualidad en los orígenes ("*push*" y "*pull*") se representa en el diagrama mostrado a continuación como dos flujos totalmente independientes que convergen en el mismo destino, un *bucket* en AWS S3 denominado "*Landing*". En este *bucket* se almacenarán todos los ficheros de estas fuentes sin procesar o con transformaciones mínimas, en formato CSV.

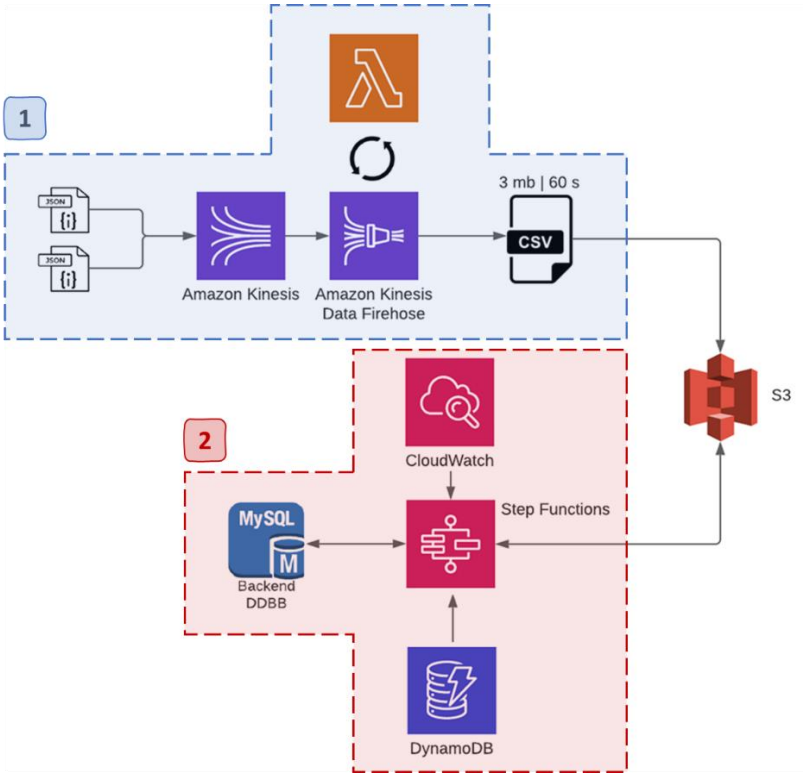

Ilustración 23. Diagrama de la arquitectura de ingesta desde el origen hasta "landing".

La primera parte del diagrama (resaltada en azul en la ilustración anterior, Ilustración 23), refleja la arquitectura desarrollada para capturar los datos de la ingesta tipo "*push*", es decir, aquella donde la información se envía desde el origen de datos hasta el propio sistema. En este caso se envían las ubicaciones de los taxis disponibles en Madrid en tiempo real, estos datos se envían en formato JSON.

Los datos llegan a una cola de AWS Kinesis Data Stream con 1 *shard* de capacidad, cada *shard* permite procesar hasta 1 MB por segundo o 1.000 registros por segundo y enviar hasta 2 MB por segundo. En esta cola de Kinesis la información se puede retener hasta un máximo de 24 horas. A continuación, los datos capturados en la cola son recogidos por AWS Kinesis Data Firehose, este servicio almacena en un búfer la información antes de entregarla en un *bucket* de AWS S3. La entrega se dispara cuando se cumple una de las siguientes condiciones definidas: se llega a un límite de 5 MB de datos o se alcanza

un tiempo de espera máximo de 5 minutos. Además, permite realizar ciertas transformaciones mediante funciones de AWS Lambda. En este caso, mediante una función Lambda desarrollada en Python se convierten los registros recibidos en formato JSON a formato CSV. Finalmente, los datos llegan al *bucket* de "*Landing*" en AWS S3.

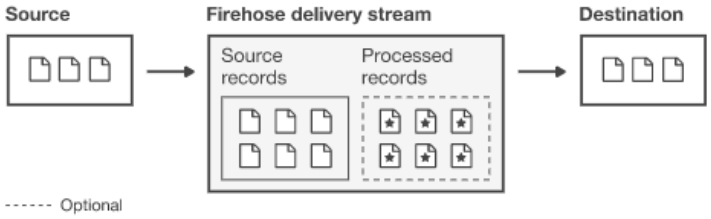

Ilustración 24. Funcionamiento de AWS Firehose [53].

Por otro lado, en la segunda parte del diagrama (resaltada en rojo, Ilustración 23), se muestra la arquitectura encargada de capturar los datos de la ingesta tipo "*pull*", es decir, aquella donde es necesario conectarse a la fuente para extraer los datos. En este caso, el origen es una base de datos relacional MySQL donde se almacena la información operativa sobre los viajes realizados en taxi, incluyendo información como el origen y el destino, el tiempo de espera o la duración del trayecto.

Este proceso de ingesta se activa periódicamente, programado mediante una regla en AWS Cloudwatch que inicia la ejecución cada hora de forma automática. El flujo de tareas se ha definido mediante AWS Step Functions, una máquina de estados que permite crear flujos de trabajo. Para definir esta máquina de estados se utiliza un lenguaje conocido como ASL (*Amazon States Language* o lenguaje de estados de Amazon), el cual está basado en JSON y tiene una estructura muy similar. Por ejemplo, mediante el siguiente fragmento de código (Ilustración 25), se define la tarea que ejecuta una función Lambda encargada de conectarse a la base de datos MySQL para extraer los datos (corresponde al punto 3 del diagrama mostrado en la Ilustración 26).

```
"Extract data from backend DB": {
  "Type": "Task",
  "Resource": "arn:aws:lambda:eu-west-1:function:data-ingestion-backend-pro:$LATEST",
  "ResultPath": "$",
  "Catch": [
    {
      "ResultPath": "$.error_log",
      "ErrorEquals": [
        "States.ALL"
      ],
      "Next": "Backend Data Extraction Execution Log"
    }
  ],
  "Next": "Backend Data Extraction Execution Log"
},
```

Ilustración 25. Definición de la tarea encargada de extraer datos de la base de datos.

Por último, cada una de las tareas del flujo de trabajo se ha planteado como una función AWS Lambda, desarrollada en Python. A continuación, se muestra un diagrama (Ilustración 26) con el flujo de trabajo definido en AWS Step Functions y compuesto por diferentes funciones Lambda.

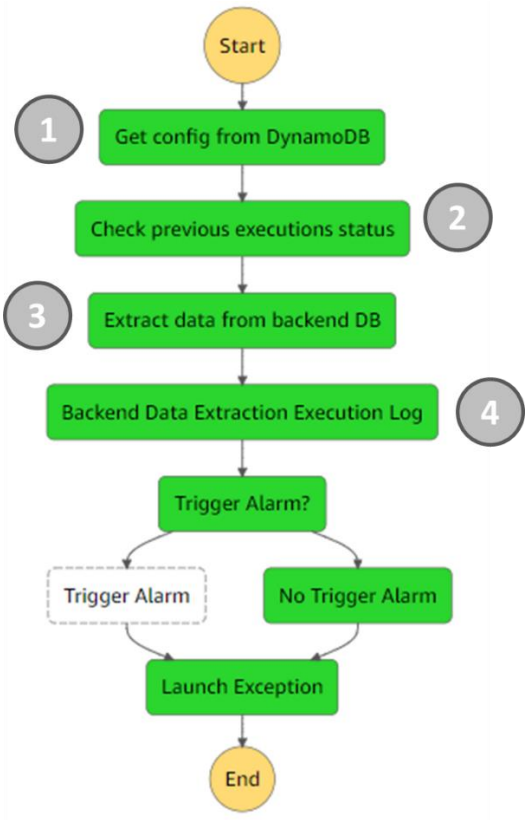

Ilustración 26. Flujo de procesos en ingestas "pull".

En primer lugar, (1) se extraen los parámetros de configuración de AWS DynamoDB una base de datos NoSQL totalmente administrada por AWS. Entre estos parámetros se encuentran detalles de la conexión a la base de datos, información acerca del *bucket* donde depositar la información, la ruta del fichero SQL con las consultas necesarias para extraer la información de la base de datos MySQL, etc. A continuación, (2) se comprueba el estado de la última ejecución, con el objetivo de determinar la última fecha de los datos extraídos (se trata de una ingesta incremental). La función Lambda en el proceso (3) se conecta a la base de datos MySQL mediante la librería "PyMySQL" y se ejecuta una consulta SQL para extraer los datos y almacenarlos en un *bucket* de AWS S3. Por último, (4) se comprueba el estado final de la ejecución, se actualizan los registros y en caso de que se haya producido algún problema se dispara una alerta.

3.4. Procesamiento de datos

Cuando los datos se han recogido desde las diferentes fuentes y han sido almacenados en un *bucket* S3 denominado como "*Landing*", se inicia el siguiente paso de la ingesta, el procesamiento de los datos. Los principales objetivos en esta parte del sistema son: armonizar los tipos de datos de los diferentes campos; transformarlos a formato parquet y actualizar el catálogo de datos. Con este propósito, se han definido tres etapas por las que pasa cada fichero de datos, representadas en la siguiente ilustración (Ilustración 27).

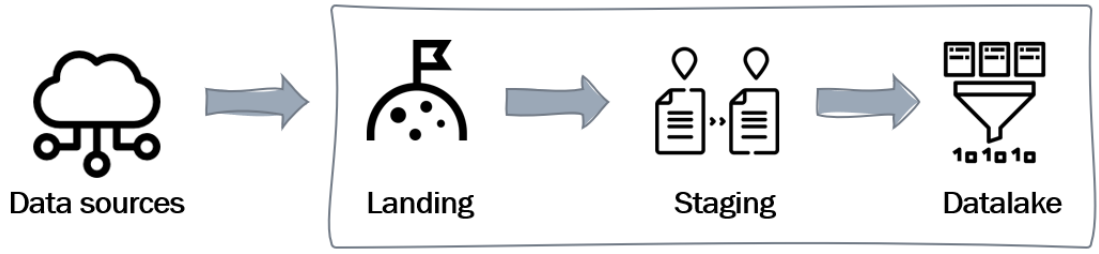

Ilustración 27. Flujo de ingesta, etapas que recorren los datos.

Cada una de estas etapas se desarrolla en un *bucket* de AWS S3 diferente. Inicialmente, los datos se encuentran en el *bucket* de "*Landing*", en este *bucket* los datos permanecen inalterados en su forma original. Desde el *bucket* de "*Landing*" se crea una copia en el *bucket* de "*Staging*", este es un almacenamiento temporal donde los datos son transformados antes de ser guardados definitivamente en el *bucket* denominado "*Datalake*", en formato parquet. Por último, los datos originales e inalterados de "*Landing*" son movidos a un *bucket* de "*Glacier*", donde se archivan indefinidamente en un almacenamiento en frío (esto es un almacenamiento de largo plazo y con un precio muy reducido, donde el coste tanto monetario como temporal se encuentra en la recuperación de los datos).

Al igual que ocurre en la ingesta de datos desde las fuentes de origen, el procesamiento de los datos se realiza mediante un flujo de trabajo definido en una máquina de estados mediante AWS Step Functions y compuesto por funciones AWS Lambda desarrolladas en lenguaje de programación Python. En el siguiente diagrama (Ilustración 28) se ilustran los diferentes pasos que recorren los datos desde su llegada a "*Landing*" hasta ser almacenados en formato parquet en el "*Datalake*", una vez procesados.

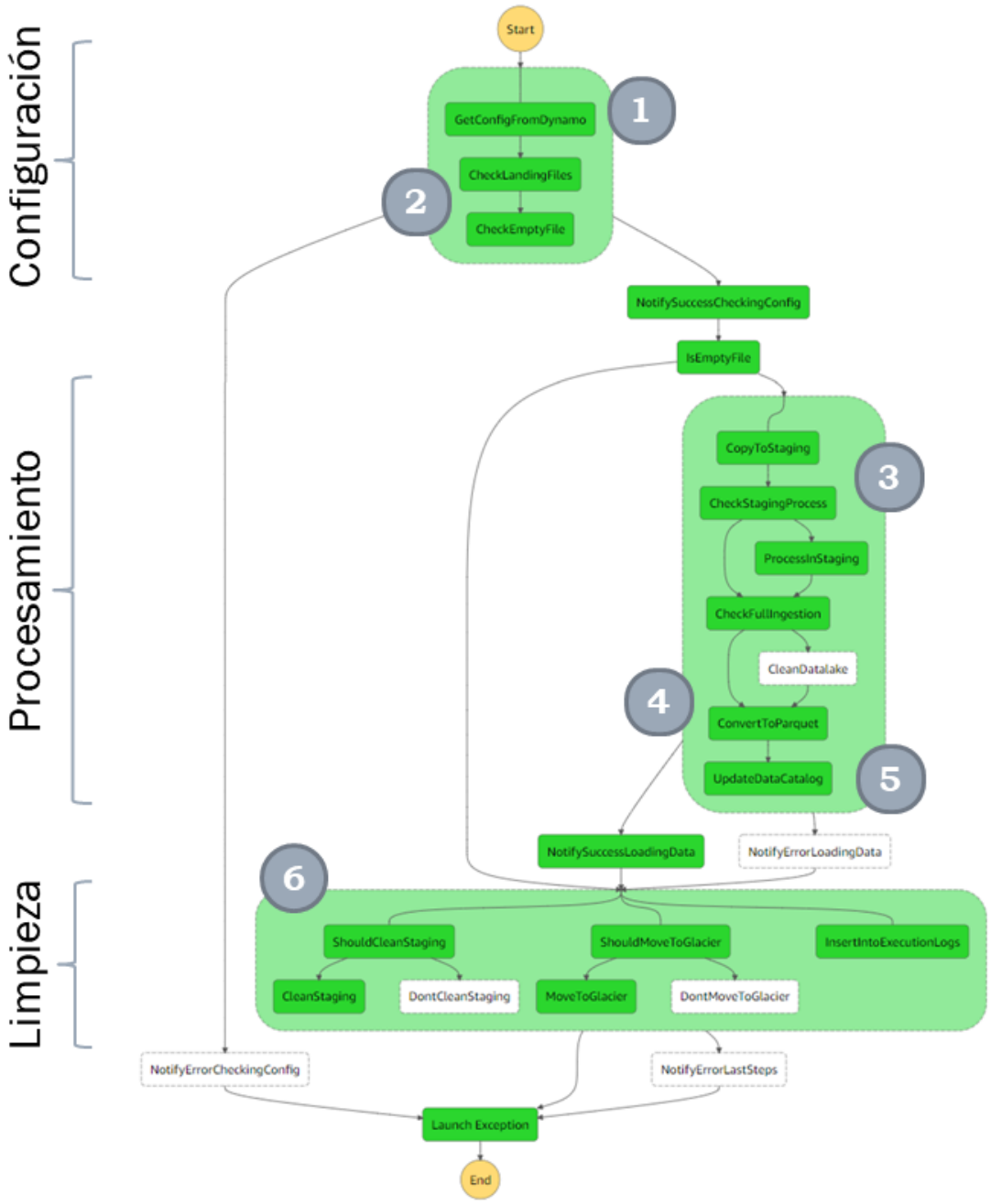

Ilustración 28. Flujo de ingesta desde landing hasta el datalake.

En primer lugar, (1) se extraen los parámetros de configuración de una base de datos NoSQL en AWS DynamoDB, incluyendo detalles de conexiones a base de datos, información sobre las rutas de los diferentes *buckets*, nombres de tablas en el catálogo de datos, mapa de los campos del dataset y el tipo de datos asociado a cada uno, detalles del separador usado en el fichero de origen, número de líneas de la cabecera, etc. En el siguiente paso, (2) se comprueba si el fichero está vacío o contiene datos, excluyendo las líneas de cabecera. En caso de que el fichero contenga datos, (3) se realiza una copia de

los ficheros de datos desde el *bucket* de "*Landing*" hasta el *bucket* de "*Staging*". En este *bucket* se lleva a cabo el procesamiento de los datos y la armonización de los tipos de para, a continuación, (4) convertirlos a formato parquet y almacenarlos en el *bucket* de "*Datalake*". En el siguiente paso, (5) se actualizan los metadatos en el catálogo de datos, incluyendo los nuevos datos procesados y la(s) partición(es) donde se han almacenado.

Por último, (6) se realizan las tareas de limpieza y archivado: se eliminan los datos temporales, en el *bucket* de "*Staging*"; se mueven los datos de "*Landing*" a "*Glacier*", a modo de *backup* y se comprueba el estado final de la ejecución, actualizando los registros y en caso de que se haya producido algún problema se dispara una alerta.

3.5. Tratamiento analítico de los datos

Una vez que los datos han sido extraídos desde las respectivas fuentes, procesados y están disponibles en formato parquet en AWS S3, comienza la parte más analítica del proyecto cuyo objetivo es tratar la información, analizar los datos, combinar diferentes fuentes de información o crear nuevas características que mejoren el rendimiento de los modelos.

