UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Trabajo Fin de Máster

Ingeniería y Ciencia de Datos

Modelo de Geolocalización de Objetos Móviles en la nube

María del Carmen Sierra Fernández

Dirigido por: Rafael Pastor Vargas

Curso: 2021-2022: Convocatoria Junio

Agradecimientos

A mi tutor Rafael Pastor Vargas, por su paciencia y por acompañarme en esta larga aventura llena de dificultades, pero sin duda, nos ha dejado un gran aprendizaje. Gracias por su confianza y apoyo incondicional.

A la empresa Wave Location Technolgies por permitirnos el acceso a la información, en especial a Grzegorz Adam Hankiewicz por su paciencia, disposición y por compartir su conocimiento.

A mis padres quienes me enseñaron que el saber no ocupa lugar y que las cosas en la vida se logran con esfuerzo y trabajo.

A Danilo por apoyarme siempre en los momentos difíciles y en mis aventuras. En especial en esta decisión de realizar el Master, que surge en un momento laboral y económicamente difícil.

A mis hijos Gus y Dani quienes me han apoyado y acompañado durante este recorrido, animándome para seguir estudiando “a pesar de mi edad” como siempre me dicen de manera graciosa. Gracias, por esos abrazos mientras estudiaba frente al ordenador. Vosotros sois el impulso que me ayuda a levantarme y seguir adelante ante las dificultades.

A mis compañeros de Master de la UNED con quienes formamos un gran equipo. Y en especial a mi compañero José Alberto Benítez por su apoyo incondicional y paciencia infinita, gracias por compartir tu conocimientos.

Resumen

Los avances tecnológicos nos permiten disponer de sistemas de posicionamiento global (GPS) en la mayoría de los dispositivos que utilizamos a diario (móviles, relojes inteligentes, tabletas, etc.). Cada vez son más las aplicaciones que incorporan funcionalidades de geolocalización como por ejemplo: compartir una ubicación en tiempo real, localizar un restaurante o amigos que están cerca de ti. Existen otras aplicaciones como **AlertCops** o **Sister** orientadas a proteger a personas vulnerables como: personas mayores, personas víctimas de violencia de género o doméstica o personas con capacidad reducida que necesitan ser ubicadas en cualquier momento o que puedan emitir una señal de alarma, cuando tenga una percepción de inseguridad.

El objetivo de este trabajo de Fin de Master es implementar un modelo de datos de geoposicionamiento utilizando los servicios del proveedor en la nube **Amazon Web Services (AWS)**, a partir de los datos generados por la aplicación Sister de Wave Location Technologies. Desplegaremos un proceso de extracción de datos utilizando contendores **Docker** y que ejecutaremos en AWS utilizando los servicios de **Amazon Elastic Container Registry (AWS ECR)** y **Amazon Elastic Container Service (AWS ECS).** Los datos generado serán almacenaremos en una Base de Datos documental MongoDB creada en una instancia de **Amazon EC2**.

Una vez obtenidos los datos, realizaremos un análisis exploratorio, a fin de descubrir patrones o tendencias, identificar anomalías, comprobar relaciones entre las variables o eliminar variables que no agreguen valor al modelo. Para finalizar aplicaremos algoritmos de clustering o agrupamiento para analizar como estos aglomeran los datos de geolocalización.

Abstract

The new technologies help us to use the global positioning systems (GPS) on common device such as cellphone, smart watch, tablets, etc. Every time more applications adding a GPS feature to allow the people share their positioning, find a restaurant or near friends. There are other applications as AlertCops or Sister focusing to protect vulnerable people such as old, gender violence victims or disability people, that they need to be find trough turn on an alarm signs when they feel not secure on any place.

The goal of this senior project is deploying a geolocation data model using the Amazon Web Services. The data is going to be generated by the app named Sister Wave Location Technologies. We are going to run a Docker's containers, running Amazon Elastic Container Registry to collect information, after that they will be storage on a MongoDB database.

The data collected will be comprehensive reviewed in order to find patterns, trends, identify anomalies, review the variables behavior to take a decision to keep or remove them. After that we will to apply a cluster's algorithm to analyze how the information were organized.

Palabras claves

Geolocalización

Análisis exploratorio de datos (EDA)

Extracción de datos

Aprendizaje automático

Visualización de datos

Machine Learning

Funciones Lambda

Proceso batch

AWS

Clustering o Agrupación

Servicios en la nube

KMeans

DBSCAN

Python

Docker

Índice general

Índice de figuras

Índice de Tablas

1. Introducción General y objetivos

1.1 Introducción

Según la Ley Orgánica 1/2004, de medidas de protección integral contra la violencia de género, señala que la violencia de género es la que se produce como manifestación de la discriminación, la situación de desigualdad y las relaciones de poder de los hombres sobre las mujeres, y se ejerce por parte de quienes son y han sido sus cónyuges o de quienes estén o hayan estado ligados a ellas por relaciones similares de afectividad, aún sin convivencia. Se entiende entonces, por violencia de género todo acto de violencia, física o psicológica, incluidas las agresiones a la libertad sexual, las amenazas, las coacciones o la privacidad arbitraria de libertad. (Instituto Nacional de Estadistica (INE), 2022)

En nota de prensa publicada por el Instituto Nacional de Estadística (INE) el 11 de mayo del 2021, se señala que en el año 2020 se inscribieron como víctimas de violencia de género y violencia doméstica 37.819 personas, un 4,9% menos que en 2019. De las cuales, 34.446 fueron mujeres y 3.373 hombres. Aun cuando, el número de mujeres víctimas de violencia de género disminuyó un 8,4% en el año 2020, hasta 29.215. La tasa de víctimas de violencia de género fue de 1,4 por cada 1.000 mujeres de 14 y más años. Por otra parte, el número de víctimas de violencia doméstica creció en 2020 un 8,2% con 8.279 víctimas. (Instituto Nacional de Estadística - EVDVG, 2020)

Sin duda, la manifestación más extrema de la violencia de género, son las mujeres que mueren a manos de sus parejas o exparejas. El Instituto Nacional de Estadística, señala que en el año 2021 murieron 43 mujeres a manos de sus parejas o exparejas, aun cuando es una cifra menor a la del año 2020, en el que fallecieron 47 mujeres. (Instituto Nacional de Estadistica (INE), 2022)

Una encuesta realizada por Sister[1] a través de Google Forms en febrero del 2022 a 35.014 mujeres en Madrid, señala que el 83% de las mujeres encuestada *"…siente miedo de regresar al casa de noche en solitario, tras volver de una cena o de una fiesta con amigos y familiares"*. El estudio señala además, que algunas de las situaciones donde las españolas sienten más miedo

[1] Sister es un proyecto liderado por mujeres como parte de la startup española Wave Location Technologies, líder en geolocalización privada en tiempo real, que cuenta con otras apps como Wave, Wola y Wola Schools.

son atravesando zonas oscuras (26.576 votos), al llegar al portal de su casa (11.919 votos) y cuando va sola por la calle (10.905 votos). En este mismo estudio, menciona que los mecanismos por los cuales optan las mujeres contra el miedo son: evitar zonas oscuras, tomar rutas alternas más seguras, refugiarse en establecimientos abiertos o confiar en amigos o pareja para que las acompañen virtualmente a casa, compartiendo su ubicación. (Mujeres de la Ciudad de Madrid, 2022)

Por otra parte, los avances tecnológicos nos permiten disponer de sistemas de posicionamiento global (GPS) en nuestros dispositivos móviles, tabletas y relojes inteligentes (smart watch). Cada vez son más las aplicaciones que incorporar funcionalidades que nos permiten compartir nuestra ubicación en tiempo real. También existen aplicaciones de seguridad, que están orientadas a proteger a personas vulnerables como personas mayores, personas víctimas de violencia de género o doméstica o personas con capacidad reducida. De manera que puedan ser ubicadas en cualquier momento o que puedan emitir una señal de alarma o alerta cuando se sientan vulnerables. En España, una de las aplicaciones más conocida es **AlertCops,** es una app diseñada por la Policía Nacional para comunicar de forma virtual cualquier tipo de incidencia a la policía. **Sister** es una aplicación desarrollada en España por Wave Location Technologies, dirigida a mujeres en situación de riesgo. Sister no es sólo una aplicación, es un proyecto que lucha contra el acoso sexual y trabaja con ONGs internacionales (Humanas, Fundación Ana Bella, Soy un mujerón, Cruces x Rosas, Back Home, Brujas feministas, Mujeres transformadas, entre otras) para mejorar la seguridad de niñas y mujeres de todo el mundo.

Esta aplicación genera miles de datos de geolocalización que resultan interesantes analizar aplicando algoritmos de Aprendizaje Automático que nos permitan a los especialista en datos identificar patrones y crear modelos previamente entrenados, que al introducir nuevos datos puedan predecir o identificar zonas donde otras mujeres experimentaron una percepción de inseguridad.

Nuestra motivación surge del trabajo de fin de grado de la UNED **"Geoposicionamiento de objetos móviles mediante plataformas GIS en la nube"** elaborado por José Luis García Fuentes, donde realiza un análisis exploratorio de los datos e implementa algunas pruebas de concepto.

El objetivo de este trabajo de fin de Master es implementar un modelo de datos de geoposicionamiento en Amazon Web Services (AWS), a partir de los datos generados por la aplicación Sister de Wave Location Technologies. Nos enfocaremos en la implementación del

de los procesos de extracción, depuración y transformación de los datos, con la finalidad de mejorar la calidad de estos.

El proceso de extracción de datos es la clave de este trabajo, lo dividiremos en dos (2) subprocesos: el primero se encargará de realizar la extracción masiva de los datos (proceso batch) y el segundo extraerá datos nuevos, que serán el insumo de cualquier modelo de aprendizaje previamente entrenado. Para la ejecución de los procesos de extracción de datos utilizaremos los diferentes servicios que nos ofrece el proveedor Amazon Web Services (AWS), por ser este aliado tecnológico de la empresa Wave Location Technologies.

Una vez obtenidos los datos realizaremos un análisis exploratorio, a fin de descubrir patrones o tendencias, identificar valores atípicos o anomalías, identificar relaciones entre las variables o eliminar variables que no agreguen valor al modelo. Desarrollaremos además, procesos que nos ayuden a optimizar el espacio a través de la conversión de los tipos de datos y limpieza.

Una vez mejorada la calidad de los datos, podrán ser utilizados para entrenar y evaluar diferentes modelos de aprendizaje automático. Como parte del alcance de este trabajo aplicaremos algoritmos de agrupación o clustering, de manera de analizar como estos agrupan los datos de geolocalización generados por los usuarios a través de la aplicación SIster.

Durante el desarrollo de este trabajo intentaremos responder a preguntas como: **¿Los procesos de extracción de datos creados generan datos de mejor calidad?, ¿Ejecutar los procesos de extracción de datos en la nube reduce los tiempos de ejecución?, ¿Cuáles son las principales dificultades que se presentan al momento de utilizar servicios en la nube?, ¿Es posible determinar el número óptimo de clústeres? ¿Qué información podemos obtener de las métricas de evaluación interna?**.

1.2 Objetivos

1.2.1 Objetivo General

- Implementar un modelo de datos de geolocalización en Amazon Web Services (AWS), a partir de los datos generados por la aplicación Sister de Wave Location Technologies.

1.2.2 Objetivos específicos

- Desplegar los procesos de extracción masiva de datos de geolocalización en Amazon Web Services (AWS).
- Evaluar la calidad de los datos obtenidos en el proceso de extracción, a través de un análisis exploratorio.
- Aplicar algoritmos de agrupación al conjunto de datos de geolocalización de la Comunidad Autónoma de Madrid.

2. Estado del arte

A continuación materializaremos la investigación documental realizada sobre las áreas de investigación para la construcción del conocimiento y lograr una reflexión sobre las tendencias, técnicas y/o tecnologías a aplicar en nuestro trabajo. Es importante señalar que muchas tecnologías utilizadas están supeditadas porque son utilizadas actualmente por la empresa Wave Location Technologies, como es el caso de la Base de Datos Firestore y del proveedor de servicio en la nube AWS Amazon Web Services.

2.1 Geolocalización

"La geolocalización es la capacidad para obtener la ubicación geográfica real de un objeto, como un radar, un teléfono móvil o un ordenador conectado a Internet. La geolocalización puede referirse a la consulta de la ubicación, o bien para la consulta real de la ubicación. El término geolocalización está estrechamente relacionado con el uso de sistemas de posicionamiento, pero puede distinguirse de estos por un mayor énfasis en la determinación de una posición significativa (por ejemplo, una dirección de una calle) y no solo por un conjunto de coordenadas geográficas…" Este proceso es utilizado por los sistemas de información geográfica o aplicaciones diseñadas para capturar, almacenar, manipular y analizar la información geográfica. (Wikipedia - Geolocalización, 2022)

La aplicación Sister cuenta con dos modos de usos prevención y emergencia, el primero muestra al usuario una mapa con la ruta más rápida para llegar a casa y permite compartir en tiempo real su ubicación con sus contactos de confianza. El segundo modo, envía la posición del usuario en tiempo real a sus contactos, avisándoles que se encuentra en peligro. Cuando se activa el modo emergencia la aplicación comienza a registrar las geolocalizaciones del usuario en la base de datos hasta que la alarma se desactive. Es así, como se generan los datos que utilizaremos para la implementación de nuestro modelo de datos.

Considerando que el volumen de datos en el entorno de producción supera los 2,6 millones de documentos y que es una aplicación disponible a nivel mundial, se hace necesario acotar nuestro ámbito de estudio, por ello hemos seleccionado la Comunidad Autónoma de Madrid. Sin embargo, todos los procesos que desarrollaremos serán parametrizables, de manera que sea posible realizar extracciones de datos de otras ciudades.

Una vez tomada la decisión de limitar los datos, se nos presenta la primera línea de investigación, porque del proceso de extracción de datos obtenemos coordenadas (latitud y longitud) que nos permiten ubicar geográficamente ese punto, sin embargo, necesitamos identificar si esas coordenadas pertenecen o no a la Comunidad Autónoma de Madrid. Tomando en cuentas que las soluciones se desarrollaran utilizando el lenguaje de programación Python, nuestra búsqueda se centró en librerías disponibles en Python, para acceder a servicios de geocodificación. Una de las librerías encontradas en nuestra búsqueda es Geopy. Esta facilita la localización de direcciones, ciudades, países y puntos de referencia en todo el mundo utilizando geocodificadores de terceros y otras fuentes de datos, entre los que se encuentra OpenStreetMap Nominatim. (Geopy, 2022)

Nominatim es una herramienta de OpenStreetMap[2] (OSM) para la búsqueda de datos por nombre y dirección (geocodificación), también nos permite generar direcciones a partir de coordenadas, lo que se conoce como geocodificación inversa. (Nominatim, 2022) La funcionalidad de geocodificación inversa, nos permitirá obtener una dirección completa a partir de las coordenadas latitud y longitud. De esta dirección extraeremos la ciudad y seleccionaremos sólo aquellos datos que corresponda a la Comunidad Autónoma de Madrid.

2.2 Visualización de datos de geolocalización

Cuando hablamos de latitud y longitud sabemos que nos referimos a dos (2) coordenadas que nos permiten ubicar una posición o lugar geográficamente. Cuando estas coordenadas las dibujamos en un mapa, esa ubicación geográfica nos brinda una serie de información adicional, como nombre de calles, negocios cercanos, distancia desde el punto donde nos encontramos, entre otros datos.

Cuando iniciamos esta investigación, donde trabajaremos con información de geolocalización (latitud, longitud, altitud, radio); se hizo necesario investigar sobre las diferentes librería Python disponibles para el análisis geoespacial. Una de la librerías más recomendadas por los programadores Python es Folium.

Folium es una librería que nos ayuda a visualizar datos geoespaciales, además de crear mapas interactivos, en los cuales podemos agregar funcionalidades de clustering o mostrar información adicional sobre el punto o ubicación geoespacial marcado en el mapa. Posee una colección de mapas de OpenStreetMap, con diferentes características, por ejemplo mapas de relieve o de

[2] OpenStreetMap (también conocido como OSM) es un proyecto colaborativo para crear mapas editables y libres. (OpenStreetMap - Wikipedia, la enciclopedia libre)

carreteras. Este mapa o mosaico compone la primera capa del mapa en Folium, sobre la cual podemos agregar elementos gráficos (íconos o imágenes) que nos permiten marcar una ubicación en un mapa (latitud, longitud, radio). A cada marcador podemos configurarle una serie de parámetros, como por ejemplo popup; este parámetro nos permite configurar una ventana emergente que muestra información adicional sobre el punto marcado en el mapa.

Folium también nos permita crear rutas o trayectorias sobre el mapa, que serán muy útiles para la visualización de los recorridas de nuestros usuarios en la Comunidad Autónoma de Madrid. (Folium - Documentation, 2022)

2.3 Servicios en la nube

Los servicios en la nube se pueden definir como el suministro o la distribución de recursos de TI (servidores, almacenamiento, base de datos, software, modelos de Machine Learning previamente entrenados, etc.) bajo demanda a través de Internet mediante un esquema de pago por uso.

Los proveedores de servicios en la nube son empresas que ofrecen productos tales como: Infraestructuras como Servicio (IaaS), Plataformas como Servicio (PaaS) y Software como Servicio (SaaS). Existen diferentes proveedores de servicios en la nube, tales como: Amazon Web Services (AWS), Google Cloud Platform (GCP), IBM Cloud, Oracle Cloud, Microsoft Azure, entre otros. Para nuestro estudio nos enfocaremos en los servicios que ofrece el proveedor de servicios Amazon, porque este es el principal aliado tecnológico de la empresa Wave Location Technologies.

Amazon Web Services (AWS): es una plataforma en la nube, que ofrece más de 200 servicios, que van desde tecnologías de infraestructura como servicio, almacenamiento y bases de datos hasta tecnologías emergentes como Machine Learning e inteligencia artificial, lagos de datos[3] y análisis e Internet de las cosas IoT. (AWS Amazon Web Services , 2022)

Amazon Simple Storage Service (Amazon S3): es un servicio de almacenamiento de objetos que permite almacenar y recuperar cualquier volumen de datos desde cualquier ubicación. Ofrece escalabilidad, disponibilidad de datos, seguridad y rendimiento. (AWS Amazon S3, 2022)

AWS Batch: es un servicio totalmente gestionado que permite planificar, programar y ejecutar cargas de trabajo por lotes. **AWS Batch** se integra de manera nativa con la plataforma de AWS,

[3] Un lago de datos es un repositorio de almacenamiento que contiene una gran cantidad de datos en su formato nativo y sin procesar (Lagos de datos - Azure Architecture Center | Microsoft Docs)

lo que le permite aprovechar las capacidades de escalado, redes y administración del acceso de AWS. Además de facilitar los procesos de lectura y escritura de datos de manera segura en almacenes de datos de AWS, como Amazon S3 o Amazon DynamoDB, y desde ellos. (AWS Amazon Batch, 2022)

Amazon Elastic Container Registry (Amazon ECR): es un servicio de registro de contenedores completamente administrado por AWS seguro, escalable y fiable. Amazon ECR permite trabajar con repositorios privados con permisos basados en AWS Identity Access Management (IAM) IAM. Algunos componentes de Amazon ECR son: (AWS Amazon ECR, 2022)

- **Registry:** Es un registro privado de Amazon ECR; donde se pueden crear repositorios y almacenar imágenes. Para ello, se requiere un **token de autorización o autenticación** y controlar los acceso a través de **políticas de acceso**.
- **Repositorio:** Un repositorio de Amazon ECR contiene las imágenes de **Docker**, **Open Container Initiative (OCI)** y artefactos compatibles con **OCI**.
- **Imagen:** En los repositorios creados, podemos insertar y extraer imágenes de un contenedor, utilizarlas localmente o en tareas de Amazon ECS y especificaciones del pod[4] de Amazon EKS.

Amazon Elastic Container Service (Amazon ECS): es un servicio de gestión de contenedores escalable y de alto rendimiento que permite contenedores Docker y ejecutar aplicaciones en un clúster que puede contener más de una instancia Amazon Elastic Compute Cloud (Amazon EC2).

Con Amazon ECS no es necesario instalar infraestructura de administración de clústeres. Con una llamada a la API, se pueden ejecutar y detener aplicaciones en contenedores, además de realizar tareas de monitoreo y seguridad del clúster. (AWS Amazon ECS, 2022)

AWS Fargate: es un servicio informático sin servidor, de pago por uso que permite implementar y administrar aplicaciones sin tener que administrar los servidores (instalación de parches, seguridad). **AWS Fargate** es compatible con Amazon Elastic Container Service (ECS) y Amazon Elastic Kubernetes Service (EKS). (AWS Amazon Fargate, 2022)

[4] Un pod de Kubernetes es un conjunto de uno o varios contenedores de Linux ® y constituye la unidad más pequeña de las aplicaciones de Kubernetes. (¿Qué es un pod de Kubernetes? (redhat.com))

Elastic Kubernetes Service (EKS): Es un servicio de contendores administrado que nos permite ejecutar aplicaciones Kurbentes [5] en la nube, específicamente en AWS. Amazon EKS administra de forma automática la disponibilidad y la escalabilidad de los nodos de Kubernetes.

Amazon EKS nos permite ejecutar las aplicaciones de Kubernetes tanto en Amazon Elastic Compute Cloud (Amazon EC2) como en AWS Fargate (Ver Figura 1). Aprovechando el rendimiento, la escala, la fiabilidad y la disponibilidad de la infraestructura de AWS, así como las integraciones con las redes y los servicios de seguridad de AWS (balanceo de carga, integración con IAM y soporte de AWS Virtual Private Cloud (VPC)). (AWS Amazon EKS, 2022)

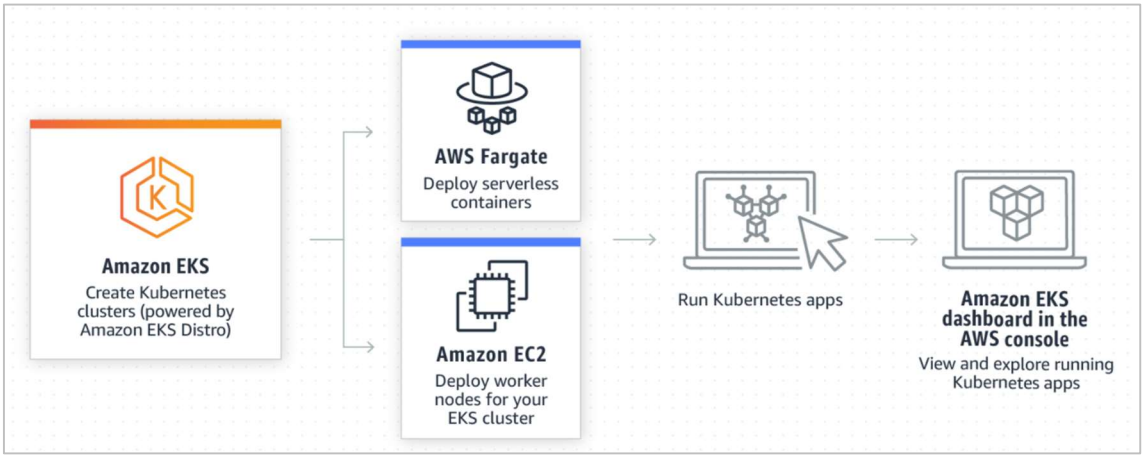


Figura 1: Modelo para implementar aplicaciones en contenedores basado en : Amazon Elastic Kubernetes Service (Amazon EKS)

AWS Lambda: Es un servicio que nos permite ejecutar código sin aprovisionar o administrar servidores. Es decir, nos permite ejecutar código en una infraestructura de alta disponibilidad, mientras que las tareas de administración, mantenimiento, aprovisionamiento, escalado y monitoreo son realizadas por el proveedor AWS.