El tratamiento de los datos se ha llevado a cabo en un clúster EMR con JupyterHub instalado y utilizando varios Jupyter Notebooks. Se ha empleado lenguaje de programación Python y se han utilizado algunas de sus librerías más extendidas en el procesamiento de datos como Numpy, Pandas o Matplotlib y otras menos frecuentes como H3 o GoogleMaps, todas ellas mencionadas en el apartado 2.5.

Dentro de este tratamiento de datos se han llevado a cabo diversas tareas que abarcan desde el cálculo de los límites geográficos con los que definir nuestra área de análisis, hasta el enriquecimiento de la información disponible mediante la creación de nuevas variables y la agrupación de los datos. Todas estas transformaciones están descritas en mayor profundidad en los siguientes apartados.

3.5.1. Cálculo de los límites geográficos y filtrado de datos

El primer paso en el tratamiento de datos ha consistido en identificar y definir el área geográfica sobre la que acotar el problema, concretamente centrada en el municipio de Madrid, puesto que es la zona donde opera principalmente la flota de taxis. Para ello, se ha utilizado el dataset de GADM, descrito en la sección 3.2.3. Esta fuente de información incluye los datos geográficos de todas las áreas administrativas de España.

Mediante la librería de Geopandas, se han cargado los datos en un *geodataframe* donde ha sido posible filtrar la información para extraer los polígonos geométricos que definen los límites geográficos de Madrid. En la siguiente ilustración (Ilustración 29) se ha representado esta área utilizando la librería Matplotlib.

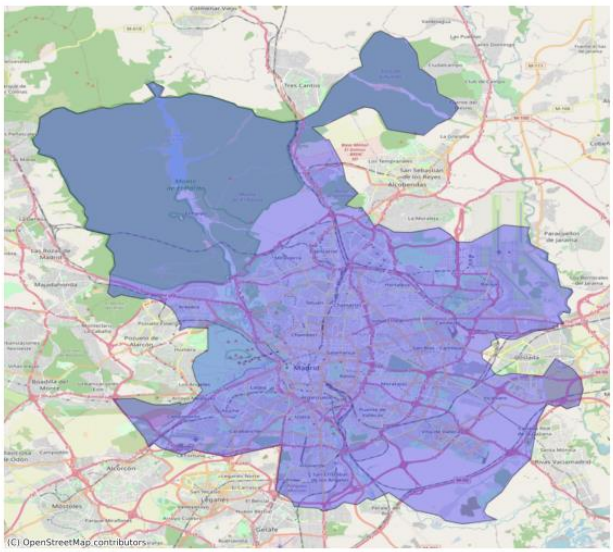

Ilustración 29. Representación geográfica del municipio de Madrid.

A partir de las coordenadas de latitud y longitud de los diferentes registros asociados a la disponibilidad de taxis (*dataset* descrito en la sección 3.2.1) se ha generado una figura geométrica (punto) utilizando la librería de Geopandas y se han identificado aquellos registros ubicados fuera del área del municipio de Madrid mediante un cruce espacial (aplicando la cláusula "*within*", "dentro de") con el *dataset* que contiene la definición del municipio. En el caso del *dataset* con las ubicaciones de los taxis disponibles en Madrid, tras filtrar los registros de fuera del municipio, se han conservado 227.294 registros de los 276.380 originales, es decir, se han descartado 49.086 registros, un 18% del total.

3.5.2. División en áreas geográficas

A partir de los datos geométricos que definen los límites del municipio de Madrid, se ha utilizado la librería H3 de Uber, para dividir la zona asociada al municipio en un conjunto de áreas hexagonales más reducidas. Partiendo del polígono que define Madrid, se ha transformado en un diccionario GeoJSON y se ha extraído la geometría definida para calcular las áreas hexagonales que lo componen mediante la función "*polyfill_geojson*" de H3. Para ello, se ha configurado una resolución de 7, es decir, cada hexágono tendrá un área de aproximadamente 5 km^2.

En la siguiente ilustración (Ilustración 30), se incluye el código fuente de una función en Python que itera a través de los diferentes polígonos que definen el área geográfica de Madrid y calcula cada uno de los hexágonos que la componen. Cada área hexagonal se diferencia mediante un identificador (*h3_hex*), además, se han calculado sus límites geográficos (un polígono, denominado como *h3_geo_boundary*), mediante la función de H3, "*h3_to_geo_boundary*", y se ha definido el centroide (*h3_centroid*), con sus coordenadas de latitud y longitud, mediante la función "*h3_to_geo*". Todas las áreas hexagonales creadas se han ido añadiendo a un nuevo *dataframe*.

```
# Create an empty dataframe to write data into
madrid_muni_h3_df = pd.DataFrame([],columns=['country','autonomous_community',
    'province','municipality','type','h3_id','h3_geo_boundary','h3_centroid'])

# Iterate over every row of the geo dataframe
for row in madrid_muni_gdf.itertuples():
    # Parse out info from columns of row
    country = row.country
    autonomous_community = row.autonomous_community
    province = row.province
    municipality = row.municipality
    type = row.type
    multipolygon = row.geometry
    # Convert multi-polygon into list of polygons
    polygon_list = list(multipolygon)
    for polygon in polygon_list:
        # Convert Polygon to GeoJSON dictionary
        poly_geojson = gpd.GeoSeries([polygon]).__geo_interface__
        # Parse out geometry key from GeoJSON dictionary
        poly_geojson = poly_geojson['features'][0]['geometry']
        # Fill the dictionary with Resolution 7 H3 Hexagons
        h3_hexes = h3.polyfill_geojson(poly_geojson, 7)
        for h3_hex in h3_hexes:
            h3_geo_boundary = shapely.geometry.Polygon(
                h3.h3_to_geo_boundary(h3_hex,geo_json=True)
            )
            h3_centroid = h3.h3_to_geo(h3_hex)
            # Append results to dataframe
            madrid_muni_h3_df.loc[len(madrid_muni_h3_df)]=[
                country,
                autonomous_community,
                province,
                municipality,
                type,
                h3_hex,
                h3_geo_boundary,
                h3_centroid
            ]
```

Ilustración 30. Creación de áreas hexagonales en el municipio de Madrid.

En la siguiente ilustración (Ilustración 31), se puede observar cómo se han generado los 116 hexágonos que cubren el municipio de Madrid al completo, con un área de aproximadamente 5 km^2 cada uno de ellos.

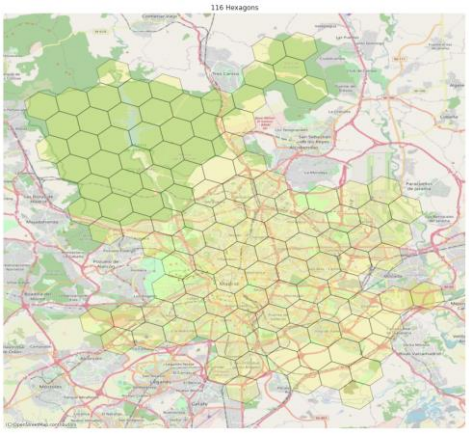

Ilustración 31. Municipio de Madrid dividido en áreas hexagonales.

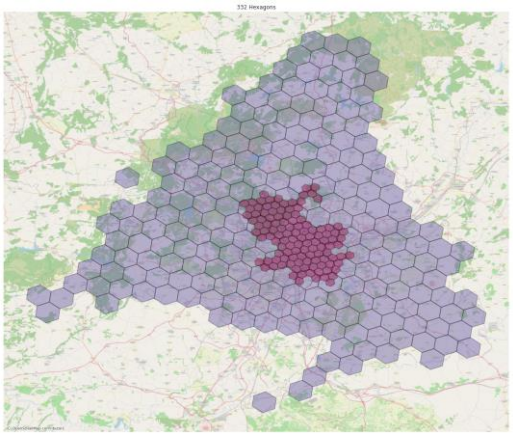

Ilustración 32. Comunidad de Madrid dividida en áreas hexagonales.

En un futuro, se podrían expandir las operaciones del taxi a una zona mayor de influencia que abarque más municipios o incluso la comunidad de Madrid al completo. En ese caso, puesto que la flota de taxis debe cubrir una mayor cantidad de terreno, el volumen de taxis disponibles por área podría ser muy reducido y por tanto el volumen de los datos sería insuficiente. En ese caso, se debería ampliar la estrategia definida anteriormente, utilizando hexágonos de menor precisión, es decir, que abarquen un área con mayor número de kilómetros cuadrados, para así garantizar que se captura un volumen de datos adecuado para poder entrenar los modelos. Incluso se podrían combinar los datos con la información del censo de los municipios de la Comunidad de Madrid para definir el tamaño de las áreas en función de la concentración de población en el municipio. En la siguiente ilustración (Ilustración 32) se representa la forma en que se podría dividir la Comunidad de Madrid combinando áreas de más precisión en la capital y áreas más amplias en el resto de la comunidad.

3.5.3. Enriquecimiento de datos con nuevas variables

El siguiente paso en el tratamiento de los datos ha consistido en el enriquecimiento de la información asociada a la disponibilidad de taxis en el área del municipio de Madrid.

Esta fuente de datos cuenta con información relativa a: la fecha del registro, es decir, el momento concreto en que el taxi estaba disponible para realizar una reserva; la ubicación del taxi en ese momento (latitud y longitud) y un identificar único del taxi asociado. A partir de estos datos se han construido las siguientes variables temporales:

- **Día de la semana**: Monday (0), Tuesday (1), Wednesday (2), Thursday (3), Friday (4), Saturday (5) y Sunday (6).
- **Indicador de fin de semana**: *False* (día laboral), *True* (fin de semana).
- **Hora del día**: 00 – 24.
- **Periodo del día**: Asociado a las horas: *Night* (00:00 – 05:59), *Morning* (06:00 – 11:59), *Afternoon* (12:00 – 17:59) y *Evening* (18:00 – 23:59).
- **Parte de la hora**: Asociado a los minutos: *OClock* (00 - 14), *QuarterPast* (15 - 29), *HalfPast* (30 - 44) y *QuarterTo* (45 - 59).

Por otro lado, se ha identificado el área hexagonal donde se ubican cada uno de los registros, mediante la función "geo_to_h3" y se han calculado tanto los límites del área geográfica como su centroide.

3.6. Desarrollo de modelos predictivos

En esta sección se detallará la forma en que se han desarrollado los dos modelos predictivos: El primero de ellos, es el encargado de identificar la probabilidad de que en un área y momento determinados exista algún taxi disponible. El segundo, debe ser capaz de estimar el tiempo de viaje entre dos puntos, origen y destino, en un momento concreto. En las siguientes páginas se analizan ambos, profundizando en la forma en que se han construido, los datos que se han utilizado y la validación que se ha llevado a cabo.

3.6.1. Modelo de predicción de disponibilidad

En este primer caso, sabiendo la localización del usuario y el momento temporal es necesario predecir el área desde la que se enviará el taxi que recoja al usuario. Como se ha mencionado en apartados anteriores, el tiempo de respuesta debe ser mínimo (ms), por lo que no es posible contactar con el sistema de taxis para localizar el taxi disponible más cercano al usuario e identificar su ubicación real. Por lo que es necesario desarrollar un modelo predictivo que dé respuesta a esta problemática.

Para ello, se utilizará el dataset que contiene la información histórica de todos los taxis disponibles, que incluye datos como la fecha, las coordenadas del taxi y el área hexagonal donde se ubica. Como ya se comentó anteriormente, tras filtrar las ubicaciones fuera del municipio de Madrid (18% del total), se cuenta con un dataset de 227.294 registros.

Partiendo de esta información es necesario preparar los datos de entrada al modelo. En primer lugar, es necesario hacer una preselección de variables y agregar la información a este nivel. En este caso, serán las siguientes:

1. **Identificador del área**: Es un identificador hexadecimal, único para cada una de las 116 áreas definidas.
2. **Día de la semana:** 0 (lunes), 1 (martes), 2 (miércoles), 3 (jueves), 4 (viernes), 5 (sábado) y 6 (domingo).
3. **Nombre del día de la semana:** *Monday* – *Sunday* (lunes – domingo).
4. **Indicador de fin de semana:** *False* (día laboral), *True* (fin de semana).
5. **Hora del día:** 00 – 23.
6. **Periodo del día:** Asociado a las horas: *Night* (00:00 – 05:59), *Morning* (06:00 – 11:59), *Afternoon* (12:00 – 17:59) y *Evening* (18:00 – 23:59).
7. **Parte de la hora:** Asociado a los minutos: *OClock* (00 - 14), *QuarterPast* (15 - 29), *HalfPast* (30 - 44) y *QuarterTo* (45 - 59).

Por otro lado, es necesario definir la variable objetivo, es decir, aquella que se quiere predecir. En este caso, esta variable se calculará como la media de taxis disponibles en un área y momento concretos.

Por último, el dataset cuenta con información acerca de las áreas y momentos temporales donde había algún taxi disponible, sin embargo, se desconoce la información cuando no ha habido ningún taxi disponible en esa zona o momento temporal, ya que al no haber ningún taxi disponible no se ha generado ningún registro asociado. Por ello, es necesario construir un dataset completo, es decir, incluyendo todas las posibles combinaciones de variables y calculando la tasa media de taxis en todos los casos, siendo cero en los casos donde no exista ningún taxi disponible. Para construir este dataset, se deben combinar todos los posibles valores en cuanto a áreas hexagonales (116), días de la semana (7), horas del día (24) y minutos, agrupados en cuartos de hora (4). Como resultado, tenemos un conjunto de datos que cuenta con un total de 77.952 de registros.

A continuación, partiendo de este conjunto de datos donde se han seleccionado las *features* que entrarán en el modelo y la variable objetivo, se han transformado las variables categóricas y se han divido los registros en dos conjuntos: una muestra de entrenamiento (*training*) y otra de prueba (*test*). Representan un 70% en el caso del conjunto de entrenamiento y un 30% en el caso del conjunto de prueba. Tras esto, se han

definido los diferentes parámetros del modelo. Concretamente se han realizado diferentes pruebas ajustado los valores de los siguientes parámetros [54]:

- **n_estimators**: Número de árboles incluidos en el modelo. Por defecto 100. En general, aumentar el número de árboles mejora el modelo sin producir *overfitting*, sin embargo, tras superar cierto umbral la mejora es mínima y simplemente implica un coste computacional.
- **max_depth**: Profundidad máxima que pueden alcanzar los árboles. Por defecto "None", es decir, ilimitado.
- **min_samples_split**: Número mínimo de observaciones que debe tener un nodo para poder dividirse. Por defecto, 2.
- **min_samples_leaf**: Número mínimo de observaciones que debe tener cada uno de los nodos hijo para que se produzca la división. Por defecto, 1.
- **max_features**: Número máximo de predictores considerados en cada división. Por defecto, "auto", es decir, se utilizan todos los predictores.
- **max_leaf_nodes**: Número máximo de nodos terminales que pueden tener los árboles. Por defecto, "None".
- **n_jobs**: Número de núcleos utilizados en el entrenamiento. El algoritmo de *random forest* permite entrenar los árboles en paralelo, ya que se ajustan de forma independiente. Con un valor "-1" se utilizan todos los núcleos disponibles.
- **random_state**: Semilla utilizada. Permite que los resultados sean reproducibles.

Ajustar algunos de estos parámetros no es una tarea a priori sencilla, por lo que una posible solución para encontrar los valores óptimos consistiría en desarrollar una función capaz de ir iterando a través de diferentes combinaciones de dichos parámetros. Por ejemplo, en el caso del número de estimadores, en la siguiente gráfica (Ilustración 33) podemos ver el punto donde se alcanza el valor óptimo (sin tener en cuenta modificaciones en otros parámetros).

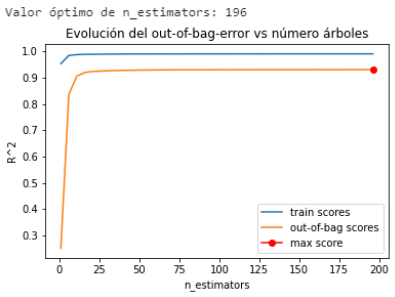

Ilustración 33. Evolución del out-of-bag-error en función del número de estimadores.

La librería de Sklearn, ofrece una funcionalidad para realizar este tipo de *tuning* u optimización del modelo. Mediante la función "*Grid Search*" es posible definir una búsqueda exhaustiva sobre todas las combinaciones en un conjunto de valores previamente fijados por el usuario. En este caso se han establecido los siguientes:

- **n_estimators**: 100, 250, 500.
- **max_features**: auto, sqrt, log2.
- **max_depth**: *None*, 2, 3, 4, 5.
- **min_samples_split**: 2, 500, 1000.

Este proceso iterará sobre todas las posibles combinaciones, por lo que cuanto mayor sea el número de parámetros diferentes, más aumentará el tiempo de computación necesario. En este caso tenemos un total de 135 posibles combinaciones, es decir, se entrenarán 135 modelos diferentes con estos parámetros hasta obtener la configuración óptima. En la siguiente tabla (Tabla 4), se muestran las cinco combinaciones que han obtenido mejores resultados:

Tabla 4. Resultados del modelo Random Forest aplicando diferentes parámetros.

	oob_r2	max_depth	max_features	min_samples_split	n_estimators
1	0.930408	NaN	auto	2	250
0	0.930007	NaN	auto	2	100
4	0.775136	NaN	sqrt	2	100
8	0.775136	NaN	log2	2	100
9	0.770696	NaN	log2	2	250

Tras revisar los resultados obtenidos, se ha entrenado el modelo aplicando la configuración óptima en base al análisis realizado: utilizando un total de 250 estimadores, sin limitar ni el número de *features* ni la profundidad del árbol y estableciendo un mínimo de dos observaciones para que un nodo pueda dividirse. Como resultado de entrenar el modelo aplicando el algoritmo de *random forest* con estos parámetros, se han identificado los *features* o características, que más influyen en el resultado. En la siguiente figura (Ilustración 34) se muestran ordenadas por importancia.

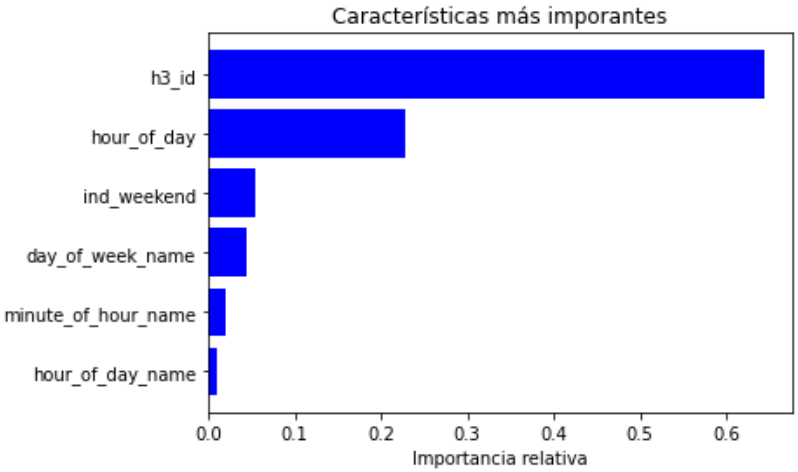

Ilustración 34. Características más importantes, Random Forest.

A continuación, se ha creado una matriz con todas las posibles combinaciones, es decir, áreas hexagonales (116), días de la semana (7, se incluyen dentro de este grupo los fines de semana), horas del día (24, se incluyen dentro de este grupo los periodos del día) y minutos, agrupados en cuartos de hora (4). Como resultado, tenemos un conjunto de datos que cuenta con un total de 77.952 de registros. Sobre este *dataset* se aplicará el modelo desarrollado para predecir la tasa de disponibilidad de taxis en cada uno de los registros.

Por último, se ha construido la matriz final, donde para cada área de destino y en cada uno de los momentos temporales, se ha identificado el área de origen del taxi. En este proceso, se han combinado dos factores: (1) la tasa de disponibilidad de cada área en un momento dado (calculada en la matriz anterior aplicando el modelo) y (2) el número de celdas que separan ambas áreas. Para ello, se ha diseñado un sistema de puntuaciones donde se han valorado por separado ambos factores:

Por un lado, la distancia hasta el área de recogida desde cada una de las demás áreas:

$$Score\_dist = 100\ x\ factor\_distancia/(distancia\_relativa + 1)$$

Donde *factor_distancia* es el peso de la componente de distancia (50%) y *distancia_relativa* es el número de hexágonos intermedios entre las dos áreas.

Por otro lado, la tasa de disponibilidad del área (calculada aplicando el modelo):

$$Score\_disp = 100\ x\ factor\_disponibilidad\ x\ categoria\_disponibilidad$$

Donde *factor_disponibilidad* es el peso de la componente de disponibilidad (50%) y *categoria_disponibilidad* es un entero (0 - 100), definido en función de la tasa de disponibilidad, de la siguiente forma:

- Tasa de disponibilidad > 0.75, categoría_disponibilidad = 100
- Tasa de disponibilidad > 0.60, categoría_disponibilidad = 50
- Tasa de disponibilidad > 0.50, categoría_disponibilidad = 25
- Tasa de disponibilidad > 0.25, categoría_disponibilidad = 10
- Tasa de disponibilidad <= 0.25, estos elementos han sido descartados.

Por último, se han sumado ambas puntaciones para seleccionar, para cada área destino (es decir, el punto de recogida del usuario) cual es el área de origen con mayor puntuación. Como se puede observar en esta clasificación, en ambos casos la ponderación es muy superior en la parte alta, es decir, cuando el hexágono está muy cerca y/o cuando tiene una alta tasa de taxis disponibles.

3.6.2. Modelo de estimación de tiempos

El segundo de los problemas consiste en que partiendo del área de destino (la ubicación del usuario) y habiendo calculado el área de origen (ubicación del taxi que prestará el servicio) tras aplicar el modelo del apartado anterior (3.6.1), es necesario calcular el tiempo estimado de viaje entre ambas áreas.

La primera aproximación para resolver este problema podría ser conectar con un servicio externo, como Google Maps, para obtener las estimaciones en tiempo real en el momento de la petición, sin embargo, no es posible utilizar este tipo de servicios externos puesto que como se ha comentado anteriormente, la exigencia de dar respuesta en un periodo de tiempo mínimo (ms) no permite aplicar este tipo de enfoque con garantías. Por lo tanto, para poder solucionar este reto, es necesario elaborar un modelo que nos permita realizar estimaciones fiables de los tiempos de viaje.

La primera barrera que ha surgido al plantear el desarrollo de este modelo ha sido encontrar un conjunto de datos con información de trayectos realizados entre puntos de Madrid, puesto que la fuente de datos disponible (servicio de taxis) no cuenta con un volumen de datos de viajes lo suficientemente amplio como para poder entrenar un modelo. Durante estos tres meses de histórico se han recogido datos de 159 viajes.

Por ello, es necesario localizar otra fuente externa que proporcione un volumen de datos aceptable. En otras ciudades del mundo, existen diferentes fuentes de datos abiertos de movilidad, por ejemplo, una de las más famosas es el *dataset* de viajes en taxi en la ciudad de Nueva York, que contiene información de millones de trayectos. Es publicado por la "*NYC Taxi and Limousine Commission (TLC)*" y se actualiza periódicamente [56]. Sin

embargo, no ha sido posible encontrar este tipo de datos abiertos para el área acotada en nuestro problema, el municipio de Madrid.

Por estos motivos, ha sido necesario construir una fuente de datos propia, utilizando servicios externos, como las APIs de Google Maps, para generar estos datos. Google Maps cuenta con diferentes APIs que permiten calcular el tiempo de trayecto entre varios puntos de manera precisa. Gracias a los millones de usuarios que utilizan sus servicios de movilidad, cuenta con información histórica de millones de trayectos y dispone de la situación del tráfico en tiempo real o del tráfico histórico con el que estimar el futuro.

El funcionamiento de estas APIs se ha descrito anteriormente en la sección 2.5.4. Como se detalla en dicha sección, debido al coste por llamada y al presupuesto limitado, es posible construir un dataset con un máximo de 50.000 registros de trayectos. Debido a esta restricción, se ha elaborado una estrategia para optimizar al máximo estas peticiones.

Para ello, se ha elaborado una clasificación de áreas en función de la disponibilidad media de cada zona: áreas "*hot*", "*mild*", "*cold*" y "*empty*". En la sección 4.2.1 se profundiza en la metodología utilizada para establecer esta agrupación de áreas. A partir de esta clasificación se han distribuido las peticiones a la API de Google Maps de forma que las áreas de mayor disponibilidad cuenten con más peticiones, para garantizar precisiones más fiables. Por otro lado, las áreas "*empty*" no serán incluidas como origen en ninguna petición ya que en ningún momento hubo taxis disponibles en estas zonas. En la siguiente tabla (Tabla 5) se resume la forma en que se han distribuido estas peticiones:

Tabla 5. Distribución de las peticiones a la API de Google Maps.

Origen	#Áreas	Destino	#Áreas	Repeticiones	Comentarios	Total
Cold	34	***Empty***	30	**1**	Petición única por área (1)	**1.020**
Cold	34	***Cold***	34	**1**	Petición única por área (1)	**1.156**
Mild	33	***Empty***	30	**1**	Petición única por área (1)	**990**
Mild	33	***Cold***	34	**7**	Una petición por cada día de la semana (7) y por área	**7.854**
Mild	33	***Mild***	33	**7**	Una petición por cada día de la semana (7) y por área	**7.623**
Hot	19	***Empty***	30	**1**	Petición única por área (1)	**570**
Hot	19	***Cold***	34	**7**	Una petición por cada día de la semana (7) y por área	**4.522**
Hot	19	***Mild***	30	**28**	Una petición por día de la semana (7), periodo del día (4) y área	**15.960**
Hot	19	***Hot***	19	**28**	Una petición por día de la semana (7), periodo del día (4) y área	**10.108**
TOTAL						**49.803**

Como se puede apreciar en la tabla anterior (Tabla 5), los trayectos desde áreas “hot” a otras áreas “hot” o “mild” representan el 50% de las peticiones realizadas, debido a que la mayoría de los taxis disponibles se concentrará en estas zonas. Respecto a las áreas “mild” o trayectos entre áreas “hot” y “cold”, se encuentran moderadamente representadas con un 40% de las peticiones. Por último, en las áreas “cold”, aquellas con una disponibilidad de taxis mínima, apenas contarán con un 10% de las peticiones puesto que serán los casos menos frecuentes.

Puesto que el volumen de datos disponible es reducido, el objetivo de esta estrategia es minimizar la tasa de error en los casos más frecuentes, priorizando las áreas de origen con mayor tasa de disponibilidad, y, por tanto, reduciendo el impacto negativo en la calidad del servicio (derivado de imprecisiones en las estimaciones) al mínimo aceptable. En futuras iteraciones, una de las principales prioridades debería ser incrementar el volumen de información disponible en este conjunto de datos.

Por otro lado, debido al reducido número de peticiones disponibles se ha decidido tomar como puntos de origen y destino los centroides de cada una de las áreas en vez de seleccionar puntos aleatorios dentro de estas. Por ello, otro de los factores clave para mejorar el rendimiento de este modelo en futuras iteraciones, será ampliar la variedad de puntos seleccionados, tomando ubicaciones aleatorias. En la siguiente imagen (Ilustración 35), se puede apreciar gráficamente la forma en que se distribuyen los puntos de origen y los de destino, en este ejemplo, viajes desde áreas “hot” a áreas “mild”.

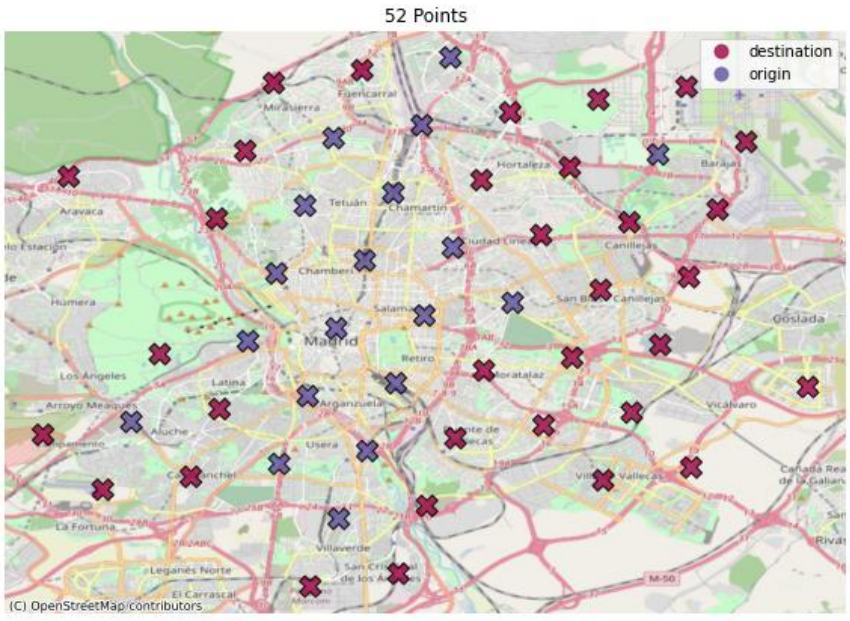

Ilustración 35. Trayectos desde áreas "hot" a áreas "mild".

Por último, para garantizar una muestra diversa de fechas y horas en los viajes, se ha decidido utilizar "*Directions API*" (peticiones individuales) en lugar de "*Distance Matrix API*" (lista de orígenes y destinos). Puesto que la primera de ellas permite realizar peticiones individuales, indicando fechas y horas diferentes en cada petición. Las fechas han sido seleccionadas de forma aleatoria, dentro de unos rangos definidos en función del día de la semana o momento del día deseado.

Mediante una función desarrollada en Python, se ha automatizado la creación del *dataframe*. Esta función debe recibir como parámetros: una lista de orígenes, destinos y fechas de viaje; el número de repeticiones entre la misma combinación de pares origen-destino (por ejemplo, 7 si se desea repetir el análisis para cada día de la semana); si se desea incluir los datos de tráfico de Google y el tiempo de espera entre peticiones, para evitar ciertas restricciones de la API. La función itera entre todos los orígenes y destinos, tantas repeticiones como se indica, realizando peticiones a la API, procesando los resultados en formato JSON y añadiéndolos como registros en un *dataframe*.

En la siguiente figura (Ilustración 36), se incluye una muestra del código fuente desarrollado en Python para automatizar la captura de los datos a través de la API de Google y la creación del *dataframe* que incluirá la información disponible acerca de los trayectos entre estos puntos de origen y destino.

Por último, utilizando esta metodología se ha construido el conjunto de datos que incluye los campos mencionados en la sección 3.2.2. Además, se han creado una serie de campos adicionales con el objetivo de enriquecer la información disponible y mejorar el rendimiento del modelo desarrollado:

- Día de la semana del trayecto.
- Indicador de fin de semana.
- Periodo del día asociado al trayecto (mañana, tarde, noche, madrugada).
- Distancia en línea recta entre los centroides de ambas áreas.
- Número de áreas entre el origen y el destino.
- Indicador de si las áreas origen y destino son vecinos.

```python
def directions(origins_df, destinations_df, departure_time, include_traffic, repetitions, max_delay):

    # Create an empty dataframe to write data into
    estimations_df = pd.DataFrame([],columns=['departure_time', 'origin_id','origin_lat','origin_lon',
                                              'origin_address', 'destination_id','destination_lat',
                                              'destination_lon','destination_address', 'distance_meters',
                                              'duration_seconds','duration_in_traffic_seconds'])

    max_iterations = len(departure_time)
    i = -1
    dept_dt = None
    seed(1)

    for x in range(repetitions):

        for origin_row in origins_df.itertuples():

            # Find origin coordinates
            origin_centroid = origin_row.h3_centroid
            origin_id = origin_row.h3_id

            for destination_row in destinations_df.itertuples():

                # Iterate to the next element in departure times array
                i = i + 1
                rnd = random() * max_delay
                print('iteration', i+1, 'of', max_iterations, '... Sleep:', rnd, 'seconds')

                # Delay to avoid Google Maps API restrictions
                time.sleep(rnd)

                # Find the departure time
                dept_dt = departure_time[i]

                # Find destination coordinates
                destination_centroid = destination_row.h3_centroid
                destination_id = destination_row.h3_id

                # Check if origin and destination are in the same area
                if origin_id != destination_id:

                    if include_traffic:
                        # Use Google Maps API to calculate time estimations between 2 points,
                        # considering traffic conditions
                        result = gmaps.directions(
                                                  origin_centroid,
                                                  destination_centroid,
                                                  mode="driving",
                                                  departure_time=dept_dt,
                                                  traffic_model='best_guess',
                                                  units='metric')
                    else:
                        # Use Google Maps API to calculate time estimations between 2 points,
                        # considering traffic conditions
                        result = gmaps.directions(
                                                  origin_centroid,
                                                  destination_centroid,
                                                  mode="driving",
                                                  units='metric')

                    # Process the results
                    #departure_time
                    #origin_id
                    #origin_centroid
                    origin_lat = result[0].get('legs')[0].get('start_location').get('lat')
                    origin_lon = result[0].get('legs')[0].get('start_location').get('lng')
                    origin_address = result[0].get('legs')[0].get('start_address')
                    #destination_id
                    #destination_centroid
                    destination_lat = result[0].get('legs')[0].get('end_location').get('lat')
                    destination_lon = result[0].get('legs')[0].get('end_location').get('lng')
                    destination_address = result[0].get('legs')[0].get('end_address')
                    distance_meters = result[0].get('legs')[0].get('distance').get('value')
                    duration_seconds = result[0].get('legs')[0].get('duration').get('value')

                    if include_traffic:
                        duration_in_traffic_seconds = result[0].get('legs')[0].get('duration_in_traffic').get('value')
                    else:
                        duration_in_traffic_seconds = None

                    # Append results to dataframe
                    estimations_df.loc[len(estimations_df)]=[
                        dept_dt,
                        origin_id,
                        origin_lat,
                        origin_lon,
                        origin_address,
                        destination_id,
                        destination_lat,
                        destination_lon,
                        destination_address,
                        distance_meters,
                        duration_seconds,
                        duration_in_traffic_seconds
                    ]

    return estimations_df
```

Ilustración 36. Función para calcular tiempos entre varios orígenes y destinos.