Las funciones Lambda podemos invocarlas utilizando la API de Lambda o como respuesta a eventos de otros servicios de AWS, por ejemplo al escribir o subir un fichero en un bucket de AWS S3. Las funciones Lambda son una excelente opción para cargas de trabajo cortas y basadas en eventos, ya que las funciones de Lambda tienen un tiempo máximo de ejecución de 15 minutos por invocación. Aun cuando esto limitación de tiempo puede ser considerada como un inconveniente, podemos convertirlo en una ventaja si creamos funciones más pequeñas y que estas invoquen a otras funciones. (AWS Amazon Lambda, 2022)

[5] Kubernetes es un software de orquestación de código abierto para implementar, administrar y escalar contenedores. (¿Qué es Kubernetes? | Microsoft Azure)

AWS Identity and Access Management (IAM): es un servicio que nos permite controlar de forma segura el acceso a los recursos de AWS. IAM nos permite controlar quién está autenticado (ha iniciado sesión) y autorizado (tiene permisos) para utilizar los recursos.

Podemos utilizar IAM para proporcionar credenciales para las aplicaciones que se ejecutan en instancias EC2. Estas credenciales proporcionan permisos a la aplicación para obtener acceso a otros recursos de AWS, por ejemplo a un bucket de S3 o a una base de datos. IAM nos permite además, utilizar autenticación en dos factores para mayor seguridad.

Una forma de administrar o gestionar los acceso en AWS es creando políticas y asignándoselas a identidades de IAM (usuarios, grupos o roles) o a recursos de AWS. Una política es un objeto de AWS que al asociarlo a una identidad o un recurso, define sus permisos. Estos permisos determinan si la solicitud se permite o se deniega. Las mayoría de las políticas en AWS, se almacenan como un documento JSON[6]. (AWS Amazon IAM, 2022)

Amazon SageMaker: Es un servicio administrado de Machine Learning, que nos permite crear y entrenar modelos de aprendizaje automático e implementarlos, a través de servicios como Amazon SageMaker Neo.

Amazon SageMaker incluye una instancia de Jupyter integrada, donde podemos crear Notebook para acceder a nuestros orígenes de datos y realizar estudios exploratorio de datos, entrenar modelos de ML e implementarlos. También ofrece algoritmos de aprendizaje automático optimizados para ejecutarse en conjuntos de datos grandes en entornos distribuidos.

La Figura 1 muestra el flujo de trabajo para la creación de una modelo de aprendizaje automático. Este flujo parte de la generación de los datos para realizar el entrenamiento de un modelo. Este proceso conlleva la obtención, limpieza y la transformación de los datos. Una vez los datos están listos, el siguiente paso es el entrenamiento del modelo que incluye además la evaluación para determinar si la precisión es aceptable. Una vez tenemos nuestro modelo previamente entrenado el siguiente paso es implementarlo.

Como el aprendizaje automático es un proceso continuo una vez implementado debemos evaluar el modelo para identificar desviaciones, para ello aplicamos el modelo previamente entrenado a nuevos datos, si identificamos desviaciones, podemos volver a entrenar el modelo con el nuevo conjunto de datos. (AWS Amazon SageMaker, 2022)

[6] JSON (JavaScript Object Notation - Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos. Es una colección de pares de nombre/valor. (JSON)

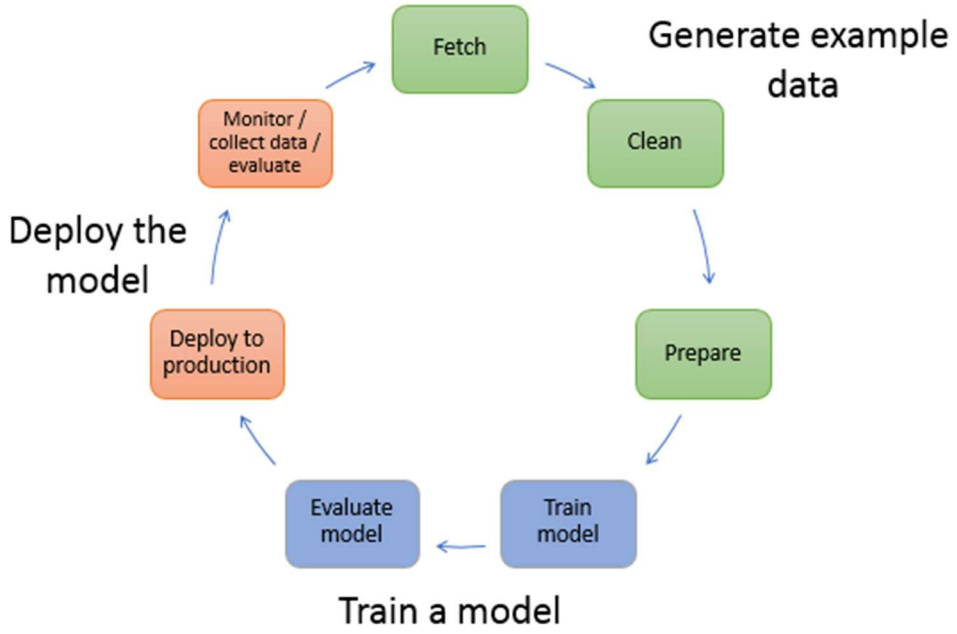


Figura 2: Flujo de trabajo típico para la creación de un modelo de aprendizaje automático. Fuente: Machine Learning con Amazon SageMaker - Amazon SageMaker

2.4 Extracción de datos

Los datos generados por la aplicación Sister son almacenados en Cloud Firestore una base de datos NoSQL orientada a documentos con una estructura jerárquica compleja. De estos datos nos interesan únicamente los datos de geolocalización generados a partir del momento que se active una alerta o alarma por parte de una usuario. El siguiente desafío es diseñar un proceso que permita extraer los datos de forma masiva o por lotes, cuando hablamos de procesamiento por lotes lo primero que nos viene a la cabeza son procesos batch. Este será entonces nuestro siguiente desafío, considerando que la empresa Wave Location Technologies tiene como aliado tecnológico a Amazon AWS, nos centraremos en los servicios que este proveedor de computación en la nube para diseñar y ejecutar proceso batch.

El proceso de extracción de los datos esta divido en dos subprocesos: el primero tiene como objetivo realizar una extracción masiva de los datos, que serán utilizados en el análisis exploratorio y para el entrenamiento del modelo de Machine Learning seleccionado. El segundo subproceso, tendrá como objetivo la extracción de los datos, a partir de la fecha de la primera extracción. Este proceso continuo, será el responsable de extraer nuevos datos que servirá de ingesta a modelo de Aprendizaje Automático previamente entrenados.

Para el primer proceso de extracción de datos masivos, Amazon AWS ofrece un servicio llamado AWS Batch que permite la ejecución de trabajos por lotes, con este servicio no necesitamos instalar ni gestionar ni servidores ni software, porque es un servicio completamente gestionado. Con AWS Batch sólo necesitamos empaquetar el código, especificar las dependencias y ejecutar el trabajo por lotes. Sin dudarlo, este fue el servicio que seleccionamos para desarrollar y ejecutar el proceso para la extracción masiva de datos. Durante la implementación del proceso en AWS, nos encontramos con un inconveniente con la instalación de la librería firebase, seguimos diferentes vías para subir las librerías en ficheros desde S3, pero limitaciones del tamaño o la forma de instalación de esta librería en particular, nos obligó a cambiar el diseño, una de las soluciones a nuestro problema era desplegar la aplicación dentro de un contenedor.

Pasamos entonces, de una solución pensada utilizando el servicios AWS Batch a implementar una aplicación dentro de un contenedor Docker. Después de analizar algunos de los servicios disponibles en Amazon AWS, una alternativa es utilizar los servicios: **Amazon Elastic Container Registry (Amazon ECR), Amazon Elastic Container Service (Amazon ECS). El primero nos** permite crear y acceder al repositorio con la imagen de la aplicación, mientras que el segundo nos permite gestionar e implementar el contenedor con la imagen de la aplicación. Una vez nuestro contendor esté disponible en Amazon ECS, podemos ejecutar la aplicación dentro del contenedor definiendo una tarea utilizando cualquiera tres(3) tipo de lanzamiento disponibles: AWS Fargate, EC2 o External.

Para implementar el segundo proceso de extracción de datos nuevos, que alimentará al modelo de ML previamente entrenado, utilizaremos el servicio AWS Lambda. En cada función Lambda crearemos una capa o layer[7] con todas las librerías necesarias para la ejecución de las funciones. Debido a que el tiempo de ejecución máximo de una función lambda es de 15 minutos, hemos dividido el proceso de extracción en dos funciones lambda: La primera función extraerá el identificador del usuario y los identificadores de monitoreo que se crear cada vez que un usuario activa un alerta o alarma. La segunda función lambda tomará estos datos y extraerá de la base de datos todas las localizaciones o geolocalizaciones registradas en cada identificador de monitoreo (`monitoring_id`).

Los datos generados por ambos procesos de extracción de datos serán almacenados en ficheros CSV o en una base de datos NoSQL orientada a documentos MongoDB creada en una

[7] Una capa de Lambda es un archivo .zip que puede contener código u otros datos adicionales. Una capa puede contener bibliotecas, un tiempo de ejecución personalizado, datos o archivos de configuración. (Conceptos de Lambda - AWS Lambda (amazon.com))

instancia de EC2. El tipo de almacenamiento de los datos se definirá a través del parámetro de tipo booleano `SAVE_DDBB`.

2.5 Docker

"*Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos…".* (Wikipedia - Docker, 2022) El uso de contenedores adiciona a nuestra aplicación flexibilidad y portabilidad de ejecutar la aplicación localmente, en la nube pública, privada o hibrida.

Con Docker no necesitamos un sistema operativo como intermediario para ejecutar la aplicación. Este utiliza los denominados contenedores, un contendor es un programa, servicio o aplicación que queremos desplegar, como no utiliza máquinas virtuales el rendimiento es mayor y el consumo de recursos es menor, esto se debe a que utiliza un kernel de Linux que permite aislar cada contenedor.