A continuación, se ha llevado a cabo el desarrollo del modelo encargado de estimar los tiempos de viaje entre un origen (ubicación del taxi) y un destino (ubicación del usuario), aplicando un algoritmo de *gradient boosting*. Este método se basa en producir un conjunto de modelos predictivos más sencillos (débiles), generalmente árboles de decisión, y a partir de ellos construir un modelo más robusto. En la sección 2.3 se ha analizado en profundidad este tipo de algoritmos y en la sección 2.5.6 se han examinado en detalle la librería Python utilizada para implementar dicho algoritmo, XGBoost.

Partiendo del conjunto de datos creado mediante la API de Google Maps, en primer lugar, se ha definido la variable objetivo a predecir (el tiempo de duración del viaje, en segundos), se ha realizado una preselección de las variables que entrarán en el modelo y se han tratado estas variables aplicando los siguientes principios:

- Analizar y tratar los valores ausentes (*missings*).
- Excluir las variables con varianza próxima a cero. Puesto que las variables con un único valor (varianza cero) no aportan información al modelo.
- Estandarizar la escala de variables numéricas. Para evitar que aquellas con una mayor escala influyan en mayor medida en el modelo.
- Convertir en varias variables binarias las variables categóricas.

A continuación, se ha dividido el *dataset* en dos conjuntos de entrenamiento o *training* (70%) y prueba o *test* (30%). Finalmente se han definido los parámetros de configuración y se ha llevado a cabo el entrenamiento del modelo. XGBoost cuenta con conjunto muy numeroso de parámetros de configuración, divididos en tres categorías: parámetros generales, parámetros del *booster* y parámetros de aprendizaje [55]. Entre todos ellos, se han seleccionado algunos, debido a su importancia, para optimizar el modelo:

- **max_depth**: Profundidad máxima de los árboles. A mayor profundidad aumenta el coste computacional y puede derivar en problemas de sobreajuste u *overfitting*. Por defecto, 6.
- **learning_rate (eta)**: Tasa de aprendizaje. Reduce la contribución de cada árbol, multiplicando su influencia original por este valor. Este parámetro ayuda a evitar el sobreajuste u *overfitting*. Por defecto, 0.3.
- **subsample**: Proporción de observaciones utilizadas en el ajuste de cada árbol. Por defecto 1, es decir, se seleccionan todas las observaciones.

- **booster:** Indica el tipo de refuerzo a utilizar entre las diferentes posibilidades: *gbtree*, *gblinear* o *dart*. Los dos primeros utilizan modelos basados en árboles, por otro lado, *dart* utiliza modelos basados en funciones lineales. Por defecto, el valor es *gbtree*.

Finalmente, al igual que en el modelo de *random forest*, se ha llevado a cabo la optimización de los parámetros de entrada mediante una serie de pruebas combinando todos ellos y evaluando los resultados obtenidos hasta encontrar la combinación óptima. En la siguiente tabla (Tabla 6) se muestran los resultados para las diez combinaciones que han obtenido mejores resultados:

Tabla 6. Resultados del modelo XGBoost aplicando diferentes parámetros.

	param_booster	param_learning_rate	param_max_depth	param_subsample	mean_test_score	std_test_score	mean_train_score	std_train_score
17	gbtree	0.1	None	1	-91.973870	0.480129	-64.285822	0.706910
23	gbtree	0.1	10	1	-92.269562	0.424406	-31.437946	3.270415
16	gbtree	0.1	None	0.5	-93.461750	0.517370	-63.478279	1.365415
22	gbtree	0.1	10	0.5	-94.206244	0.255947	-36.447036	3.976339
20	gbtree	0.1	5	0.5	-99.641649	0.640304	-79.054711	0.533393
21	gbtree	0.1	5	1	-101.591654	0.932667	-83.940655	0.846982
14	gbtree	0.01	10	0.5	-110.629547	0.794638	-79.933263	0.079942
15	gbtree	0.01	10	1	-113.678738	0.977849	-80.853208	0.630393
8	gbtree	0.01	None	0.5	-178.714809	1.551560	-169.360388	0.132197
9	gbtree	0.01	None	1	-187.555597	2.718895	-178.196529	1.326409

Una vez seleccionada la combinación de parámetros óptima se ha entrenado el modelo final aplicando esta configuración. Como resultado de dicho modelo, en la siguiente figura (Ilustración 37) se han incluido las variables de mayor relevancia.

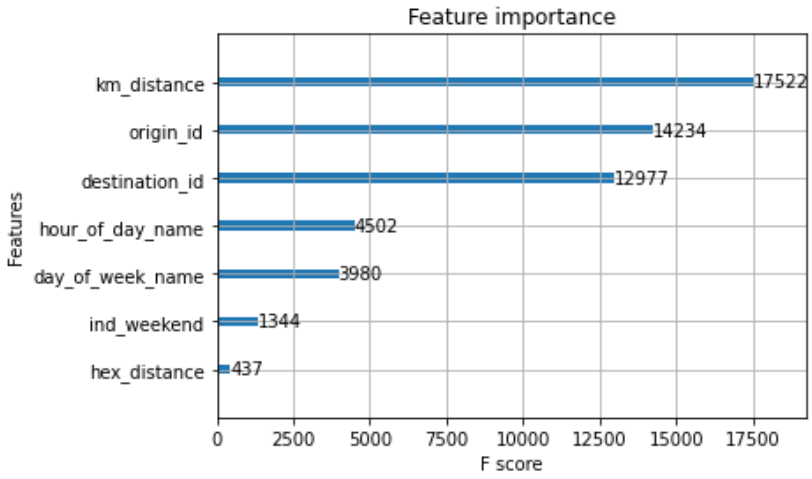

Ilustración 37. Características más importantes, XGBoost.

Además, se ha representado visualmente la dispersión de los valores predichos frente a los valores reales, utilizando la muestra del conjunto de prueba. En las siguientes

imágenes se puede observar la comparativa del modelo inicial (Ilustración 38) frente al resultado final tras realizar la optimización de los parámetros (Ilustración 39).

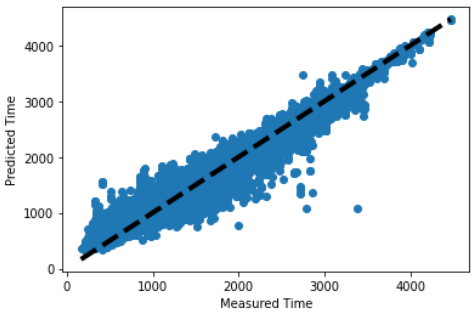

Ilustración 38. Modelo inicial XGBoost.

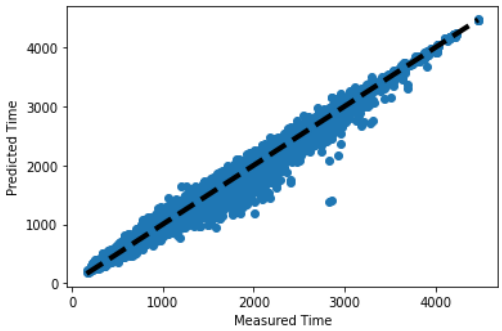

Ilustración 39. Modelo optimizado XGBoost.

.

Por último, se han calculado diferentes métricas para evaluar el modelo final [57]:

```
MSE: 7289.270547101851

RMSE: 85.377224990637

MAE: 57.3726597824806

R2: 0.9875648037079984
```

Ilustración 40. Métricas de evaluación calculadas para el modelo XGBoost.

RMSE: Raíz cuadrada del error cuadrático medio (**MSE**). Es la métrica utilizada más frecuentemente para evaluar modelos de regresión. Representa a la raíz cuadrada de la distancia cuadrada promedio entre el valor real y el valor pronosticado. Indica el ajuste absoluto del modelo frente a los datos, es decir, como de cerca están los valores observados frente a los predichos por el modelo. Cuanto más bajo sea el RMSE más ajustado estará el modelo. Se formula como:

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

MAE: Error absoluto medio o *Mean Absolute Error*. Es el promedio de diferencias absolutas entre los valores objetivo y predichos. Es una puntuación lineal, lo que significa que todas las diferencias individuales se ponderan por igual en el promedio, es decir, es menos sensible a valores atípicos y se considera más robusto que RMSE. Al igual que en RMSE, cuanto más bajo sea el valor más ajustado será el modelo. Se formula como:

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

$\boldsymbol{R^2}$: Coeficiente de determinación o *coefficient of determination*. Indica la bondad del modelo y muestra si las variables independientes seleccionadas explican la variabilidad en sus variables dependientes. Una de sus grandes ventajas es que su escala va de $-\infty$ a 1, donde un valor negativo indica que el modelo es peor que predecir la media, 0 indica que el modelo desarrollado no mejora la predicción sobre el modelo medio y 1 indica un error cercano a cero. Se formula como:

$$R^2 = 1 - \frac{\sum(y_j - \hat{y}_j)^2}{\sum(y_j - \bar{y}_j)^2}$$

3.6.3. Creación de una matriz combinada

Finalmente, una vez que se han seleccionado los modelos definitivos mediante un análisis detallado aplicando diferentes métricas de evaluación sobre los resultados e iterando hasta obtener los parámetros óptimos, se creará una matriz que combinará ambos modelos para dar respuesta a cualquier petición. Es decir, se construirá una matriz con 77.952 registros a partir de todas las posibles combinaciones de:

- Áreas de destino, es decir, la ubicación de recogida del usuario (116).
- Fecha del viaje, día de la semana (7, incluido si es fin de semana o no).
- Fecha del viaje, hora del día (24, incluye los 4 periodos del día).
- Fecha del viaje, parte de la hora (4).

A partir de estos datos, se calculará el área de origen, es decir, la ubicación desde donde vendrá el taxi asignado, aplicando el primero de los modelos. Y, a partir de esta nueva información se estimará el tiempo de recogida aplicando el segundo de los modelos.

Finalmente, todos los valores deben estar precalculados para ahorrar el tiempo de procesamiento necesario cuando se reciba una petición y reducir el tiempo de respuesta al mínimo. Esta matriz será almacenada en una base de datos NoSQL donde la clave será la combinación de área de destino, día de la semana, hora del día y minutos de la hora (en cuartos de hora) y el valor será la estimación del tiempo de espera.

4. Integración y resultados

En este capítulo se detallará la forma de integrar la solución desarrollada en un entorno real, con exigentes requisitos en cuanto a tiempo de respuesta se refiere. Por otro lado, se revisarán los resultados obtenidos a partir de evaluar el sistema desarrollado frente a datos reales recogidos de viajes en taxi solicitados a través de una aplicación móvil.

4.1. Integración con la App

Cuando un usuario utiliza una aplicación móvil, uno de los principales problemas para garantizar una experiencia de usuario óptima es reducir al máximo los tiempos de carga de las diferentes pantallas. Para lograr este objetivo, es necesario que todos los componentes estén listos en el menor tiempo posible.

En este caso, el tiempo de espera estimado hasta la recogida de un taxi, es uno de estos elementos. Para estimar este tiempo, en primer lugar, se debe determinar la posición del usuario y la fecha actual y enviar una solicitud al servicio. En los capítulos anteriores, se ha descrito la forma en que se ha creado una matriz que incluye todas las posibles combinaciones de ubicaciones (áreas) y fechas junto con el tiempo de espera estimado para todas ellas. Por lo tanto, el único problema restante es definir una arquitectura capaz de devolver estas estimaciones en un tiempo de respuesta mínimo.

Tras estudiar las diferentes alternativas en cuanto a bases de datos tradicionales, como PostgreSQL u otras alternativas como Redis, una base de datos en memoria (caché), se ha definido una solución final a través de un modelo híbrido que combina una base de datos NoSQL con un caché en memoria. Esto proporciona un rendimiento teórico, en base a las especificaciones, de milisegundos (de un solo dígito) a microsegundos [58].

Se trata de una base de datos NoSQL, AWS DynamoDB, combinada con un servicio de caché DAX (Amazon DynamoDB Accelerator). Además, este servicio es totalmente administrado, lo que permite al usuario abstraerse de ciertas tareas como gestionar la invalidación del cache, encargarse del rellenado de datos o de la administración de clústeres [59]. En el siguiente diagrama (Ilustración 41), se describe el funcionamiento de esta arquitectura.

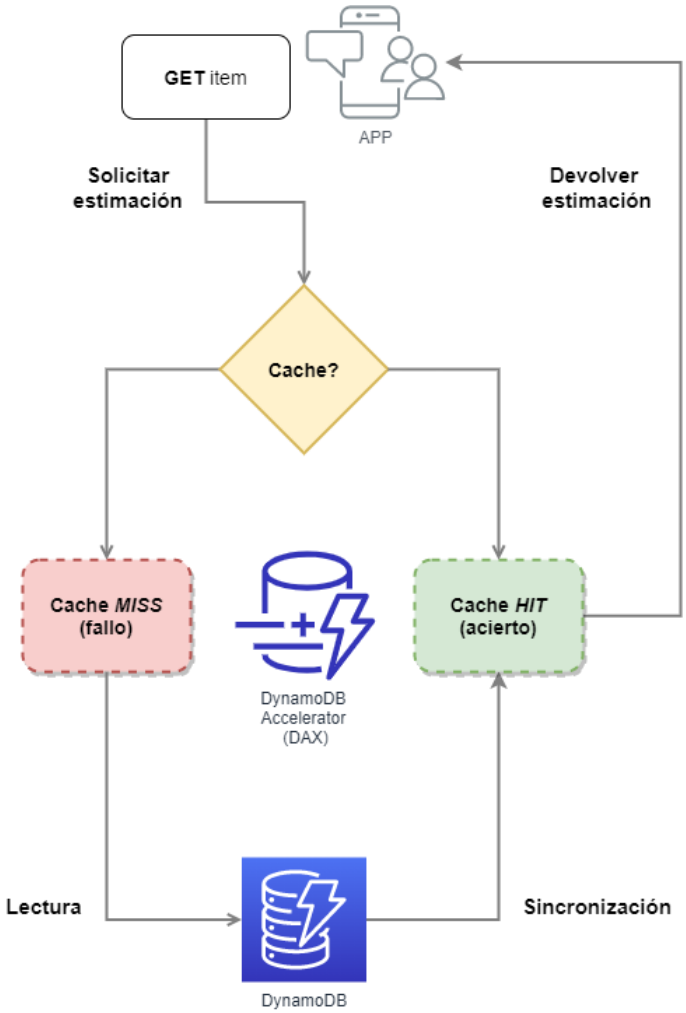

Ilustración 41. Gestión de solicitudes mediante AWS Dynamo y DAX.

Cuando la aplicación solicita una estimación (operación de lectura), se comprueba si dicho elemento se encuentra en la caché (DAX), en caso afirmativo se produce un acierto o *cache hit* y se devuelve el ítem (estimación) a la aplicación móvil. En caso contrario, el elemento no está presente en la caché (DAX), por lo que se busca en la base de datos (DynamoDB), se sincroniza la caché (para que este elemento esté disponible en la próxima lectura) y se devuelve la estimación a la aplicación móvil.

Los elementos se almacenarán en DynamoDB en un formato clave-valor, donde la clave es una combinación del área destino (punto de recogida) y las variables temporales (día de la semana, hora y minutos) y el valor es la estimación del tiempo de recogida.

Esta arquitectura tiene una gran ventaja y está totalmente alineada con el planteamiento del problema, ya que la forma en que se han clasificado los puntos de origen (por áreas)

y como se ha categorizado el tiempo en días de la semana y horas del día, hace que el uso de la memoria caché sea constante y por tanto el tiempo de respuesta sea mínimo.

4.2. Análisis exploratorio de los datos

A partir del conjunto de datos construido en las diferentes secciones del capítulo 3, se ha llevado a cabo un análisis exploratorio de la información disponible, tratando de descubrir las características de las diferentes áreas y el comportamiento de los taxis en diferentes momentos temporales.

4.2.1. Análisis por áreas geográficas

En este primer análisis, se ha aumentado el nivel de agregación de los datos hasta agrupar por cada una de las 116 áreas hexagonales. Como se puede apreciar en la siguiente imagen (Ilustración 42), la mayoría de los taxis disponibles se agrupan en las zonas interiores del municipio de Madrid, siendo especialmente intenso en la zona centro. Por otro lado, las áreas periféricas sufren una gran escasez de taxis disponibles. Se observan algunas particularidades como por ejemplo el área que incluye al aeropuerto de Madrid, donde la disponibilidad de taxis es superior al de áreas vecinas, incluidas aquellas que están más cerca del área interior.

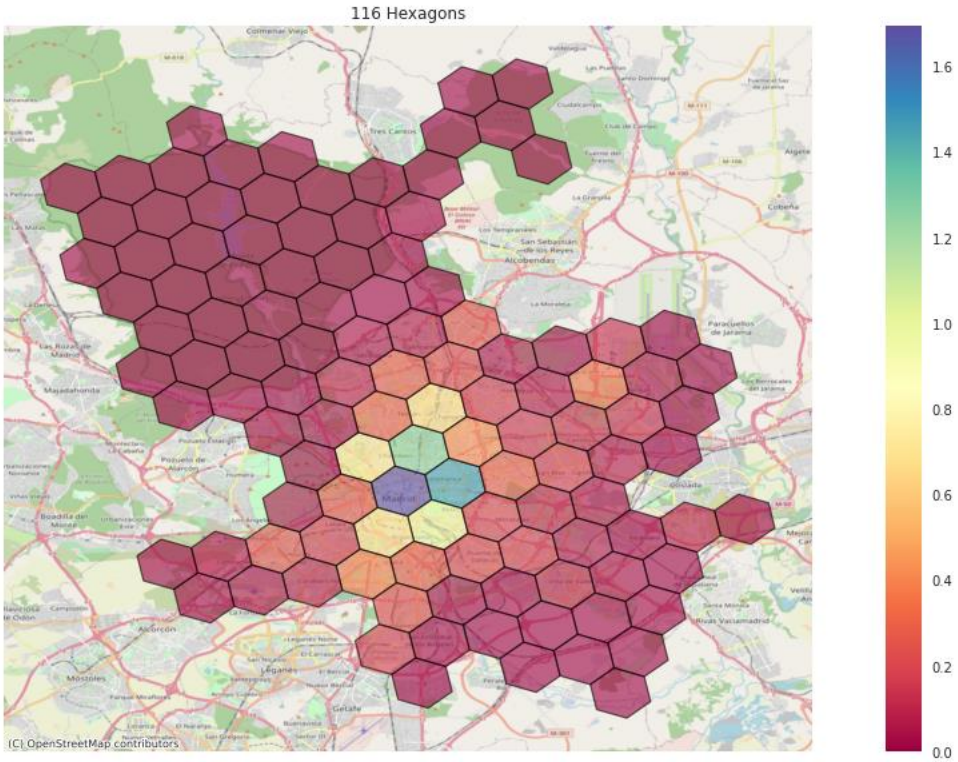

Ilustración 42. Disponibilidad de taxis en las áreas de Madrid.

A partir de este análisis, se han clasificado las áreas en cuatro grandes grupos en función de la disponibilidad media de taxis calculada para cada zona a partir de la información histórica. Estas áreas son: "*hot*", "*mild*", "*cold*" y "*empty*".

Las áreas "*hot*" son aquellas que presentan una mayor concentración media de taxis disponibles. Como se puede apreciar en la siguiente imagen (Ilustración 43) existen 19 áreas con estas características, las cuales se encuentran principalmente en el interior del municipio, lo que se conoce como "almendra central", es decir, dentro de los límites marcados por la circunvalación M-30. Además, destaca la presencia de un área bastante alejado del interior, se trata del aeropuerto de Madrid.

Ilustración 43. Disponibilidad de taxis en áreas "hot".

Por otro lado, las áreas "*mild*" son aquellas que cuentan con una concentración intermedia de taxis, muy inferior al de áreas "*hot*". Principalmente se trata de zonas que rodean a la circunvalación M-30 pero se encuentran en el interior de los límites de la M-40, mayoritariamente barrios residenciales. Además, en este grupo, se pueden observar áreas de nueva construcción que presentan ciertos problemas de movilidad debido a la carencia de transporte público (es decir, no existen estaciones de metro ni cercanías) como por ejemplo el barrio de "El Cañaveral", entre Coslada y Vicálvaro. También destacan áreas en las afueras, cercanas a estaciones de cercanías y centros empresariales. Como se puede apreciar en la siguiente imagen (Ilustración 44) dentro de este grupo se pueden distinguir 33 áreas hexagonales diferentes.

Ilustración 44. Disponibilidad de taxis en áreas "mild".

Respecto a las áreas "*cold*", son aquellas con una tasa media de disponibilidad de taxis muy reducida. Todas las áreas se localizan en las afueras del municipio, además, se trata de zonas con una densidad de población mínima. En este grupo se incluyen numerosos tramos de carretera, alejados de los núcleos de población y del centro del municipio. En la siguiente imagen (Ilustración 45) se pueden observar estas 34 áreas hexagonales y su distribución en el entorno del municipio de Madrid.

Ilustración 45. Disponibilidad de taxis en áreas "cold".

Por último, las áreas "*empty*", son aquellas donde no se ha registrado ningún taxi disponible en los datos capturados durante todo el periodo histórico. Estas áreas están principalmente ubicadas en zonas boscosas, principalmente el monte de "El Pardo", donde no hay núcleos de población ni acceso por carreteras convencionales. En este grupo se encuentran un total de 30 áreas diferentes que pueden observarse en la siguiente imagen (Ilustración 46).

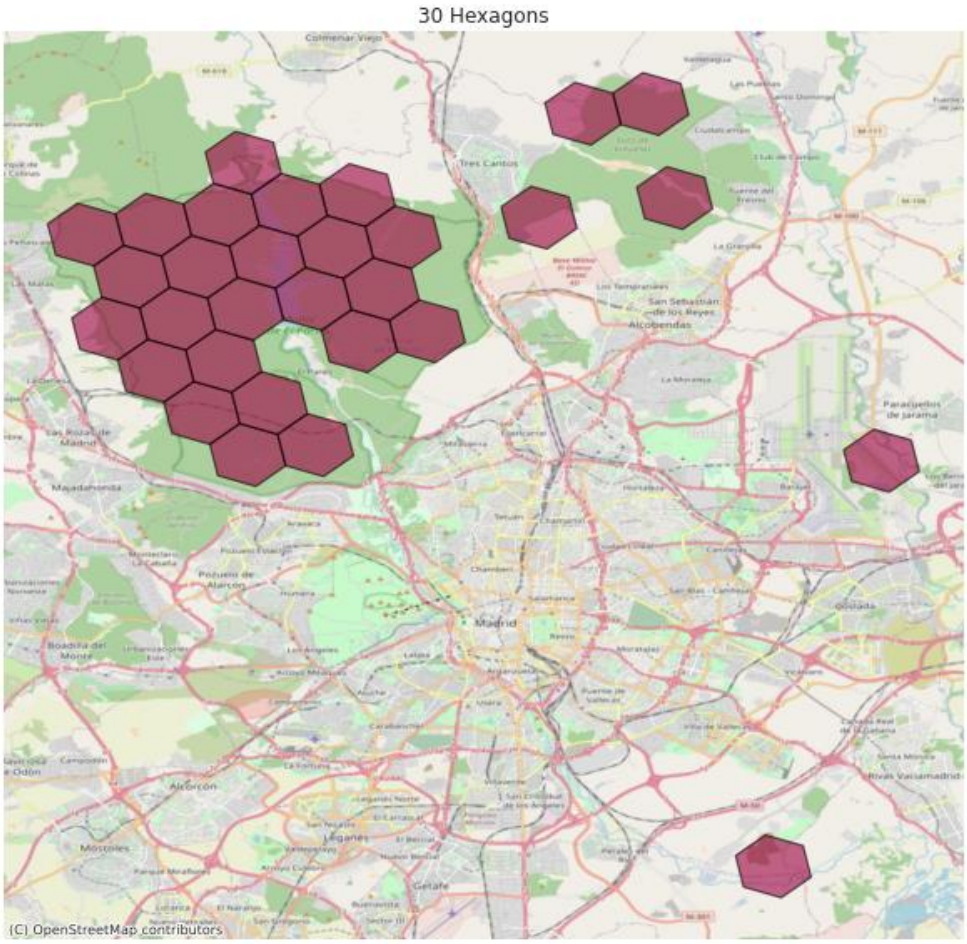

Ilustración 46. Disponibilidad de taxis en áreas "empty".

Esta distribución de áreas, en función de la tasa de disponibilidad media de taxis se ha utilizado para definir la estrategia con la que construir el *dataset* de tiempos de viaje entre dos puntos con la API de Google Maps, como se ha descrito en la sección 3.6.1.

4.2.2. Análisis por día de la semana

En el siguiente análisis realizado, se ha tratado de medir el efecto causado por la variable del día de la semana en la disponibilidad de taxis, además, se ha ampliado este estudio para evaluar su impacto en las diferentes zonas definidas.

En el siguiente gráfico (Ilustración 47) podemos observar cómo se distribuye la disponibilidad media de taxis por área en función de los días de la semana. En el eje x se representa la tasa media de taxis disponibles y en el eje y se representa el número de áreas con esas tasas.

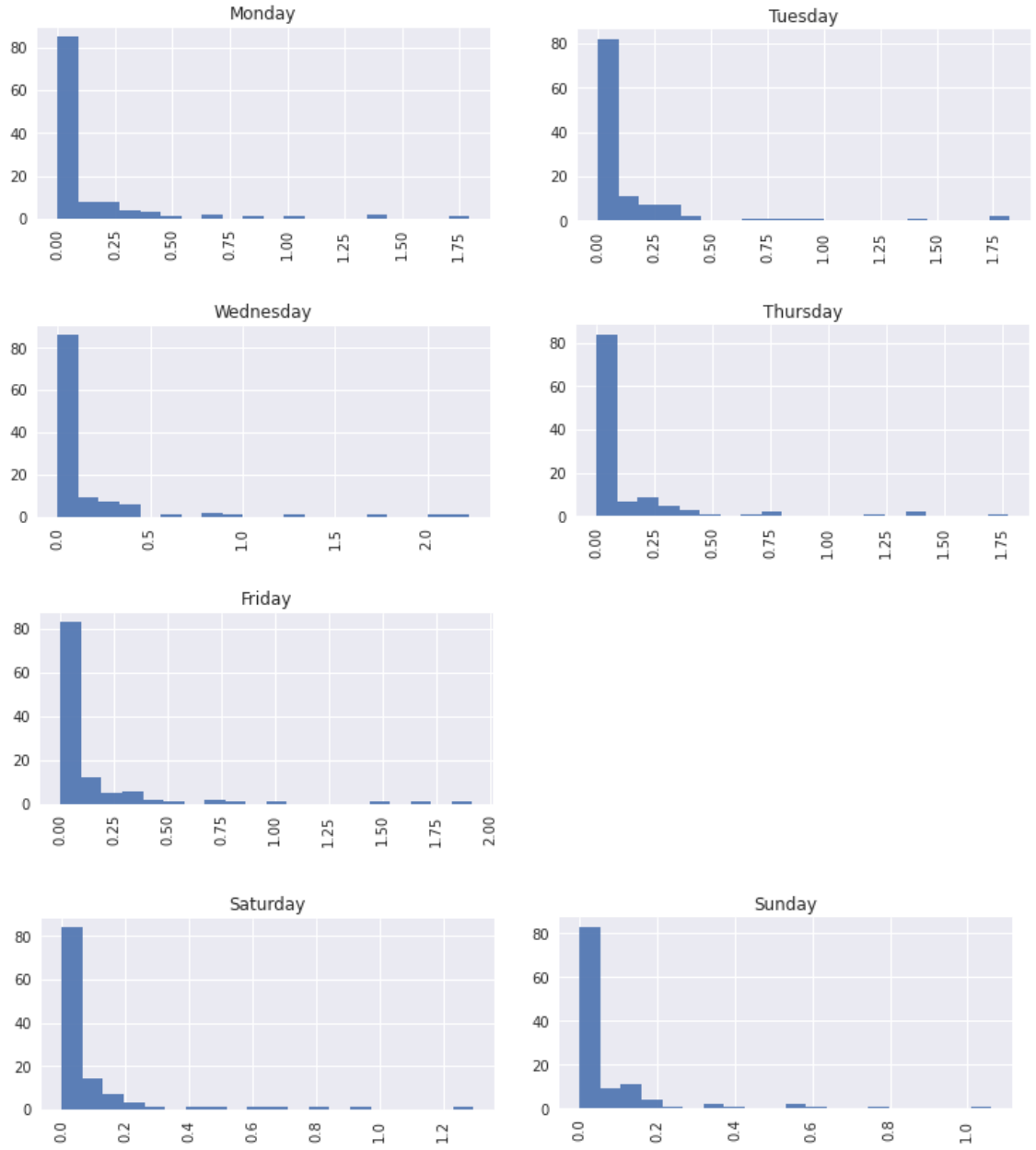

Ilustración 47. Distribución de la disponibilidad de taxis por día de la semana.

Como se puede apreciar en la imagen anterior (Ilustración 47), durante los días laborales la tasa de taxis disponibles es muy similar y la distribución de esta tasa por áreas es bastante estable, sin embargo, durante el fin de semana se reduce considerablemente la tasa de disponibilidad en todas las áreas, como se puede apreciar en los valores representados en el eje x.

En la siguiente tabla (Tabla 7), se refleja la tasa media de taxis disponibles, agregada para cada uno de los días de la semana, teniendo en cuenta los datos de todas las áreas sin diferenciar entre ellas. Como se puede observar, el efecto del fin de semana es muy notable con una reducción en torno al 40% frente a los días laborables.

Tabla 7. Tasa de disponibilidad media por día de la semana.

Día de la semana	Tasa de disponibilidad media
Monday	0,127155
Tuesday	0,131093
Wednesday	0,157231
Thursday	0,128578
Friday	0,138235
Saturday	0,081489
Sunday	0,071197

Tras analizar estos resultados se plantean dos cuestiones a resolver, por un lado, aunque la tasa de disponibilidad sea similar para todos los días laborables, ¿las áreas de alta, media y baja disponibilidad serán las mismas independientemente del día de la semana?

Por otro lado, los fines de semana tienen una menor tasa de disponibilidad, ¿esto afectará a la forma en que se distribuyen las áreas de alta, media y baja disponibilidad o se mantendrá la misma distribución con una caída generalizada proporcional en la disponibilidad de los taxis?

En la siguiente imagen (Ilustración 48) resolveremos la primera duda planteada. Como podemos observar, el comportamiento es muy similar durante todos los días laborables de la semana, en todos ellos se mantiene una alta tasa de disponibilidad en la zona centro y en los alrededores, es decir, dentro de la "almendra central" de Madrid, así como en la zona del aeropuerto.