Un contendor es una instancia de una imagen y una imagen es una plantilla donde se define el sistema operativo origen, los programas, librerías u otras dependencias que necesitamos para ejecutar nuestra aplicación o servicio. Las imágenes Docker están disponibles en un repositorio llamado Docker Hub. En este repositorio podemos encontrar una gran cantidad de imágenes con el software más popular y listo para ser ejecutado. (Durán, 2021)

Docker es una tecnología muy utilizada por los principales proveedores de servicios o computación en la nube, por ser una herramienta transversal. Además porque ofrece ventajas tales como:

1. Facilitar la instalación y configuración del entorno de desarrollo, también nos ayuda a identificar las dependencias necesarias
2. Realizar implementaciones en poco tiempo y el control sobre las versiones y su distribución.
3. Garantizar que la aplicación en producción dispone de todas las librerías y dependencias que necesita para su ejecución
4. Mejorar y agilizar los procesos de despliegue continuo, logrando que ejecuten de la misma manera en todos sus entornos (desarrollo, pruebas, preproducción y producción).

En nuestro trabajo el uso de contendores Docker fue la solución encontrada al problema derivado de la forma de instalación de las librerías de Firebase en el entorno Python de Amazon Batch, que explicaremos con más detalle más adelante.

Otra alternativa, que nos permite crear una aplicación en un contendor es: Amazon Elastic Kubernetes Service (Amazon EKS). Una alternativa interesante para evaluar y comparar por ejemplo la facilidad de implementación y el rendimiento de nuestra aplicación utilizando contenedores Kubernetes en comparación con contenedores Docker.

2.6 Base de Datos NoSQL

Cuando hablamos de base de datos, lo primero que nos viene a la cabeza son las bases de datos relacionales o SQL. Esto se debe a que el uso de las bases de datos relacionales está muy extendido y seguramente nos ofrecen un mayor soporte. El surgimiento de nuevas tecnologías, el crecimiento en el volumen de los datos y la evolución de las aplicaciones para entornos web, dispositivos móviles y otros dispositivos inteligentes, nos hizo entender que aun cuando las bases de datos SQL son excelentes para garantizar la integridad y la consistencia de los datos, nos limitan por su baja escalabilidad, la complejidad para crecer horizontalmente y su baja flexibilidad porque trabaja con datos estructurados.

Las base de datos NoSQL aparecen como consecuencia de la necesidad de almacenar información no estructurada como documentos, email, SMS, geolocalizaciones, audios, videos y otros tipos de información. Por otra parte, los grandes volumen de datos conocido como Big Data, crean nuevas necesidades en velocidad de procesamiento, alta escalabilidad e implementación de estructuras distribuidas.

La Tabla 1, recopila algunas de las diferencias que existen entre las base de datos relacionales (SQL) y las base de datos NoSQL. Las base de datos SQL son excelentes candidatas cuando la consistencia de los datos es la prioridad, mientras que las base de datos NoSQL son útiles cuando el volumen de datos crece rápidamente; cuando necesitamos una gran escalabilidad; o, cuando el esquema de la base de datos no es homogéneo.

Base de datos relacional	NoSQL
Almacena datos en forma tabular (filas y columnas)	Almacenan datos de varias formas: key-value, documento, basado en columnas o en grafos
Esquema predefinido	Esquema dinámico
Escalable verticalmente	Escalable horizontalmente

Base de datos relacional	NoSQL
SQL soporta consultas complejas y anidadas	Soportan las operaciones CRUD básicas: Trasladan la complejidad a la aplicación. No soportan joins.
Son ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad)	Son BASE (Basically Available Soft-state Eventually consistency).
Se utilizan en sistemas transaccionales (Banca, compras)	Se utilizan en servicios Web2.0 (redes sociales, blogs, etc.), aplicaciones IoT, almacenamiento de perfiles sociales entre otras

Tabla 1: Diferencias entre base de datos relacionales (SQL) y base de datos NoSQL

Las base de datos NoSQL nos permiten almacenar y recuperar diferentes tipos de formatos, por ellos existen diferentes tipos: Clave-Valor (Redis), Grafos (Neo4j), Columnas (Cassandra) y documentos (Cloud Firestore, MongoDB). Para este trabajo, trabajaremos con las base de datos orientadas a documentos, porque nuestro origen de datos se encuentra en Cloud Firestore.

Una base de datos orientada a documentos reemplaza el concepto de *fila* utilizado en las base de datos relacionales, por un modelo más flexible, el *documento*. El enfoque orientado a documento permite incluir documentos incrustados y arreglos, lo que hace posible representar relaciones jerárquicas complejas con un solo registro.

En las base de datos orientadas a documentos, no existen esquemas predefinidos: las claves y los valores de un documento no son de tipos o tamaños fijos. Esto permite, que el desarrollo sea más rápido y fácil para realizar pruebas con diferentes modelos. (Shannon Bradshaw, 2019)

Cloud Firestore: es una base de datos NoSQL, orientada a documentos, flexible y escalable para el desarrollo móvil, web y de servidor de Firebase y Google Cloud. Cloud Firestore nos permite mantener los datos sincronizados entre las aplicaciones de los clientes en tiempo real y sin conexión, facilitando la creación de aplicaciones que funcionan independientemente de la latencia de la red o la conectividad a Internet. Los documentos admiten diferentes tipos de datos diferentes, además podemos crear otras colecciones dentro de un documento y crear estructuras de datos jerárquicas que se escalan a medida que crece la base de datos. (Firebase - Cloud Firestore, 2022)

MongoDB: es una base de datos de código abierto, orientada a documento y multiplataforma. Proporciona alto rendimiento, alta disponibilidad y escalabilidad. Combina la capacidad de escalar horizontalmente con funciones como índices secundarios, consultas de rango y expresiones regulares, ordenar, agregaciones e índices geoespaciales. La replicación es otra

características a resaltar, esto significa que proporciona múltiples copias de datos con múltiples servidores. Proporciona tolerancia a fallas, es decir, que si un servidor de base de datos se cae, la aplicación usa otros servidores de base de datos. (Aaron Ploetz, 2018)

2.7 Aprendizaje Automático o Machine Learning (ML)

Machine Learning (ML) se puede definir como la ciencia o arte de programar ordenadores para que puedan aprender de los datos. (Géron, 2019)

El Machine Learning (ML) también puede considerarse un subconjunto de la inteligencia artificial[8] en el que hacemos que una máquina aprenda a través de los datos proporcionados, para luego utilizar lo aprendido para predecir un resultado o generar recomendaciones utilizando información similar. (Singh, 2020)

El Aprendizaje automático es una excelente opción para: (Géron, 2019)

- Problemas en los cuales un enfoque tradicional requiere introducir muchos ajustes o reglas, en estos casos un algoritmo de ML nos permite simplificar el código
- Problemas complejos donde un enfoque tradicional no ofrece una buena solución.
- Entornos fluctuantes, porque un sistema de aprendizaje automático puede adaptarse a nuevos datos.
- Obtener información sobre problemas complejos y grandes cantidades de datos.

Existen diferentes tipos de sistemas de Aprendizaje automático, si consideramos el criterio de clasificación basado en si requieren o no supervisión, podemos clasificándolos como: aprendizaje supervisado, no supervisado, semi supervisado y reforzado.

Aprendizaje supervisado

En el aprendizaje supervisado, es una rama del aprendizaje automático en la que sabemos exactamente lo que sucedió en el pasado y luego tratamos de predecir si se producirá el mismo resultado si se vuelve a producir una situación del pasado. Esto se logra porque el conjunto de datos que alimenta al algoritmo incluye etiquetas y entrenamos el modelo basado en estos datos. Existen dos tareas típicas en aprendizaje supervisado la primera es la clasificación y la segunda es la predicción conocida como regresión. Algunos de los algoritmos de aprendizaje supervisado son:

[8] La inteligencia artificial es el campo de la informática y las matemáticas que intenta imitar el comportamiento humano y tomar decisiones similares a las de los humanos.

- k-Nearest Neighbors
- Regresión Lineal
- Regresión Logistica
- Support Vector Machines (SVMs)
- Árboles de decisión y Random Forests
- Redes neuronales[9]

Aprendizaje no supervisado

En el aprendizaje no supervisado, los datos de entrenamiento no están etiquetados El sistema trata de aprender sin un maestro. El objetivo principal del aprendizaje no supervisado es comprender los patrones presentes dentro de los datos y luego agrupar los datos que siguen un patrón similar. Por lo tanto, podemos tener grupos o clústeres con características similares. Estos son algunos de los algoritmos de aprendizaje no supervisado:

- Agrupación o Clustering
  - K-Means
  - DBSCAN
  - Hierarchical Cluster Analysis (HCA)
- Detectar anomalías
  - One-class SVM
  - Isolation Forest
- Visualización y reducción de la dimensionalidad
  - Principal Component Analysis (PCA)
  - Kernel PCA
  - Locally Linear Embedding (LLE)
  - t-Distributed Stochastic Neighbor Embedding (t-SNE)
- Aprendizaje por reglas de asociación
  - Apriori
  - Eclat

Aprendizaje semi supervisado

El etiquetado de los datos puede llevar mucho tiempo y ser costoso, por ello muchas veces tendremos la mayoría de las instancias sin etiquetar y pocas instancias etiquetadas. Los

[9] Algunas arquitecturas de redes neuronales pueden ser no supervisadas o semisupervisadas

algoritmos que pueden manejar datos parcialmente etiquetados se conocen como algoritmos de aprendizaje semi supervisado.

La mayoría de los algoritmos de aprendizaje semi supervisados son combinaciones de algoritmos supervisados y no supervisados. Por ejemplo, Deep Belief Networks (DBNs) se basan en componentes no supervisados llamados Restricted Boltzmann Machines (RBMs). Los RBM se entrenan secuencialmente sin supervisión, luego todo el sistema se ajusta utilizando técnicas de aprendizaje supervisado.

Aprendizaje reforzado

En el sistema de aprendizaje reforzado, una máquina intenta aprender por sí misma analizando diferentes escenarios. La máquina recibe una recompensa por cada tarea exitosa que realiza y una penalización si no realiza una tarea. La máquina tiene como objetivo conseguir la mayor cantidad de recompensas posible.

Este sistema está conformado por un *agente*, que puede observar el entorno, seleccionar y realizar acciones y obtener recompensas o penalizaciones. El *agente* debe aprender por sí mismo cuál es la mejor estrategia, llamada *política*, para obtener la mayor recompensa con el tiempo. La política define que acción debe seleccionar o escoger el agente en una situación determinada.

2.7.1 Algoritmos de agrupamiento o clustering

Dentro del aprendizaje no supervisado tenemos los algoritmos de agrupamiento o clustering, cuyo objetivo es agrupar instancias similares en clústeres. Estos son una gran herramienta para el análisis de datos, la segmentación de clientes, los sistemas de recomendación, los motores de búsqueda, la segmentación de imágenes, detección de anomalías, la reducción de la dimensionalidad, entre otras.

Como técnica de reducción de dimensionalidad nos permite una vez agrupados los datos, medir el porcentaje de cada instancia con cada clúster, conocido como afinidad (medida que indica cómo encaja una instancia en un clúster). Este valor de afinidad es muy útil en la detección de anomalías o valores atípicos; porque cualquier instancia que tenga una baja afinidad para todos los grupos es probable que sea una anomalía.

Los algoritmos de agrupamiento pueden obtener diferentes tipos de clústeres. Algunos algoritmos buscan instancias centradas alrededor de un punto particular, llamado centroide; otras buscan regiones continuas de instancias densamente empaquetadas: estos grupos pueden formar cualquier forma. Algunos algoritmos son jerárquicos, buscan grupos de grupos.

K-Means

Es un método de agrupación no supervisado, simple capaz de agrupar datos de manera muy rápida y eficiente. Fue propuesto por Stuart Lloyd en los Bell Labs en 1957 como una técnica para la modulación de códigos de pulso, pero no fue publicado hasta 1982.

K-Means es iterativo y basado en prototipos donde todos los puntos de datos se dividen en un número k de clústeres, cada clúster está representado por sus centroides (prototipos). El centroide puede ser una media de todos los puntos de datos de ese clúster o agrupación. Los puntos de datos en un grupo están más cerca de los centroides de ese grupo y existe una gran similitud entre los puntos de datos del grupo (alta similitud intraclase), pero son diferentes a los puntos de datos de otro grupo (baja similitud entre clases). Esta similitud o diferencia se calcula utilizando la distancia entre los puntos. Esta se calcula utilizando la distancia Euclidiana o de correlación (Pearson, Spearman o Kendall).

K-Means es vulnerable a los valores atípicos. A medida que el algoritmo itera a través de los centroides, los valores atípicos impactan en la forma en que se mueven los centroides antes de alcanzar la estabilidad y la convergencia. Además, K-Means tiene problemas para agrupar con precisión los datos cuando los grupos son de diferentes tamaños y densidades. K-Means funciona muy bien en clústeres esféricos. (Géron, 2019)

Como funciona el algoritmo:

1. Elije el número k de clústeres y determina sus centroides
2. Asigna cada punto de datos a su centroide más cercano utilizando la distancia
3. Recalcula los nuevos centroides y nuevamente asigna cada punto a su centroide más cercano
4. Repite los pasos 3 y 4 hasta que los centroides no cambien o no cambie la función de criterio (*J*) :

$$J = \sum_{j=1}^{k} \sum_{i \in C_j} \left\| x_i - c_j \right\|^2$$

donde:

J = Función de criterio
k = número de clústeres
x_i *= punto de datos*
c_j *= centroide del clúster* C_j

El algoritmo K-Means encuentra el número óptimo de clústeres (*k*) mientras minimiza la función de criterio de agrupamiento (*J*). Cada clúster *k* contiene al menos un punto de datos.

En K-Means, es fundamental proporcionar el número del clústeres que se crearan a partir de los datos. Cuando no conocemos la cantidad de grupos o clústeres, necesitamos utilizar algún método que nos permita encontrar la cantidad óptima de clústeres. Elbow es un método que utiliza los valores de la `inercia` obtenidos tras aplicar el algoritmo K-Means a diferentes números de clústeres (desde 1 a N clústeres), siendo la inercia la suma de las distancias al cuadrado de cada punto del clúster a su centroide:

$$Inercia = \sum_{i=1}^{N} \|x_i - \mu\|^2$$

Una vez obtenidos los valores de la inercia, representamos en una gráfica lineal la inercia respecto del número de clústeres. En esta gráfica debemos apreciar un cambio brusco en la evolución de la inercia, la línea tiene una forma similar a la del codo. El punto en el que se observa ese cambio brusco en la inercia nos indicará el número óptimo de clústeres a seleccionar.

Uno de los problemas del algoritmo K-Means es la inicialización aleatoria, la cual se puede minimizar ejecutando repetida veces el algoritmo con distinta condiciones y guardar el mejor resultado, realizando un agrupamiento jerárquico previo para encontrar una inicialización con criterio o definir KMeans++ en el parámetro algoritmo, este nos permite una buena convergencia. (Pandata Web, 2022)

Otros problemas conocidos del algoritmos de K-Means son: (Google Developers, 2021)

1. La dificultad de agrupación de datos de diferentes tamaños y densidades. Porque asume clústeres de forma esféricas o circulares, densidades y tamaños de clústeres similares.
2. La presencia de valores atípicos puede hacer que los centroides sean arrastrados o los valores atípicos pueden quedar en una agrupación, en vez de ignorarse.
3. El escalado de las dimensiones. A medida que aumenta el número de dimensiones, una medida de similitud basada en la distancia converge a un valor constante entre los ejemplos dados. En estos caso se recomienda una reducción de la dimensionalidad (PCA o agrupamiento espectral).

DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

Es un método de agrupación espacial, basado en la densidad que puede ser utilizado para identificar grupos de cualquier forma en un conjunto de datos que contiene ruido o valores atípicos. DBSCAN define los grupos o clústeres como regiones continuas de alta densidad, que están separadas por regiones con menor densidad de puntos de datos. Este algoritmo funciona bien si todos los grupos son lo suficientemente densos y están bien separados por regiones de baja densidad.

Una característica que diferencia a DBSCAN de otros métodos, es su resistencia a los valores atípicos, por esta razón se utiliza en los sistemas de detección de anomalías. (Machine Learning Knowledge, 2021)

No necesita el número de clústeres como parámetro de entrada y produce buenos resultados cuando los grupos o clústeres tienen formas o estructuras complejas, donde algoritmos como K-Means o de agrupamiento jerárquico no son tan buenos. Este método utiliza dos (2) parámetros:

- **Épsilon (eps):** es el radio del círculo alrededor de un punto de datos, de modo que todos los puntos de datos que caen dentro del círculo se consideran vecinos.
- **Puntos mínimos (min_samples):** es el número mínimo de puntos que debe en la región para definir el clúster.

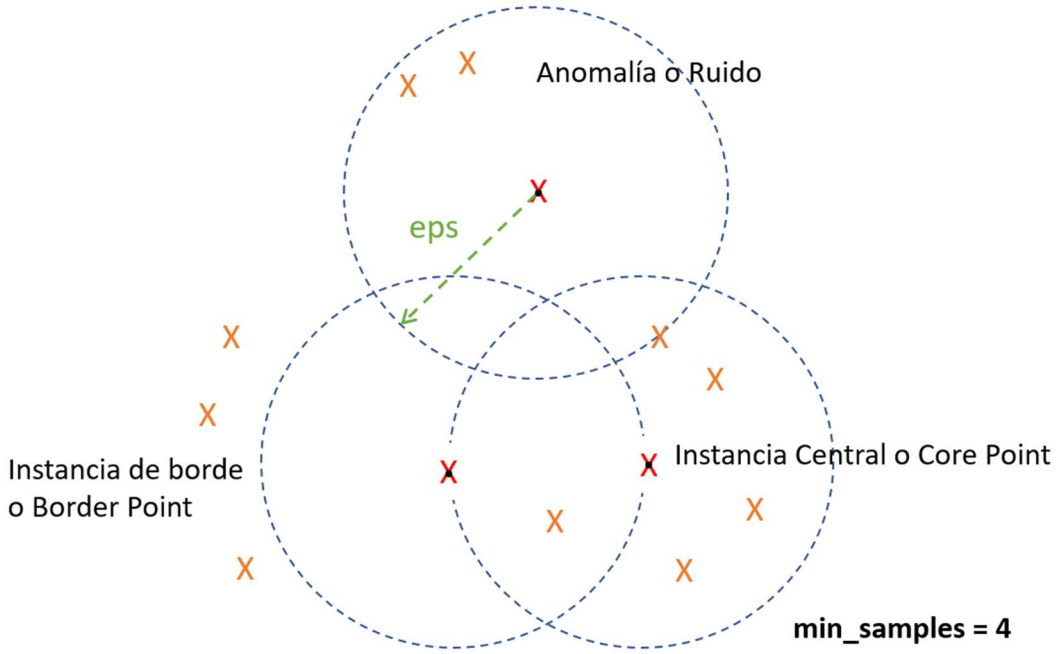


Figura 3: Algoritmo de DBSCAN

Como Funciona

1. Selecciona un punto de partida aleatorio y determina el radio o el vecindario utilizando `épsilon`.
2. Para cada punto, el algoritmo cuenta cuántos puntos o instancias se encuentran dentro de una pequeña distancia ε (épsilon) o vecindario (ε-neighborhood). (Ver Figura 3).
3. Si un punto o instancia tiene al menos `min_samples` instancias en su vecindario, entonces se considera una `instancia central`.
4. El proceso de agrupación comenzará y el punto se marcará como visitado; de lo contrario, este punto se etiquetará como ruido.
5. Todas las instancias cercanas a una instancia central se marcan como parte del grupo. Esto puede incluir otras instancias o puntos centrales, por lo tanto, una secuencia larga de instancias centrales vecinas forma un solo clúster.
6. Cualquier instancia que no sea una instancia central y no tenga una en su vecindario se considera una `anomalía o ruido`. (Géron, 2019)

La calidad de DBSCAN depende de la noción de distancia utilizada en la función $regionQuery(P, \varepsilon)$. La distancia más usada es la `distancia Euclidiana`. Cuando tenemos datos con alta dimensionalidad, es difícil encontrar el valor óptimo de épsilon debido a la llamada "maldición de dimensionalidad". (Wikipedia - DBSCAN, 2022)

Este algoritmo no es determinista, porque los puntos pueden ser alcanzados por diferentes grupos o clústeres, es decir, que una instancia o punto puede ser asignado a un grupo diferente en cada ejecución.

2.8 Métricas de Validación

Una las dificultades de los algoritmos de agrupamiento en el aprendizaje no supervisado, es definir cuando el resultado de un agrupamiento es aceptable. El informe técnico *"Evaluation Metrics for Unsupervised Learning Algorithms"* (Niño & Berzal, Mayo 2019) explica que existen dos (2) tipos de validación: la validación interna y externa.

Los métodos de validación interna nos ayudan a determinar la tendencia de los datos, el número óptimo de clústeres y a evaluar los resultados del clúster. La validación interna nos permite establecer la calidad de la estructura de agrupación, sin tener acceso a información externa. Es decir, se basan en la información proporcionada por los datos utilizados como entrada del algoritmo de agrupación. Mientras, los métodos de validación externa miden la

calidad del agrupamiento conociendo información externa y se utilizan para seleccionar un algoritmo de agrupamiento óptimo sobre un conjunto de datos. (Guzmán, 2019).

Validación Interna

En general, se pueden combinar dos tipos de métricas de validación interna: medidas de *cohesión* (SSW) y *separación* (SSB). La cohesión evalúa qué tan cerca están los elementos del mismo grupo entre sí, mientras que las medidas de separación cuantifican el nivel de separación entre grupos.

Estos métodos se clasifican según el tipo de algoritmo. Para los algoritmos de partición, a menudo se utilizan métricas basadas en la matriz de proximidad, así como métricas de cohesión y separación, como el coeficiente Calinski Harabasz (CH), Davies Bouldin, Hartigan y Silhouette. Para algoritmos jerárquicos, el más común es el coeficiente Cophenetic.

Una agrupación o clúster se considera buena cuando tiene una alta separación entre grupos y un alto cohesión dentro de los grupos. Existen métodos de validación interna que utilizan métricas que cuantifican en una misma medida el nivel de separación y de cohesión como Silhouette, Calisnki-Harabasz (CH), Hartigan, Dunn, entre otros.

El coeficiente Silhouette proporciona un marco simple para calificación. Los valores positivos indican una alta separación entre clústeres. Los valores negativos indican que los grupos se mezclan entre sí (grupos superpuestos). Cuando el coeficiente silueta es cero, indica que los datos son distribuidos uniformemente en todo el espacio euclidiano.

Validación Externa

Los métodos de validación externa no se utilizan en la mayoría de los problemas de clústeres; porque en la práctica, la información externa como los etiquetas de las clases, no se encuentra disponibles.

Para analizar los resultados obtenidos por un algoritmo de agrupamiento, o si necesitamos saber si un agrupación o clúster es aceptable. Debemos construir una *matriz de contingencia*, que contiene cuatro términos:

- **VP (Verdadero Positivo):** hace referencia a aquellos puntos que fueron ubicados por el algoritmo en el mismo clúster que indicaba la clase.
- **FP (Falso Positivo):** hace referencia a aquellos puntos que fueron ubicados por el algoritmo en un clúster `j`, pero pertenecían a otro clúster.