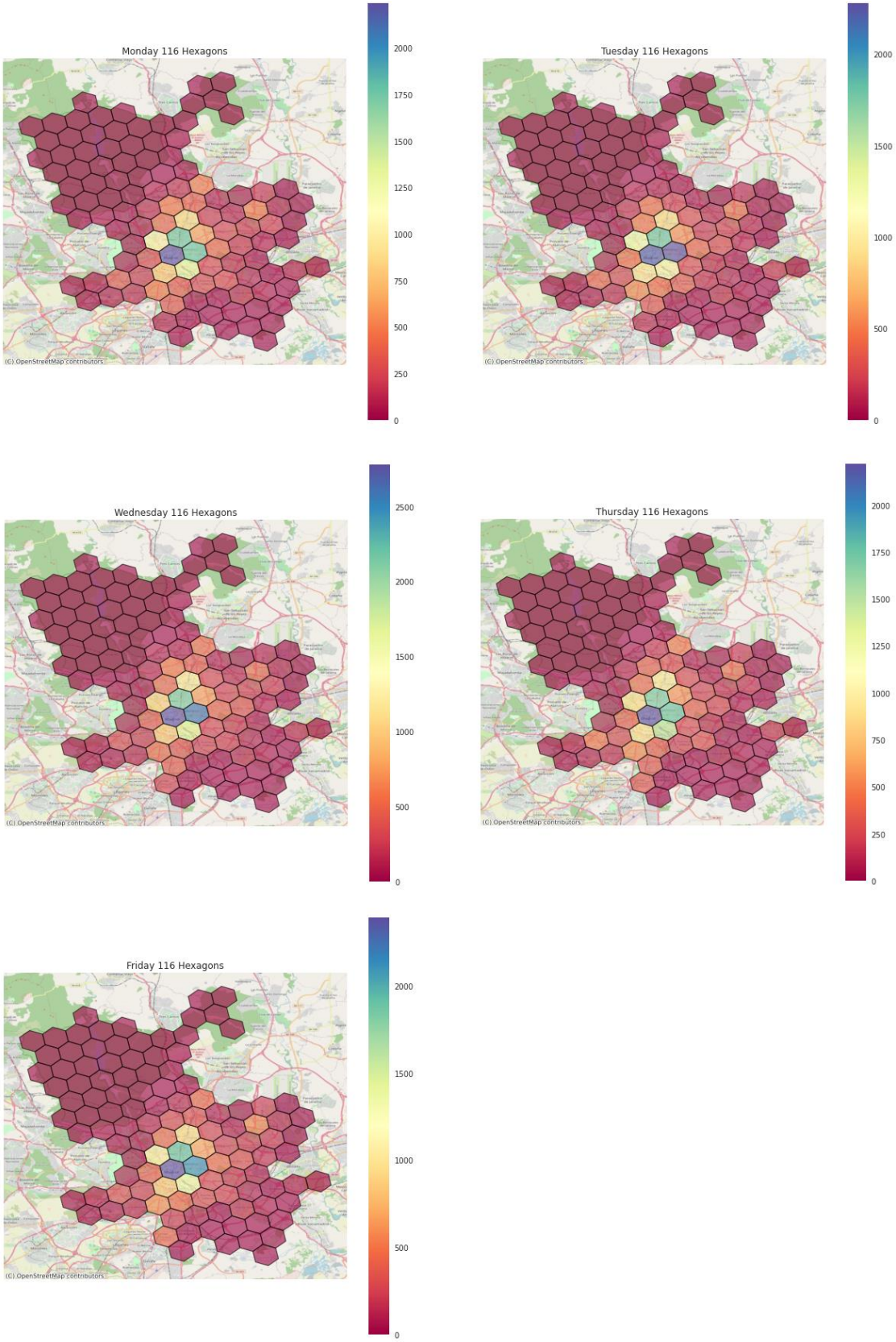

Ilustración 48. Tasa de disponibilidad por área y día de la semana (laborables).

Por último, si analizamos la tasa media de taxis disponibles durante el fin de semana, podemos observar que aunque tiene una distribución similar, sufre cierta contracción en algunas áreas interiores y se desplaza prácticamente toda la oferta de taxis disponibles a unas pocas áreas en el centro del municipio.

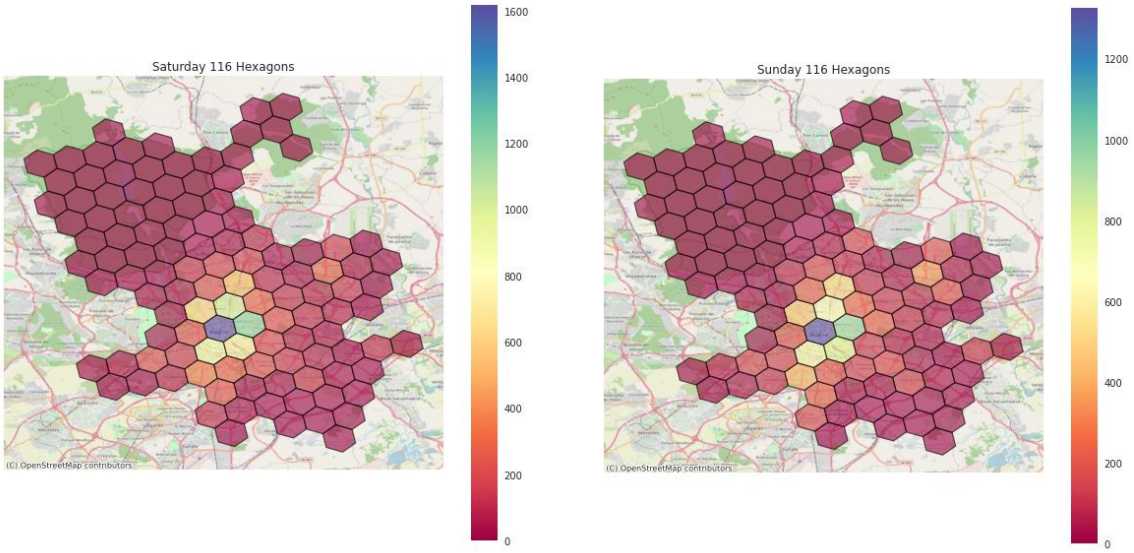

Ilustración 49. Tasa de disponibilidad por área y día de la semana (fin de semana).

4.2.3. Análisis por hora y periodo del día

Tras completar los análisis anteriores, en esta sección se medirá el efecto de la hora del día en la tasa de disponibilidad media, con el objetivo de identificar patrones en los datos que nos indiquen diferencias durante los distintos periodos del día.

En la siguiente tabla (Tabla 8) se muestra la tasa de disponibilidad media, considerando los datos de todas las áreas para cada una de las horas del día. Como se puede observar, en las horas centrales del día, desde las 9am hasta las 7pm, es cuando la tasa de disponibilidad es mayor. Por otro lado, durante la noche, desde las 10pm hasta las 7am, la tasa de disponibilidad media cae hasta niveles mínimos.

Tabla 8. Tasa de disponibilidad media por hora del día.

	Día de la semana	Tasa de disponibilidad
Night	0	0,053287
	1	0,029770
	2	0,023328
	3	0,021102
	4	0,020841
	5	0,033180
Morning	6	0,060392
	7	0,084549
	8	0,121424
	9	0,162775
	10	0,184184
	11	0,197234
Afternoon	12	0,202444
	13	0,200218
	14	0,182858
	15	0,170543
	16	0,165214
	17	0,165238
Evening	18	0,166161
	19	0,158346
	20	0,134805
	21	0,121992
	22	0,107380
	23	0,095514

Debido al volumen de datos disponible, es necesario generar grupos más amplios para poder analizar los datos por áreas geográficas en función de las horas del día. Para ello, se han definido una serie de grupos uniformes, teniendo en cuenta los periodos naturales del día, son los siguientes:

- ***Night***: Desde las 00:00 hasta las 5:59.
- ***Morning***: Desde las 6:00 hasta las 11:59.
- ***Afternoon***: Desde las 12:00 hasta las 17:59.
- ***Evening***: Desde las 18:00 hasta las 23:59.

En la siguiente imagen (Ilustración 50), se ha representado la tasa de disponibilidad para cada una de las áreas en función de los grupos definidos anteriormente para cada periodo del día:

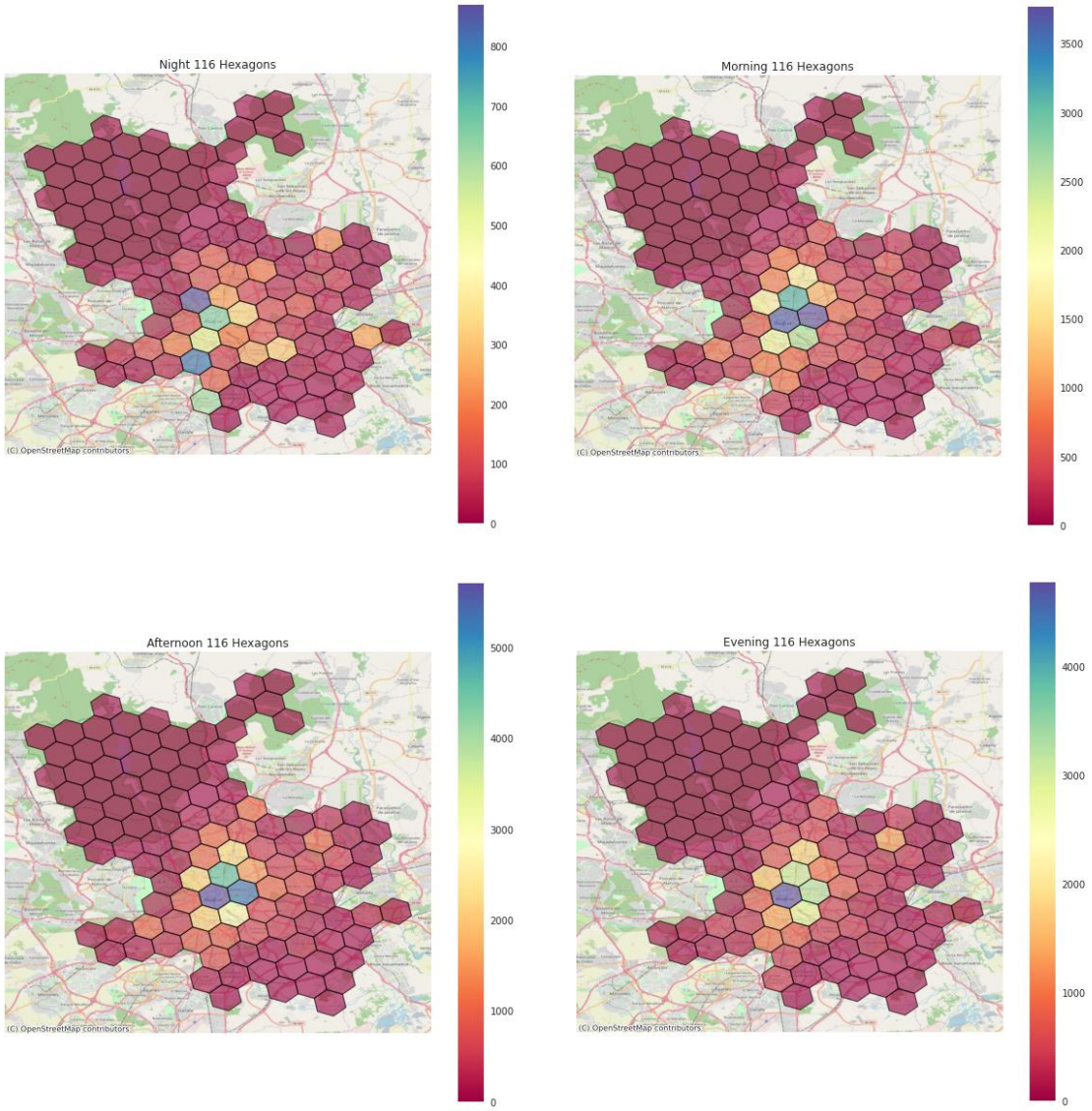

Ilustración 50. Tasa de disponibilidad por área y periodo del día.

Como se puede observar en estos gráficos, durante las horas de la mañana ("*morning*") y de la tarde ("*afernoon*") la distribución de áreas es muy similar y sigue los criterios ya mencionados anteriormente, mayor concentración en el área central del municipio, especialmente en el centro. Por otro lado, durante la tarde-noche ("*evening*") se puede observar como lás áreas del interior, fuera del centro, reducen su tasa de disponibilidad media. Por último, cabe destacar que durante la noche ("*night*"), como ya se había mencionado anteriormente, la tasa de disponibilidad cae drasticamente, además, cambia

completamente la distribución de las áreas. En este caso, se difuminan las áreas de mayor diponibilidad y el centro deja de ser la zona con mayor concentración.

4.3. Pruebas realizadas con datos reales

En esta sección se comprobará el rendimiento de los modelos desarrollados mediante datos reales de los viajes realizados por los usuarios de la aplicación.

Para ello, se ha utilizado el *dataset* mencionado en la sección 3.2.1. Este conjunto de datos contiene un total de 159 viajes que incluyen las coordenadas del punto de recogida del usuario, la fecha y hora en que solicitó el viaje y la fecha y hora en que el taxi llegó al punto de recogida del usuario. En la siguiente imagen (Ilustración 51), se muestra la localización de dichos puntos, ubicados en Madrid.

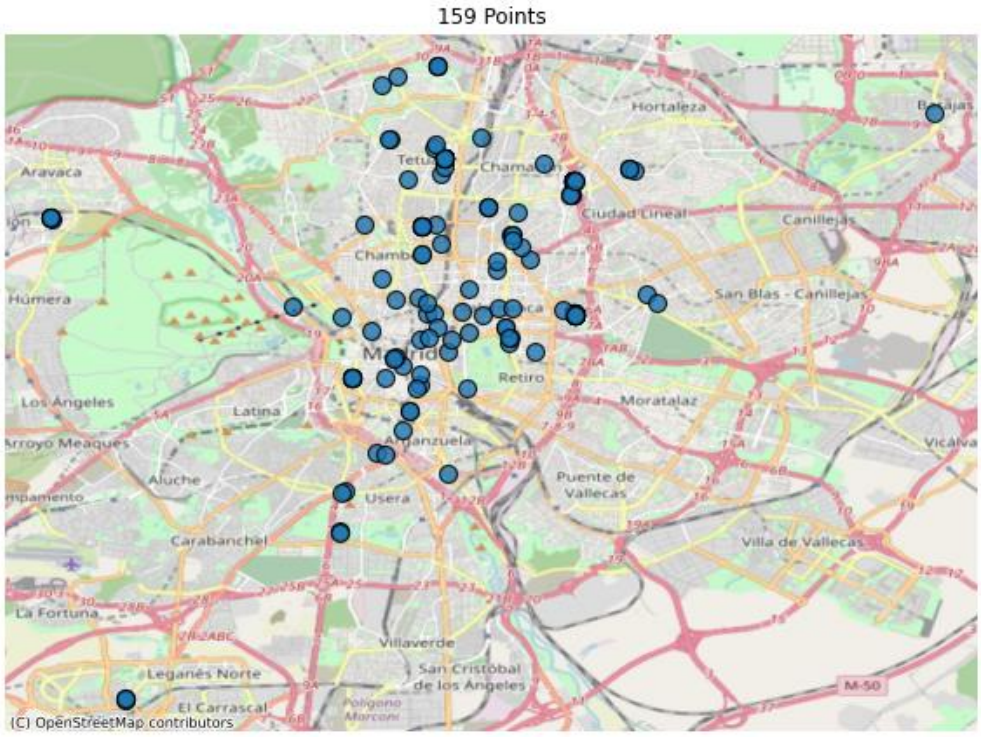

Ilustración 51. Viajes en taxi, puntos de recogida.

A partir de estos datos, se ha calculado el tiempo de recogida como la diferencia entre la fecha de solicitud del viaje y la fecha de llegada al punto de recogida donde se encuentra el usuario. A continuación, se han cruzado todos estos datos con la matriz creada a partir de los dos modelos desarrollados (predicción de disponibilidad y estimación de tiempo de viaje) y, por último, se han comparado los resultados reales frente a las estimaciones

procedentes de ambos modelos. En la siguiente imagen (Ilustración 52) se muestra de forma gráfica esta comparativa.

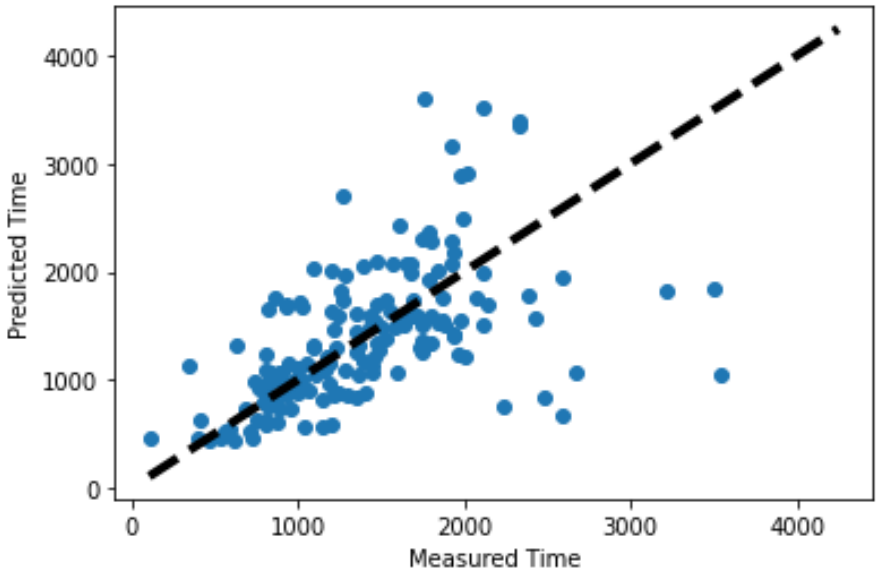

Ilustración 52. Tiempo de recogida real frente al estimado.

Como se puede apreciar en la imagen anterior (Ilustración 54), en líneas generales las estimaciones y las mediciones reales se aproximan a la realidad, aunque ha aumentado la dispersión frente a los resultados que se obtuvieron al validar los conjuntos de prueba de los modelos por separado. En este caso se ha obtenido una métrica R^2 de 0.063, un valor bastante reducido, debido a que algunas de las estimaciones se han desviado mucho del valor real.