- **FN (Falsos Negativos):** hacen referencia a aquellos elementos del clúster `j` que fueron ubicados en un clúster diferente al que indicaba su etiqueta.
- **VN (Verdadero Negativo):** hace referencia a aquellos elementos que fueron ubicados correctamente fuera del clúster `j`.

Hipótesis\Verdad	P	N
P	VP	FP
N	FN	VN

Tabla 2: Matriz de Contingencias: Las columnas se refiere a las etiquetas de clase conocidas previamente. Las filas corresponden a los resultado del algoritmo.

A partir de esta información se pueden definir varias medidas para medir la similitud entre los grupos o clústeres:

- La **precisión** cuenta los verdaderos positivos (VP), cuántos elementos están debidamente clasificados dentro del mismo grupo.

$$Precisión\ (Pr) = \frac{VP}{VP + FP}$$

- **Recall:** evalúa el porcentaje de elementos que son correctamente incluido en el mismo clúster:

$$Recall\ (R) = \frac{VP}{VP + FN}$$

- La **medida F:** se calcula a partir de la precisión y recall y se puede interpretar como la media harmónica (α) ponderada de ambas

$$F_\alpha = \frac{1 + \alpha}{\frac{1}{Pr} + \frac{1}{R}}$$

Para determinar qué tan bueno fue el agrupamiento realizado, existen otras dos métricas externas llamadas ***Entropía*** y ***Pureza***. La pureza evalúa si cada grupo contiene sólo elementos de la misma clase. Mientras que la entropía es una medida reciproca de pureza que nos permite medir el grado de desorden en los resultados del grupo.

3. Modelo de datos para sistemas de geolocalización

3.1 Origen de los Datos

Sister es un proyecto liderado por mujeres como parte de la startup Wave Location Technologies, trabajan en geolocalización privada en tiempo real, cuenta con otras apps como: Wave, Wola, Fami, Otto, Stay at Home, entre otras. Sister es una aplicación de geolocalización para mujeres, que no solo pretende disminuir la sensación de inseguridad, sino también prevenir posibles amenazas y responder en caso de emergencia. La aplicación cuenta con dos modos de uso:

1. **Prevención:** Este modo utiliza el GPS para mostrar la ruta más rápida para llegar a casa y muestra en el mapa las comisarias más cercanas, también permite a la usuaria compartir en tiempo real su ubicación con sus contactos de confianza.
2. **Emergencia:** Este modo envía la posición de la usaría en tiempo real a sus contactos de confianza notificándoles que se encuentra en peligro. Esta puede además, solicitar ayuda a través de la app, SMS o Whatsapp. (Wave Location Technologies, 2022)

Sister es un aplicación disponible en todo el mundo para dispositivos móviles iOS y Android. Existe una versión gratuita y otra versión Premium con funcionalidades adicionales como acceso a mapas de zonas de riesgo, envío de alertas por SMS y llamada falsa.

La aplicación Sister esta desarrollada sobre Firebase de Google, esta es una plataforma en la nube para el desarrollo de aplicaciones móviles y Web. Los datos de la aplicación están almacenados en Cloud Firestore, una base de datos NoSQL , flexible, escalable y en la nube que permite almacenar y sincronizar datos para el desarrollo tanto del lado del cliente como del servidor.

Cloud Firestore, es una base de datos NoSQL orientada a documentos que contienen campos que se asignan a valores. Los documentos se almacenan en contenedores llamados colecciones. Estos documentos admiten diferentes tipos de datos: desde cadenas de caracteres (strings), números hasta objetos o documentos anidados, además permite crear estructuras jerárquicas que se ajustan a medida que la base de datos crece. (Firebase - Cloud Firestore, 2022)

3.1.1 Estructura de la Base de Datos

Uno de los objetivos del trabajo, es la extracción de los datos de la aplicación **Sister**, para ello, es necesario conocer como están almacenados los datos. Como comentamos los datos de la aplicación Sister están almacenados en Cloud Firestore una BBDD NoSQL orientada a documentos. En la Figura 4, hemos representado la estructura de la base de datos, como podemos ver tiene una estructura jerárquica bastante compleja, donde tenemos colecciones dentro de documentos.

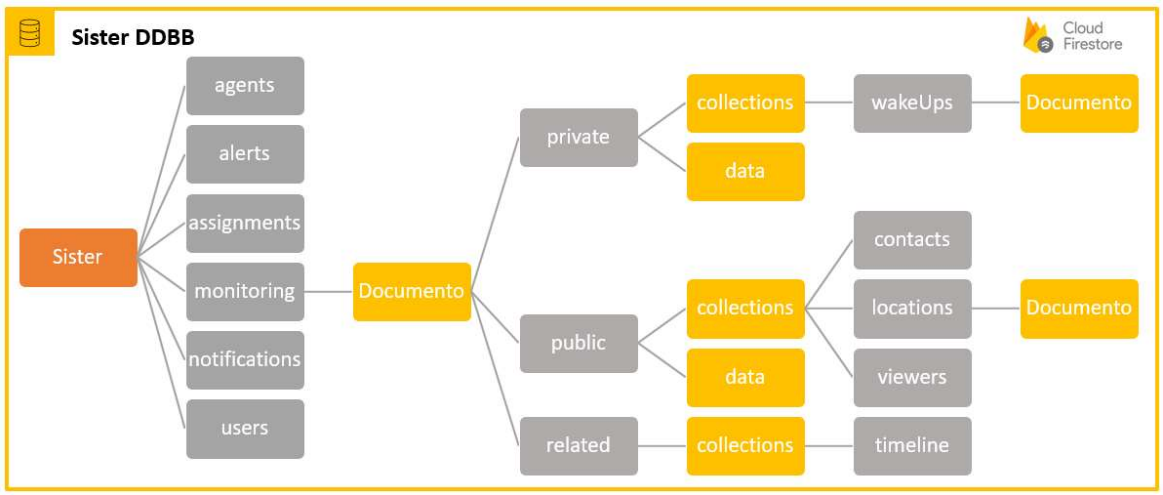


Figura 4: Diagrama que representa la estructura Jerárquica de la base de Datos SIster. Las colecciones están resaltadas en color gris y los documentos en color gris

La colección **monitoring** guarda todas las geolocalizaciones de los usuarios, estas se almacenan en un documento de la colección **locations**, cuando activan en la aplicación las siguientes funcionalidades:

1. Compartir su ubicación
2. Pulsa el botón SOS
3. Activa la localización que es compartida con sus usuarios de confianza

Al guardar las geolocalizaciones de una usuaria, también se almacena el identificador de esta, en la colección *monitoring/private/data*. Considerando que los datos de los usuarios son confidenciales, para nuestro trabajo utilizaremos únicamente este identificador (`ownerUID`), no extraeremos ningún otro tipo de información.

Cuando la aplicación guarda todas las geolocalizaciones de los usuarios, también almacena el estado de la alarma, destino, fecha de finalización de la alarma. Estos datos se almacenan en la colección *monitoring/documento/public/data*.

Adicionalmente, cuando un cliente entra en estado de alarma se generan unos "pings" que son almacenados en un documento de la colección **wakeUps**, estos se utilizan para saber si los usuarios están activos.

Los datos que extraeremos de la BBDD de la aplicación Sister, serán los generados por los usuarios cuando seleccionan alguna funcionalidad que activa el registro de las geolocalizaciones. Estos registro se encuentran en las colecciones *monitoring/documento/private/data* y *monitoring/documento/public/data monitoring/documento/public/collections/locations*. La Tabla 3, describimos todas las columnas o atributos que extraeremos de la BBDD:

Columna	Descripción	Tipo de Dato
monitoring_id	identificador único generado cada vez que un usuario selecciona alguna de las funcionalidades que activa el registro de las geolocalizaciones	String
user_Id	Número de identificación del usuario	Número entero
isAlert	Es una valor booleano (True/False) que indica si una alerta de SOS está activa. Indica que la usuario tiene una percepción de inseguridad (True) o no (False)	Booleano
activate	Es una valor booleano (True/False) que indica si la alerta continúa o no activa.	Booleano
latitude, longitude:	Son las coordenadas que permiten ubicar geográficamente la posición del usuario.	Número flotante
UpdateAt	Fecha de registro de la localización del usuario	Timestamp
radius	representa la precisión en metros del punto donde se encuentra realmente el usuario. Esta característica es dependiente de la compañía móvil del usuario.	Número flotante
speed	registra la velocidad del usuario	Número flotante
altitude	registra la altitud o la distancia vertical que existe entre la posición del usuario en relación con el nivel del mar.	Número flotante
activity	Representa el tipo de actividad realizada. Existen cinco (5) tipos de actividades: • UNKNOWN: Desconocida • STILL: Inmóvil o quieto • WALKING: Caminando • RUNNING: Corriendo • VEHICULE: Se mueve utilizando un medio de transporte	String

Tabla 3: Columnas a extraer de la base de datos Sister en Cloud Firestore

3.2 Obtener los datos

La fase de obtención de los datos esta divida en dos procesos: el primero tiene como objetivo realizar una extracción masiva de los datos, que serán utilizados en el análisis exploratorio y para el entrenamiento del modelo de Machine Learning seleccionado. El segundo, es un proceso de extracción de datos continuo, que se ejecutará diariamente, a partir de la fecha de la primera

extracción y tiene como objetivo obtener nuevos datos de entrada para modelos de Aprendizaje Automático previamente entrenado.

Debido al gran volumen de datos almacenados y los generados diariamente a través de la aplicación Sister, presente en 146 países y 4.330 ciudades alrededor del mundo. Los procesos de extracción de datos se ejecutarán para una ciudad en específico, este valor será un parámetro de entrada en ambos procesos de extracción.

3.2.1 Creando el modelo de datos

La primera implementación para crear el modelo de datos consistió en una extracción masiva de datos, desarrollada en **AWS Batch** porque ser un servicio diseñado para el procesamiento por lotes. Sin embargo, la aplicación fallaba al ejecutar los módulos para acceder a la BBDD Sister en Cloud Firestore, generando el siguiente error:

```
[ERROR] Runtime.ImportModuleError: …Failed to import the Cloud
Firestore library for Python. Make sure to install the "google-cloud-
firestore" module.
```

Este error se genera cuando la librería `google-cloud-firestore` no está instalada, sin embargo, la librería estaba instalada, existía una carpeta de `google-cloud-firestore` y la instalación se había realizado utilizando los mismos comando del entorno local, donde funcionaba perfectamente. Encontrar el origen del problema fue complejo, porque toda la documentación encontrada en libros, sitios web oficiales y en diferente foros señalaban que el problema se resolvía con la instalación de la librería. La solución al problema estaba en el foro para desarrolladores Stack Overflow, donde el usuario @Kony27 respondió a un problema similar:

"El problema es que el paquete firebase-admin usa algunas bibliotecas que se compilan de manera diferente en diferentes sistemas operativos, por lo que se debe crear una imagen e instalar todo allí, comprimirlo y cargarlo como una capa". (Stackoverflow, 2021)

La solución fue desplegar la aplicación dentro de un contenedor de software, utilizando una imagen con un sistema operativo Linux e instalar todas las librerías y módulos que necesita la aplicación desarrollada para ejecutarse.