Esto se debe principalmente a dos factores. Por un lado, al unir los modelos las tasas de error de ambos se combinan empeorando los resultados individuales. Por otro lado, debido a la limitación en las peticiones a la API de Google Maps, se ha construido un *dataset* únicamente con 49.803 registros, donde los puntos de origen y destino son siempre los centroides de las áreas definidas. Esto implica que cuanto más alejado esté un punto (origen o destino) del centroide, mayor será la tasa de error.

Para validar esta última hipótesis, se ha realizado un análisis sobre el modelo de estimación de tiempos, para ello se han seleccionado 1.000 puntos aleatorios en el municipio de Madrid y se ha comparado el tiempo medido a través de la API de Google frente a la estimación del modelo.

En las siguientes imágenes se pueden ver estos 1.000 puntos representados sobre el mapa de Madrid (Ilustración 53) y los resultados de comparar ambas métricas (Ilustración 54).

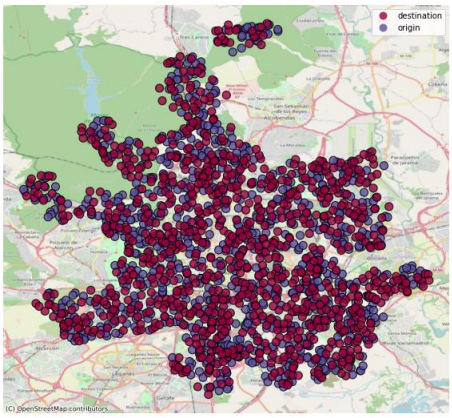

Ilustración 53. Puntos aleatorios generados en el área de Madrid.

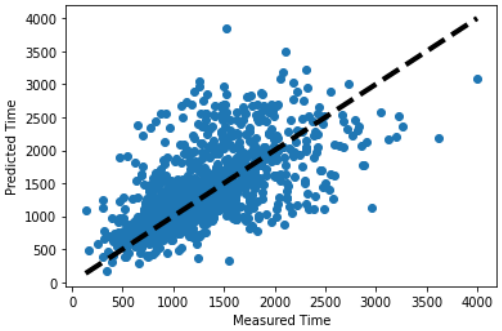

Ilustración 54. Diferencia entre las estimaciones de tiempo reales y las predichas.

Como se puede observar en dicha imagen (Ilustración 54), se aprecia un comportamiento similar al descrito anteriormente. La gran mayoría de las estimaciones del modelo se ajustan a la realidad, sin embargo, existen algunas de ellas, asociadas a los puntos más alejados del centroide de las áreas definidas donde el error es muy amplio. Esto provoca que la ratio de error haya aumentado significativamente frente a los resultados de validación del modelo (Ilustración 39) analizados en la sección 3.6.2.

En futuras iteraciones, abordar el problema derivado de la fuente de datos, es decir, debido a que los datos de tiempos de viaje están limitados a los centroides (tanto los puntos de origen como los de destino), será fundamental para mejorar los resultados y aumentar la precisión del modelo. Otra posible alternativa consistiría en reducir el tamaño de las áreas (actualmente de $5.16 km^2$) aumentando la precisión de los hexágonos definidos, reduciendo el área de cada uno a $0.73 km^2$ y multiplicando por siete el número total de centroides, hasta los 812. En ambos casos, será necesario aumentar considerablemente el volumen de los datos disponibles.

5. Conclusiones y trabajo futuro

En este capítulo se analizarán los retos, logros y problemas de las distintas fases del proyecto, así como los resultados obtenidos y se plantearán las futuras líneas de trabajo.

5.1. Conclusiones

En este Trabajo de Fin de Máster se ha desarrollado un sistema capaz de estimar el tiempo de espera para cada usuario del servicio de taxis en un momento y ubicación concretos, proporcionando información en tiempo real y con un tiempo de respuesta mínimo que garantice una experiencia de usuario óptima.

Para ello, se ha desarrollado un proyecto completo desde las primeras fases de análisis, donde se ha estudiado la situación del sector y las capacidades de la competencia, además de investigar todos los recursos necesarios para el desarrollo de este: desde recopilar fuentes de datos, analizar algoritmos, comparar plataformas *cloud* o indagar en todas las librerías que podrían aplicarse al problema. A continuación, se ha diseñado una arquitectura totalmente *serverless*, optimizada en cuanto a capacidad y coste, capaz de cubrir cualquiera de las necesidades del proyecto, desde la recopilación y procesamiento de información, hasta el análisis de los datos y la creación de modelos predictivos.

Respecto a los modelos desarrollados, se han aplicado dos algoritmos diferentes de *ensemble*, *random forest* (*bagging*) y XGBoost (*boosting*), junto con diferentes técnicas de optimización de evaluación de estos. Además, partiendo de un entorno de datos muy limitado (o nulo) se ha conseguido recopilar información externa y crear una fuente de datos adicional para obtener unos resultados ajustados que prueban la certeza de la hipótesis inicial e invitan a invertir esfuerzos adicionales en aumentar la calidad de la información disponible para mejorar los modelos predictivos desarrollados.

Por último, se ha diseñado un sistema basado en una base de datos NoSQL combinada con una caché en memoria que ha permitido dar respuesta a las peticiones en un tiempo de respuesta mínimo, de milisegundos a microsegundos; de forma que se ha logrado alcanzar otro de los objetivos críticos definidos al inicio del proyecto.

5.2. Trabajo futuro

El sistema desarrollado tiene numerosas vías de desarrollo y es posible explorar diferentes alternativas para mejorar la calidad de las estimaciones. En primer lugar, uno de los

aspectos más importantes para incrementar la precisión de los modelos consistiría en aumentar el histórico de datos disponibles, además, en el futuro habría que analizar el impacto de la pandemia en los datos recogidos durante estos meses. En este proyecto, los conjuntos de datos utilizados en el desarrollo de los modelos analíticos cuentan con un histórico de 3 meses, desde el inicio de febrero hasta el final de abril de 2021. Incrementar el número de datos disponibles repercutirá directamente en la calidad de los modelos desarrollados, incrementando la precisión de las estimaciones. Además, en el caso del *dataset* con los tiempos de viaje entre dos puntos, es indispensable enriquecer los datos disponibles con puntos aleatorios (tanto de origen como de destino), ampliando así el limitado enfoque basado en los centroides. Adicionalmente, cuando se disponga de un histórico de varios años, será posible incluir nuevas variables (meses, estaciones, etc.) lo que permitirá capturar ciertos sesgos derivados de la estacionalidad.

Por otro lado, otra mejora relevante podría ser aumentar la precisión de las zonas definidas mediante hexágonos, reduciendo la extensión geográfica (km^2) de estas, es decir, creando un mayor número de áreas con un tamaño más reducido. Esto está directamente relacionado con el punto anterior, ya que para poder definir áreas más reducidas que en el modelo actual es necesario aumentar considerablemente el volumen de los datos disponibles para garantizar un tamaño mínimo de muestras en cada una de las áreas.

Adicionalmente, se podría incluir información de fuentes externas como datos de predicciones meteorológicas, información del tráfico, calendarios de festivos y eventos e información geográfica asociada a áreas de interés, por ejemplo, la ubicación de grandes superficies comerciales, conectores de transporte público, áreas empresariales, zonas residenciales, etc.

Otra posible alternativa, consistiría en explorar otro tipo de modelos. Este proyecto ha estado centrado en algoritmos de *ensemble*, concretamente en dos tipos de modelos, *random forest* y *XGBoost*. Sería interesante explorar otras alternativas y comparar los resultados obtenidos frente a los algoritmos actuales.

Por último, se podría plantear extender el análisis realizado en este proyecto, actualizando algunos de los objetivos, a otro tipo de servicios que se encuentran en auge: el sector de los vehículos compartidos (carsharing y motosharing). Es posible aplicar el modelo de disponibilidad por áreas para detectar zonas con ineficiencias en el servicio, donde el tiempo de espera del usuario hasta encontrar un vehículo disponible es elevado.

Bibliografía

[1] Vicente Pinilla y Luis Antonio Sáez. "La despoblación rural en España: Génesis de un problema y políticas innovadoras". http://sspa-network.eu/wp-content/uploads/Informe-CEDDAR-def-logo.pdf

[2] Xavier Querol. "La calidad del aire en las ciudades: Un reto mundial". http://www.fundacionnaturgy.org/wp-content/uploads/2018/06/calidad-del-aire-reto-mundial.pdf

[3] Parlamento Europeo. "Emisiones de CO2 de los coches: hechos y cifras". https://www.europarl.europa.eu/news/es/headlines/society/20190313STO31218/emisiones-de-co2-de-los-coches-hechos-y-cifras-infografia

[4] Ayuntamiento de Madrid. "Memoria Anual Calidad del Aire 2020". http://www.mambiente.munimadrid.es/opencms/export/sites/default/calaire/Anexos/Memorias/MEMORIA_2020.pdf

[5] Gordon S. Bauer, Jeffery B. Greenblatt, Brian F. Gerke. "Cost, Energy, and Environmental Impact of Automated Electric Taxi Fleets in Manhattan. Environmental science & technology". DOI: 10.1021/acs.est.7b04732

[6] Gordon S. Bauer, Cheng Zheng, Jeffery B. Greenblatt, Susan Shaheen, Daniel M. Kammen. "On-Demand Automotive Fleet Electrification Can Catalyze Global Transportation Decarbonization and Smart Urban Mobility". DOI: 10.1021/acs.est.0c01609

[7] Franziska Bell and Slawek Smyl (2018). "Forecasting at Uber: An Introduction", retrieved from: https://eng.uber.com/forecasting-introduction/

[8] Ye Tianyu (2019). "Mind the Gap: A Case Study of Demand Prediction and Factors Affecting Waiting Time at Uber NYC and Washington D.C.", retrieved from: https://escholarship.org/uc/item/80q5f8t9

[9] Uber Technologies Inc. https://www.uber.com/

[10] Cabify España S.L.U. https://cabify.com/

[11] Auro Travel. https://www.auro.travel/

[12] Free Now. https://free-now.com/

[13] Joinup. https://joinup.es/

[14] PideTaxi. https://pidetaxi.es/

[15] Esri. "The ArcGIS Book". https://downloads.esri.com/LearnArcGIS/pdf/the-arcgis-book-second-edition.pdf

[16] Geoportal de la Comunidad de Madrid. https://www.comunidad.madrid/servicios/mapas/geoportal-comunidad-madrid

[17] Geoportal del ayuntamiento de Madrid, https://geoportal.madrid.es

[18] Esri. "ESRI Shapefile Technical Description - An ESRI White Paper". https://www.esri.com/content/dam/esrisites/sitecore-archive/Files/Pdfs/library/whitepapers/pdfs/shapefile.pdf

[19] Howard Butler, Martin Daly, Allan Doyle, Sean Gillies, Stefan Hagen, Tim Schaub. "The GeoJSON Specification (RFC 7946)". https://datatracker.ietf.org/doc/html/rfc7946

[20] Chris Lambert, "Build on top of your public Latitude location with GeoJSON & KML". https://mapsplatform.googleblog.com/2009/05/build-on-top-of-your-public-latitude.html

[21] Open Geospatial Consortium. "OGC® GeoPackage Encoding Standard". https://www.geopackage.org/spec130/

[22] Andreas C. Müller, Sarah Guido. "Introduction to Machine Learning with Python". ISBN: 9781449369415

[23] David Opitz, Richard Maclin. "Popular Ensemble Methods: An Empirical Study". DOI: https://doi.org/10.1613/jair.614.

[24] Zhi-Hua Zhou. "Ensemble Methods: Foundations and Algorithms". ISBN: 978-1-439-83003-1.

[25] Thomas G. Dietterich. "An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization". https://link.springer.com/content/pdf/10.1023/A:1007607513941.pdf

[26] Leo Breiman, "Random Forests. Machine Learning, 45, 5–32, 2001", retrieved from: https://link.springer.com/content/pdf/10.1023/A:1010933404324.pdf

[27] Saul Dobilas. "XGBoost: Extreme Gradient Boosting — How to Improve on Regular Gradient Boosting?". https://towardsdatascience.com/xgboost-extreme-gradient-boosting-how-to-improve-on-regular-gradient-boosting-5c6acf66c70a

[28] Amazon Web Services. https://aws.amazon.com/es/products/

[29] Microsoft Azure. https://azure.microsoft.com/es-es/services/

[30] Google Cloud Platform. https://cloud.google.com/products/?hl=es

[31] Raj Bala, Bob Gill, Dennis Smith, David Wright y Kevin Ji. "Magic quadrant for cloud infrastructure and platform services". https://www.gartner.com/doc/reprints?id=1-1ZDZDMTF&ct=200703&st=sb

[32] Jake VanderPlas, "Python Data Science Handbook: Essential Tools for Working with Data", ISBN-10: 149191205

[33] Wes McKinney, "Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython", ISBN-10: 1491957662

[34] Wes McKinney and the Pandas Development Team, "Pandas: powerful Python data analysis toolkit", retrieved from: https://pandas.pydata.org/docs/pandas.pdf

[35] Wes McKinney. "Pandas: A Foundational Python Library for Data Analysis and Statistics". https://www.dlr.de/sc/Portaldata/15/Resources/dokumente/pyhpc2011/submissions/pyhpc2011_submission_9.pdf

[36] Stefan van der Walt, S. Chris Colbert, Gaël Varoquaux. "The NumPy array: a structure for efficient numerical computation". DOI: 10.1109/MCSE.2011.37

[37] Robert Johansson. "Numerical Python: Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib", ISBN-10: 1484242459.

[38] Matplotlib 3.4.2 documentation. https://matplotlib.org/stable/contents.html

[39] Contextily: context geo tiles in Python. https://contextily.readthedocs.io/en/latest/

[40] Folium 0.12.1 documentation, https://python-visualization.github.io/folium/

[41] GeoPandas, API Reference. https://geopandas.org/docs/reference.html

[42] Isaac Brodsky, "H3: Uber's Hexagonal Hierarchical Spatial Index". https://eng.uber.com/h3/

[43] Hexagonal hierarchical geospatial indexing system. https://h3geo.org/docs/

[44] H3 Pyton, repositorio Github, https://github.com/uber/h3-py

[45] Anisya, Ganda Yoga Swara. “Implementation Of Haversine Formula And Best First Search Method In Searching Of Tsunami Evacuation Route”. DOI:012004. 10.1088/1755-1315/97/1/012004.

[46] Python Client for Google Maps Services, https://googlemaps.github.io/google-maps-services-python/docs/index.html

[47] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Édouard Duchesnay, “Scikit-learn: Machine Learning in Python”, Journal of Machine Learning Research, retrieved from: https://jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf

[48] Scikit-learn algorithm cheat-sheet. https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

[49] Joaquín Amat Rodrigo, “Machine learning con Python y Scikit-learn”, retrieved from: https://www.cienciadedatos.net/documentos/py06_machine_learning_python_scikitlearn.html

[50] Tianqi Chen, Carlos Guestrin, “XGBoost: A Scalable Tree Boosting System”, retrieved from: https://arxiv.org/pdf/1603.02754.pdf

[51] GADM data, version 3.6. https://gadm.org/download_country_v3.html

[52] Geoportal del ayuntamiento de Madrid, https://geoportal.madrid.es/

[53] Amazon Kinesis Data Firehose. https://aws.amazon.com/es/kinesis/data-firehose/

[54] Sklearn.ensemble.RandomForestClassifier Parameters. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[55] XGBoost Parameters. https://xgboost.readthedocs.io/en/latest/parameter.html

[56] NYC Taxi and Limousine Commission (TLC), Trip Record Data. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

[57] Cort Willmott, Steven Ackleson, Robert Davis, Johannes Feddema, Klink Katherine, Legates David, James O'Donnell, Clinton Rowe. “Statistics for the Evaluation and Comparison of Models”. DOI: 10.1029/JC090iC05p08995

[58] In-Memory Acceleration with DynamoDB Accelerator (DAX), Use Cases https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DAX.html#DAX.use-cases

[59] Amazon DynamoDB Accelerator (DAX). https://aws.amazon.com/es/dynamodb/dax/

[60] Synergy Research Group. “Amazon and Microsoft maintain their grip on the market, but others are also growing rapidly”. https://www.srgresearch.com/articles/amazon-and-microsoft-maintain-their-grip-market-others-are-also-growing-rapidly

[61] Synergy Research Group. “Cloud market ends 2020 on a high while Microsoft continues to gain ground on Amazon”. https://www.srgresearch.com/articles/cloud-market-ends-2020-high-while-microsoft-continues-gain-ground-amazon

[62] Capa gratuita de AWS, https://aws.amazon.com/es/free/

[63] AWS Educate, https://aws.amazon.com/es/education/awseducate/

[64] Microsoft Azure Free Tier, https://azure.microsoft.com/es-es/free/

[65] Google Cloud Platform Free Tier, https://cloud.google.com/free/

Definiciones

AWS Athena: Servicio de AWS para el análisis de datos mediante consultas SQL.