Este nuevo modelo de implementación basado en contenedores implicó pasar de una solución en **AWS Batch** a una solución **Amazon Elastic Container Registry (AWS ECR) y Amazon Elastic Container Service (AWS ECS)**. AWS ECR permite crear y acceder al repositorio con la imagen de la aplicación, mientras que el servicio AWS ECS permite gestionar e implementar el

contenedor con la imagen de la aplicación. Para ejecutar la aplicación en el contenedor se creará una tarea manual en **AWS Fargate Task**.

La Figura 5 muestra un diagrama de la solución utilizada para la implementación de la aplicación que ejecuta el proceso de extracción masiva de los datos para crear el nuevo modelo de datos. En este modelo se observar dos entornos:

1. El **Entorno local** contiene la aplicación desarrollada en Python, además de los módulos, parámetros y todas las dependencias que esta necesita para ejecutarse. Para implementar la aplicación se crea un contendor Docker portátil que permite ejecutar la aplicación en la nube o localmente. Posteriormente, el contendor con la imagen de la aplicación se hospeda en un repositorio en **Amazon Elastic Container Registry (AWS ECR)**.
2. El **Entorno AWS Cloud** utiliza varios servicios de AWS: **AWS ECR** contiene el repositorio que hospeda el contendor con la imagen de la aplicación, mientras que desde **AWS ECS** se ejecuta la tarea o servicio permite la ejecución la aplicación de extracción masiva de datos. Los datos producto de la extracción pueden ser almacenados en una base de datos documental MongoDB creada en una instancia de Amazon EC2 o en un fichero en formato CSV que se almacenará en un **bucket** de **Amazon S3**.

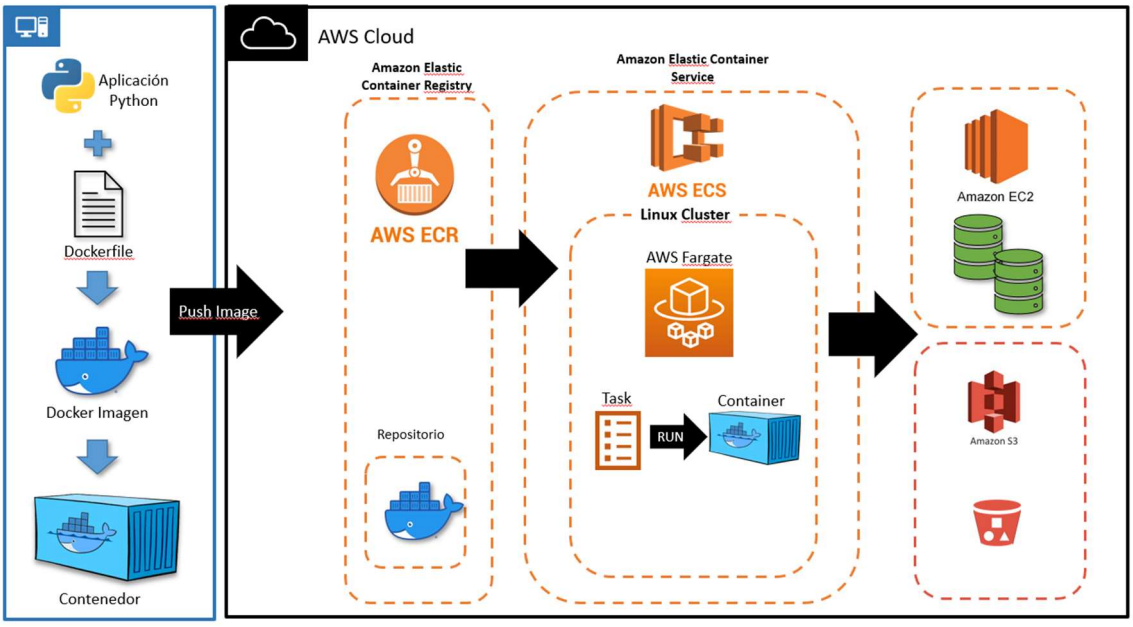


Figura 5: Modelo para la implementación del proceso de extracción masiva de datos en AWS

3.2.1.1 Entorno local

El entorno local está conformado por una aplicación desarrollada en Python; la cual se encuentra en el directorio `app_batch_docker`. Como muestra la Figura 6, este directorio contiene los paquetes, módulos, directorios y ficheros que necesita la aplicación para ejecutarse.

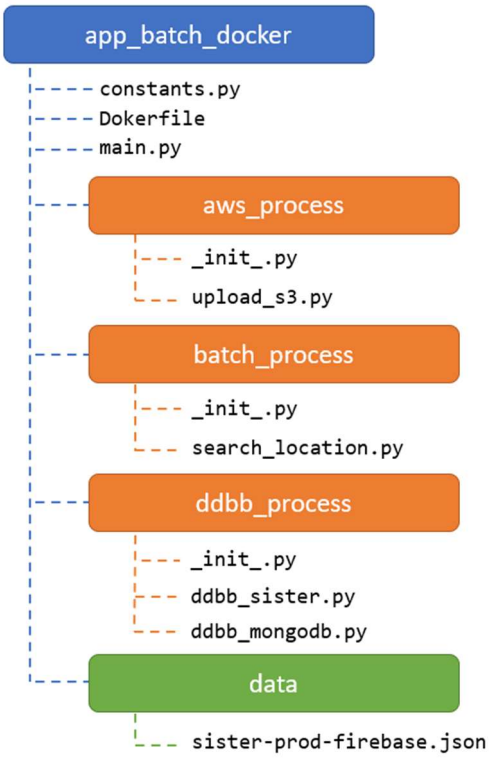


Figura 6: Estructura de la aplicación Python para la extracción masiva de datos

La carpeta o directorio `data` (color verde) contiene los ficheros en formato CSV, que se generan cuando ejecutamos localmente la aplicación, además de los ficheros de autentificación necesarios para conectarse a las BBDD.

En la Figura 6, observamos que los módulos que componen la aplicación de extracción de datos masiva están organizados en tres (3) paquetes (color naranja):

- **aws_process:** El paquete `aws_process` está conformado por el módulo `upload_s3.py`, que contiene la función que permitir subir los ficheros en formato CSV al **bucket** creado utilizando el servicio de almacenamiento en la nube de AWS (**Amazon S3**).
- **batch_process:** Este paquete está conformado por el módulo `search_locations.py` que contiene todas las funciones que permiten extraer los registros de geolocalización de un usuario, además de otras funciones de

geolocalización necesarias para extraer la ciudad a la cual corresponden las coordenadas (latitud y longitud) registradas por los usuarios en la aplicación Sister.

- **ddbb_process:** Este paquete está conformado por los módulos `ddbb_sister.py` y `ddbb_mongodb.py` que contiene todas las funciones que permiten extraer y validar los datos desde la Base de Datos de la aplicación Sister en Cloud Firestore para almacenarlos en la Base de Datos MongoDB alojada en una instancia de Amazon EC2, si el parámetro `SAVE_DDBB = True`, de los contrario se genera un fichero en formato CSV con la información extraída.

En el directorio de trabajo `app_batch_docker` además de los paquetes antes mencionados, se encuentran los módulos `main.py` y `constants.py`; además del fichero `Dockerfile` a partir del cual crearemos la imagen y el contenedor con la aplicación y sus dependencias.

- **main.py:** Es el módulos principal de la aplicación, donde se importan los módulos y las librerías que necesitan la aplicación. Además, este módulo contiene toda la lógica de la aplicación y desde aquí se invocan los diferentes funciones que permiten realizar la extracción de los datos para luego ser almacenados en una BBDD o en un fichero CSV.
- **constants.py:** Este módulo contiene todas las constantes y/o parámetros que necesita la aplicación.
- **Dockerfile:** Este archivo contiene las instrucciones para compilar la imagen Docker. La Figura 7, muestra el contenido de Dockefile, la primera línea indica que partimos de la imagen `amazonlinux`, disponible en **DockerHub**. En la cual, se realiza la instalación de Python y de las librerías de terceros necesarias. Finalmente, para completar el entorno de trabajo se copian todos los archivos y directorios desde el entorno local a la imagen Docker.

```
FROM amazonlinux:latest
RUN yum -y install wich unzip python3 pip3
RUN pip3 install boto3
RUN pip3 install firebase_admin
RUN pip3 install geopandas
RUN pip3 install geopy
RUN pip3 install google-cloud-core
RUN pip3 install google-cloud-firestore
RUN pip3 install grpcio
RUN pip3 install setuptools
RUN pip3 install wheel
RUN pip3 install pymongo

WORKDIR /app_batch_docker
COPY ./aws_process ./aws_process
COPY ./batch_process ./batch_process
```

```
COPY ./ddbb_process ./ddbb_process
COPY ./data ./data
COPY constants.py .
COPY main.py .
CMD ["python3", "main.py"]
```

Figura 7: Contenido del Fichero Dockerfile

Una vez tenemos la aplicación para la extracción masiva de datos con todas sus dependencias, el siguiente paso es crear una contendor Docker con la aplicación que posteriormente subiremos al entorno **AWS Cloud**. A continuación detallaremos este proceso:

Pasos:

1. Abrir la terminal y ubicarse en el directorio de la aplicación `app_batch_docker`. En este directorio se encuentra la aplicación, sus dependencias y el fichero Dockerfile que utilizaremos para compilar la imagen Docker.
2. Crear la imagen `app_batch_docker`, ejecutando el comando:

```
docker build -t app_batch_docker .
```

3. Ejecutar la imagen `app_batch_docker`; al ejecutar la imagen en realidad se crea el contendor. Este paso, nos permite ejecutar la imagen localmente y validar que funciona correctamente:

```
docker run -it app_batch_docker
```

3.2.1.2 Entorno AWS Cloud

Tenemos una imagen Docker `app_batch_docker` que contiene la aplicación de extracción de datos masivo y sus dependencias; la cual podemos ejecutar tanto localmente como en la nube. Para ejecutar la aplicación en la nube utilizaremos el servicio **AWS ECR**, para hospedar la imagen Docker con la aplicación y el servicio **Amazon ECS**, para ejecutar la imagen con la aplicación.

Antes de subir la imagen de la aplicación a la nube, creamos en **AWS ECR** un repositorio privado `safedeploy`. Una vez creado el repositorio, podemos acceder a `View push commands` (ver Figura 8) donde tenemos los comandos necesarios para conectarnos al repositorio y subir la imagen de la aplicación.

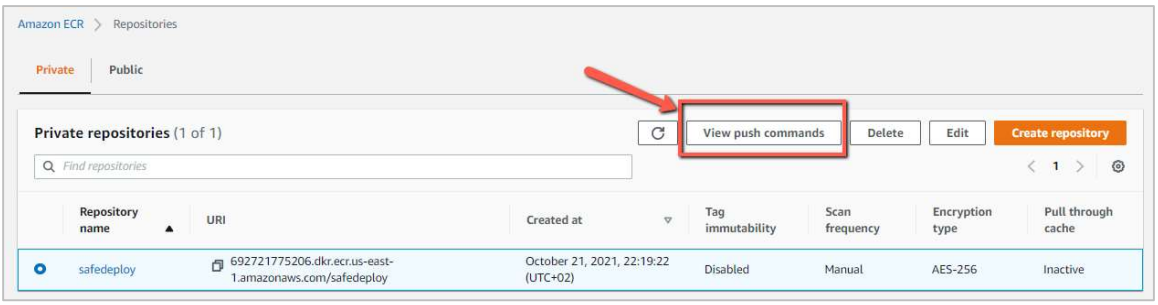


Figura 8: Repositorio safedeploy en AWS ECR

Una vez creado el repositorio, el siguiente paso consiste en subir la imagen `app_batch_docker` desde Docker al repositorio `safedeploy` en **AWS ECR**, utilizando los comando disponibles en `View push commands` (Ver Figura 9).