AWS Cloudwatch: Servicio que permite programar eventos automáticos que se ejecutan periódicamente.

AWS EMR: Amazon *Elastic Map Reduce*, es servicio de computación de Amazon Web Services basado en un clúster de computadoras.

AWS Glue Data Catalog: Servicio de catálogo de datos.

AWS Kinesis Data Streams: Servicio de colas de mensajes.

AWS Kinesis Firehose: Servicio para procesar datos en *streamming*.

AWS Lambda: Servicio que permite ejecutar el código desarrollado en una amplia variedad de lenguajes de programación.

AWS S3: Amazon *Simple Storage Service*, es un servicio de almacenamiento de objetos.

Boundaries: Límites geográficos definidos para un territorio o área.

Bucket: Contenedor de objetos. Es la forma en que se organizan los objetos en AWS S3.

Carsharing: Servicio de coches eléctricos compartidos.

Cloud: Diferentes tipos de servicios ubicados en la nube.

Cross-validation: La validación cruzada o *cross-validation*, es una técnica utilizada en la evaluación de modelos estadísticos.

Dataframe: Estructura de datos utilizada en la librería Pandas.

Dataset: Conjunto de datos.

Geodataframe: Estructura de datos geométricos utilizado en la librería GeoPandas.

Machine learning: Algoritmos de aprendizaje automático.

Missing data: Registros no informados dentro de un conjunto de datos.

Motosharing: Servicio de motos eléctricas compartidas.

Parquet: Formato comprimido que almacena los datos en forma de columnas y que incluye los metadatos dentro del propio fichero.

Serverless: Sistema que no requiere un servidor físico, es decir, no es necesario configurar ni administrar ninguna infraestructura.

Acrónimos

API: *Application Programming Interface*, es un conjunto de funciones y protocolos de un sistema, ofrecidos con una capa mayor de abstracción para ser utilizadas por otro software.

AWS: Amazon Web Services.

CRS: *Coordinate Reference System*, permite definir cómo un mapa proyectado en dos dimensiones se relaciona con ubicaciones reales de la tierra.

EPSG: *European Petroleum Survey Group*, es un repositorio de parámetros geodésicos que ofrece información acerca de sistemas de referencia y proyecciones cartográficas en todo el mundo.

GCP: Google Cloud Platform.

GIS: *Geographic Information System*.

GML: *Geography Markup Language*.

KML: *Keyhole Markup Language*.

MAE: Error absoluto medio o *Mean Absolute Error.*

MSE: Error cuadrático medio o *Mean Squared Error*.

$\boldsymbol{R^2}$: Coeficiente de determinación o *coefficient of determination*.

RMSE: Raíz cuadrada del error cuadrático medio.

Anexos

Anexo A. Análisis de plataformas *cloud*

Con el objetivo de seleccionar la plataforma *cloud* sobre la que desarrollar el proyecto, se ha llevado a cabo un minucioso análisis de las diferentes alternativas disponibles en el mercado, comparando las distintas plataformas en función de sus capacidades y analizando los costes asociados a sus servicios y su oferta de créditos de prueba, docencia e investigación.

Posición en el mercado

Las principales plataformas en cuanto a volumen de mercado son Amazon Web Services, Microsoft Azure, Google Cloud Platform, Alibaba Cloud e IBM. En los siguientes gráficos, provenientes de varios estudios elaborados por *Synergy Reserach Group* se puede apreciar el volumen de mercado y el crecimiento asociado a cada una estas plataformas en los últimos años.

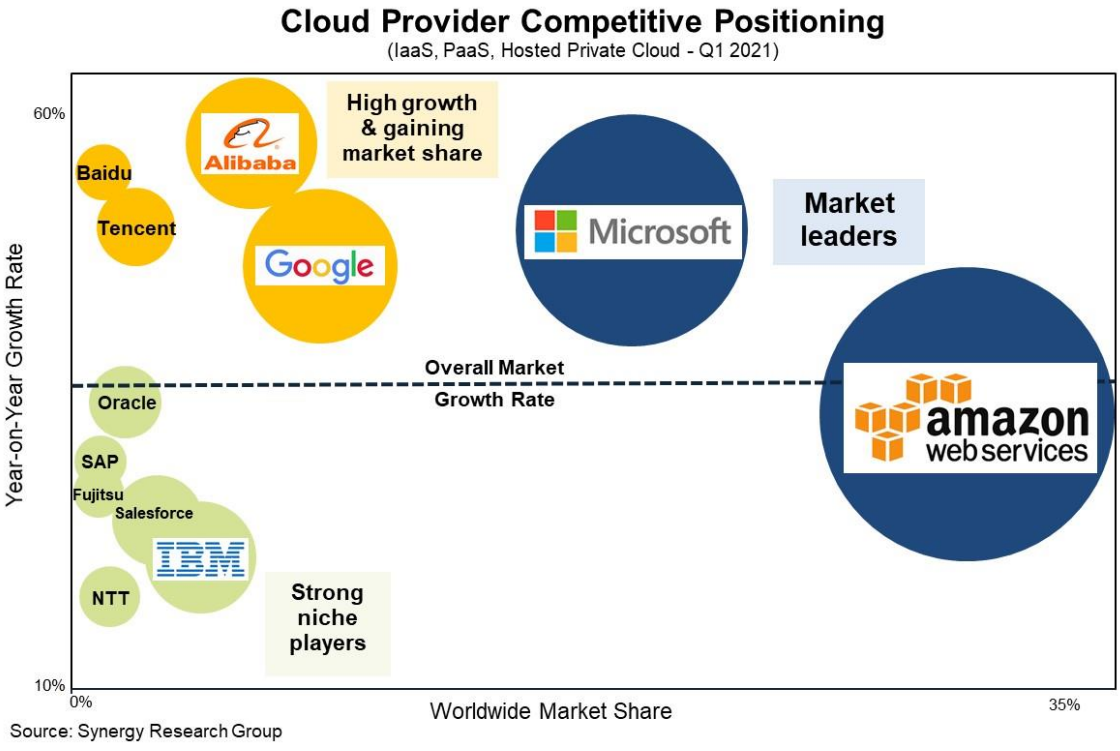

Ilustración 55. Posición en el mercado de las principales plataformas cloud [60].

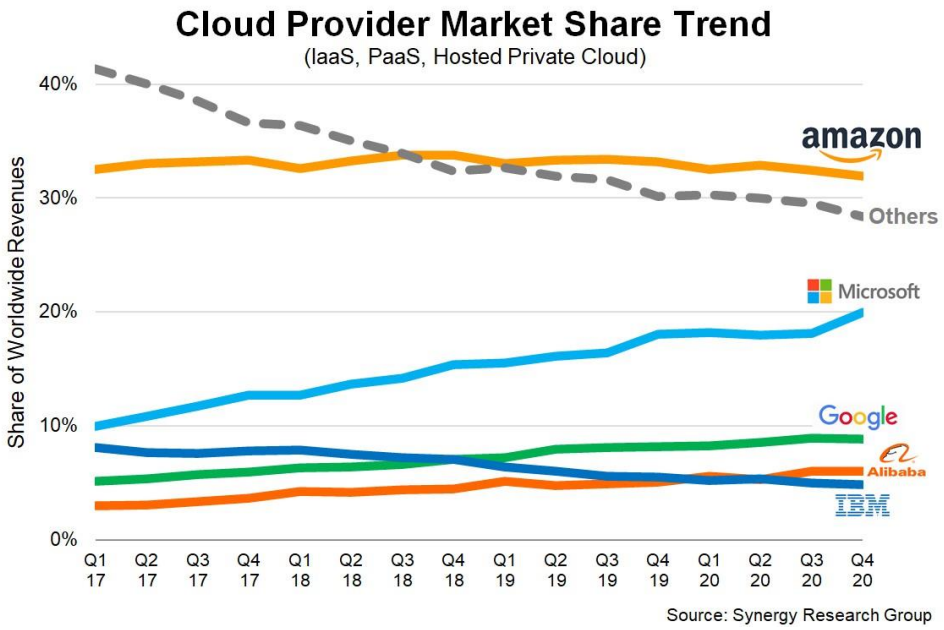

Ilustración 56. Crecimiento relativo de las diferentes plataformas cloud *[61].*

Amazon Web Services se sitúa como líder mundial en infraestructura y servicios *cloud* y mantiene una posición dominante en el mercado desde hace años. Por otro lado, IBM, a diferencia de las otras cuatro plataformas, centra su estrategia en impulsar su *cloud* privada (segmento dónde se mantiene como líder) frente a la apuesta por una *cloud* pública de Amazon, Microsoft, Google y Alibaba. En este sentido, este análisis se centrará en las tres principales plataformas basadas en *cloud*:

- **Amazon Web Services**. Líder, pionero y dominador actual del mercado, con más del 30% de la cuota de mercado. Mantiene una posición destacada, a pesar del alto crecimiento de sus competidores, gracias a su alto grado de innovación y la solidez de sus soluciones. Fue el primero de ellos en comercializar este tipo de servicios, lo que ha permitido que algunos de sus productos se conviertan en un estándar a nivel mundial. Su principal estrategia pasa por estudiar las necesidades de sus clientes e ir aumentando su catálogo de servicios frecuentemente.
- **Microsoft Azure**. Situado en segunda posición por cuota de mercado (20%), su crecimiento, evolución y adopción se ha acelerado de forma considerable. Microsoft ha sido capaz de aprovechar su amplia cartera de clientes, existente gracias a su sistema operativo y otras soluciones software. Su estrategia principal pasa por la integración directa entre todos sus productos y servicios.

- **Google Cloud Platform**. Posicionado en una tercera posición, a distancia de sus competidores directos, con menos del 10% de la cuota de mercado. Google, ha tenido un ritmo menor de adquisición de mercado. En los últimos años ha incrementado su inversión e innovación en servicios *cloud*. Su estrategia pasa por seguir incorporando y mejorando los servicios en los que tiene mayores prestaciones (computación y almacenamiento) junto con una apuesta por aumentar las capacidades analíticas avanzadas que ofrece.

Componentes y capacidades

Las tres plataformas mejor posicionadas en el mercado se encuentran un escalón por encima del resto de servicios *cloud*. La cuota de mercado de estas plataformas se puede ver directamente reflejada en una diferencia muy sustancial de madurez y funcionalidad en las capacidades técnicas ofrecidas por sus productos y soluciones. Además de las capacidades funcionales, el mayor foco en la usabilidad, sencillez, automatización e integración de todas las soluciones y herramientas incluidas es un factor clave.

Uno de los estudios con mayor reconocimiento en este ámbito es el elaborado periodicamente por Gartner. En este informe destaca el liderazgo de AWS frente al resto y el rápido crecimiento de sus principales competidores (Microsoft, Google y Alibaba). En la siguiente imagen se muestra un resumen de dicho informe.

Ilustración 57. Cuadrante de Gartner para infraestructuras y plataformas cloud [31]..

Las principales plataformas (Amazon, Microsoft y Google) presentan múltiples similitudes y ofrecen un conjunto de servicios gestionados (IaaS, PaaS y SaaS) con una filosofía similar y capacidades casi equivalentes:

- **Amazon Web Services**. Cuenta con un porfolio muy completo de servicios IaaS, PaaS y SaaS y un amplio *stack* de soluciones escalables de datos, procesamiento, *machine learning* y analítica que permiten dar solución a cualquier necesidad. Ofrece productos y servicios maduros en todos los casos, algunos de ellos referentes en el sector.
- **Microsoft Azure**. Reúne un conjunto de capacidades propias muy considerable y de gran fiabilidad, apoyándose en productos tradicionales y también en otros *partners* y fuentes *open-source* para complementar sus servicios. Dispone de capacidades avanzadas en cuanto a la integración de los diferentes componentes.
- **Google Cloud Platform**. Aporta un conjunto más reducido de soluciones, principalmente PaaS e IaaS nativas y totalmente escalables. Cuenta con un alto rendimiento y fiabilidad para el procesamiento analítico avanzado y almacenamiento de datos. El rendimiento de sus sistemas es muy elevado.

Costes y créditos de investigación

Todas las plataformas cuentan con ofertas de créditos gratuitos para utilizar sus diferentes componentes y servicios ofrecidos.

- **Amazon Web Services.** Amazon ofrece una capa gratuita que cuenta con una serie de servicios gratuitos durante 12 meses y otras ofertas que nunca vencen. Existen numerosos servicios dentro de esta capa, se han identificado los que podrían ser más relevantes en este proyecto:
    - ***Amazon API Gateway*** (servicios de aplicaciones), 1 millón de llamadas por mes.
    - ***Amazon EC2*** (computación), 750 horas al mes, con una instancia "t2.micro" de Linux (1 GiB de memoria y soporte de plataformas de 32 y 64 bits.
    - ***Amazon EBS*** (almacenamiento y entrega de contenido), 30 GB de cualquier combinación de uso general (SSD) o magnético, más 2 millones de E/S (con almacenamiento magnético de EBS) y 1 GB de almacenamiento de *snapshots*.

- ***Amazon RDS*** (base de datos relacional), 750 horas de instancias micro para ejecutar MySQL, PostgreSQL o MariaDB, entre otros. 20 GB de almacenamiento de base de datos y 10 millones de E/S.
- ***Amazon S3*** (almacenamiento y entrega de contenido), 5 GB de almacenamiento estándar, 20.000 solicitudes *Get* y 2.000 solicitudes *Put*.
- ***Amazon DynamoDB*** (base de datos), 25 GB de almacenamiento, 25 unidades de capacidad de lectura y 25 unidades de capacidad de escritura, suficiente para gestionar 200 millones de solicitudes al mes.
- ***AWS Lambda*** (computación), 1.000.000 de solicitudes gratuitas al mes.
- ***Amazon QuickSight*** (herramienta de visualización), 1 usuario, 1 GB de SPICE.
- ***Amazon Glacier*** (almacenamiento en frío), 10 GB de recuperaciones de datos.
- ***Amazon Glue*** (ETL y catálogo de datos), 1 millón de objetos almacenados en el catálogo de datos y 1 millón de solicitudes realizadas por mes al catálogo de datos.
- ***Transferencia de datos***, 15 GB de transferencia de datos salientes y 1 GB de transferencia de datos regionales agregados en todos los servicios de AWS.

Adicionalmente, Amazon Web Services ofrece AWS Educate, una plataforma dónde es posible acceder a contenido formativo gratuitamente y que además otorga créditos gratuitos (150$ renovables anualmente) para emplear en cualquiera de los servicios de la plataforma. Este servicio requiere un correo institucional.

- **Microsoft Azure**. Microsoft ofrece un crédito de 200$ para usar durante 1 mes, 12 meses de algunos servicios gratuitos y 25 servicios gratuitos siempre. Dentro de estos servicios gratuitos, se han identificado los que podrían ser relevantes:
  - ***Máquinas virtuales con Linux*** (computación), 750 horas.
  - ***Máquinas virtuales con Windows*** (computación), 750 horas.
  - ***Blob Storage*** (almacenamiento), 5 GB.
  - ***File Storage*** (almacenamiento), 5 GB.
  - ***SQL Database*** (base de datos), 250 GB.
  - ***Cosmos DB*** (base de datos), 5 GB, 400 unidades de solicitud.

- ***Machine Learning Studio*** (análisis), 100 módulos por experimento.

- **Google Cloud Platform**. Google ofrece un crédito de 300$ para usar durante 12 meses y una serie de servicios con unos límites de uso gratuitos. Dentro de estos servicios gratuitos, se han identificado los que podrían ser relevantes:
  - ***Google APP Engine*** (computación), 28 horas diarias de computación y 5 GB de almacenamiento en la nube.
  - ***Google Cloud Datastore*** (base de datos), 1 GB de almacenamiento, 50.000 lecturas, 20.000 escrituras, 20.000 borrados, por día.
  - ***Google Compute Engine*** (computación), 1 f1-micro *instance*.
  - ***Google Cloud Storage*** (almacenamiento), 5 GB de almacenamiento con una política regional.
  - ***Google Cloud Functions*** (computación), 2 millones de llamadas al mes.
  - ***Google BigQuery*** (base de datos), 1 TB de consultas mensuales y 10 GB de almacenamiento.

Conclusión

Como conclusión del análisis realizado, es posible determinar que actualmente existen tres plataformas basadas en *cloud* pública que destacan frente al resto de competidores, tanto en su posicionamiento de mercado como en sus capacidades y madurez de los servicios ofrecidos. Estas plataformas son: Amazon Web Services, Microsoft Azure y Google Cloud Platform.

Todas estas plataformas cumplen con las necesidades del proyecto, sin embargo, Amazon Web Services está un paso por delante de Microsoft y Google. Es la que ofrece una mayor variedad de servicios y un mayor nivel de madurez de estos. Además, cuenta con un gran número de componentes relevantes para este proyecto de forma gratuita.