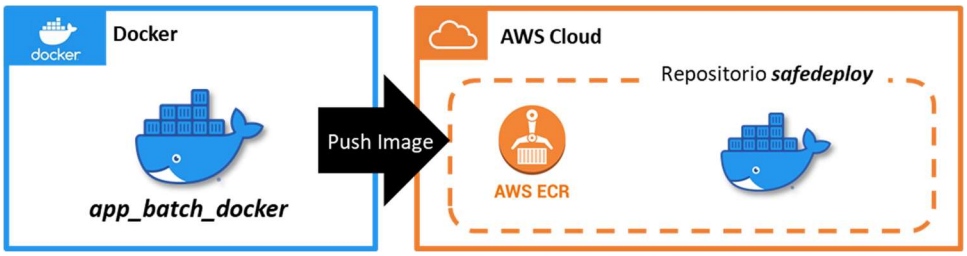


Figura 9: Subir una imagen desde Docker a AWS ECR

Este proceso consta de dos (2) pasos, que como en pasos anteriores podemos ejecutar desde la terminal o desde AWS CLI:

- Crear la conexión al repositorio en AWS ECR:

```
aws ecr get-login-password --region us-east-1 | docker login --
username AWS --password-stdin 99999999999.dkr.ecr.us-east-
1.amazonaws.com
```

- Subir la imagen `app_batch_docker` al repositorio `safedeploy` en AWS ECR:

```
docker tag app_batch_docker 99999999999.dkr.ecr.us-east-
1.amazonaws.com/safedeploy

docker push 99999999999.dkr.ecr.us-east-1.amazonaws.com/safedeploy
```

Una vez completado el proceso de subir la imagen al repositorio, en este se crea una versión de la imagen de la aplicación con el nombre `latest`. Este proceso debemos repetirlo cada vez que creemos una nueva versión de la aplicación, estas versiones quedarán almacenadas en nuestro repositorio, de manera de poder ejecutar cualquiera de las versiones. La Figura 10, muestra la última versión de la imagen creada en el repositorio `safedeploy`.

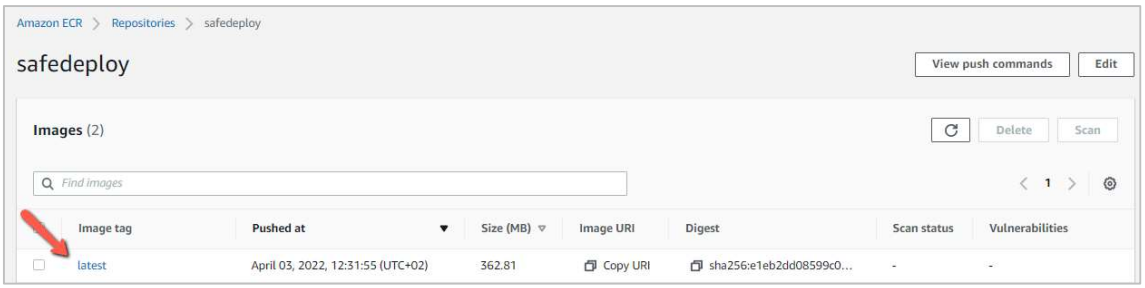


Figura 10: Repositorio safedeploy con la última imagen de la aplicación

Tenemos en **AWS ECR** un repositorio con la imagen `app_batch_docker` que contiene la aplicación lista para ejecutarse. Como comentamos para ejecutar esta imagen debemos configurar el servicio **Amazon ECS**. Lo primero es crear un **Clúster** que puede contener una o más instancias EC2 donde podemos ejecutar solicitudes de **Tareas,** es decir crear una tarea o servicio para ejecutar la imagen contenida en el repositorio `safedeploy`.

Para ejecutar la imagen contenida en el repositorio `safedeploy`, creamos la tarea `execute_app_batch_docker` en **Amazon ECS** con tipo de lanzamiento **Fargate**, porque no necesitamos aprovisionar o administrar instancias de **Amazon EC2** y los precios se basan en el tamaño de la tarea. Durante la configuración de la nueva tarea, creamos el contenedor `app_batch_container` con 12GB y 2 CPU's, al cual asociamos la imagen contenida en el repositorio `safedeploy` utilizando la URL:

`999999999.dkr.ecr.us-east-1.amazonaws.com/safedeploy`.

Nuestro entorno en Amazon ECS está configurado, el siguiente paso es ejecutar la tarea, para ellos seleccionamos la opción **Run Task** del menú **Actions** en la ventana de definición de tareas de `execute_app_batch_docker` (Ver Figura 11).

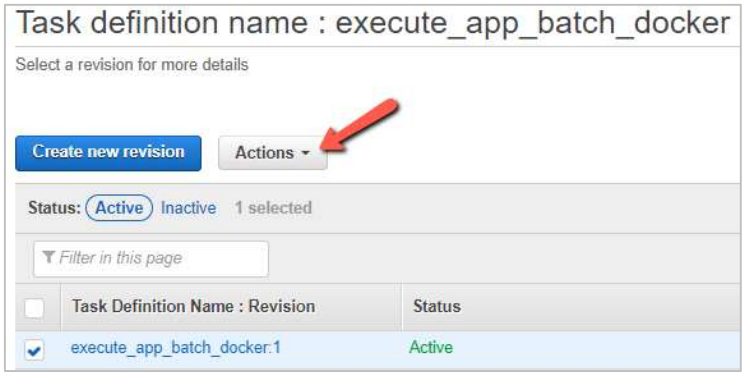


Figura 11: Ventana de definición de Tareas Activas en AWS ECS

Al ejecutar la tarea `execute_app_batch_docker`, debemos especifica el tipo de lanzamiento, en nuestro caso **Fargate** y seleccionar una de la **Subnets** disponibles. Al ejecutar la tarea,

aparece la ventana de la Figura 12, donde podemos visualizar el contendor que está en ejecución (RUNNING). En la pestaña Logs podemos visualizar la salida generada por la aplicación, además de los errores ocurridos durante el proceso.

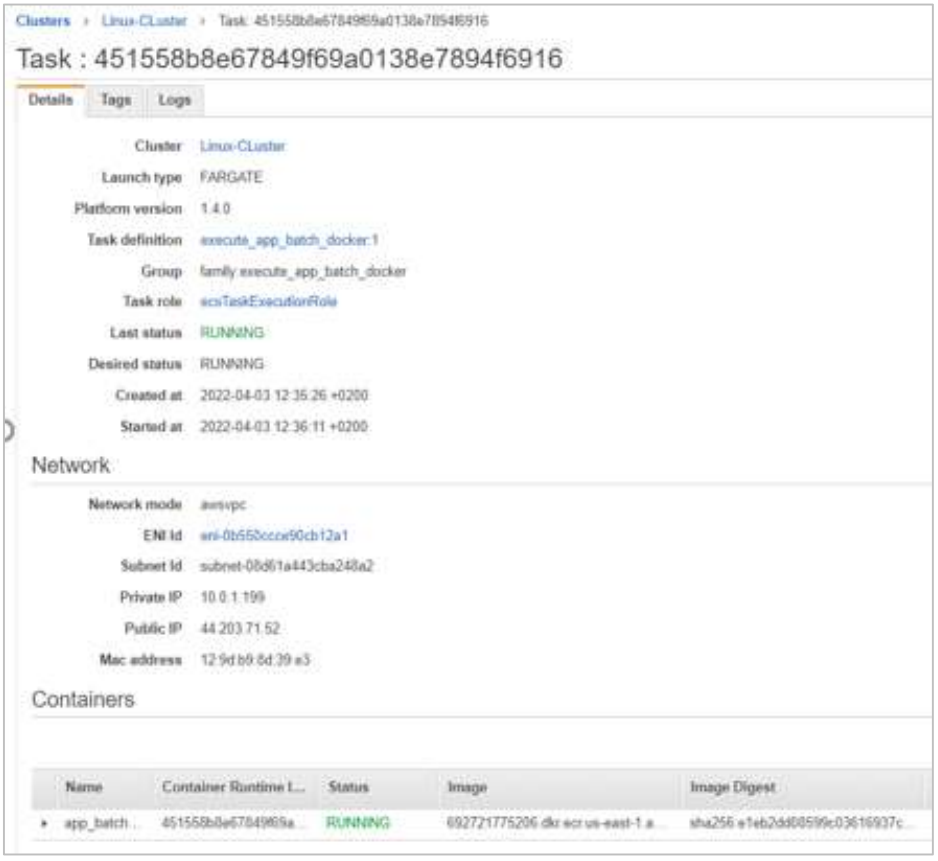


Figura 12: Ventana de la tarea en ejecución

Una de las ventajas detectadas al utilizar los servicios del proveedor en la nube es una disminución de los tiempos de ejecución. Si consideramos que los tiempos medios de ejecución del contenedor con la aplicación en el entorno local superaron las 5 horas; mientras que al ejecutar el mismo contendor en la nube, el tiempo baja a una media 3 horas. Un disminución de aproximadamente 40% del tiempo de ejecución.

3.2.1.3 La aplicación

`main.py` es el módulo principal de la aplicación; desde donde importamos las librerías y los módulos desarrollados en Python (`constants`, `aws_process`, `batch_process` y `ddbb_process`). Una vez importados los módulos tenemos acceso a todas las funciones y parámetros necesarios para ejecutar el proceso de extracción.

La aplicación se conecta a la BBDD de la aplicación `Sister en Cloud Firestore`, específicamente a las colecciones `monitoring` de donde extraemos el identificador del usuario

(user_id) y de la colección `locations` obtenemos un documento embebido entre los cuales tenemos los datos de geolocalización del registro.

Una vez obtenidos los datos de geolocalización, verificamos si los mismos corresponde a la ciudad que hemos pasado como parámetro, en nuestro caso **Madrid**. Si los datos corresponden a la ciudad, procedemos a almacenados en una BBDD datos documental en MongoDB, si el parámetro `SAVE_DDBB = True` o en un fichero CSV si el parámetro `SAVE_DDBB = False`.

BBDD Documental

La Base de Datos en MongoDB, la implementamos en una instancia de AWS EC2 `DDBB_MongoDB_Wave`, para ello configuramos una VPC y un `keypair` privada para conectarnos de forma segura a la instancia una vez se inicie la BBDD. Una vez creada la instancia realizamos la instalación del Shell de MongoDB y creamos la BBDD desde la consola.

La base de datos documental `sisterlocations` está conformada por una colección `locations` con la siguiente estructura:

```
{
    "_id": {
        "$oid": "61d995c0368cfb48b842c345"
    },
"monitoring": "ffaf0ed537a44e888d30b9a172dadf1b",
    "user_id": "268",
    "isAlert": true,
    "activate": false,
    "latitude": 40.420628,
    "longitude": -3.714041,
    "UpdateAt": {"$numberLong": "1598025865249"},
    "radius": 5,
    "speed": 0,
    "altitude": 0,
    "activity": "UNKNOWN",
    "city": "Madrid"
}
```

Fichero en formato CSV

El nombre del fichero en formato CSV, se genera concatenando la palabra "`locations`" + `fecha + .csv`. Donde fecha corresponde a la fecha y hora en la cual se ejecuta el proceso de extracción de datos. Los datos almacenados en el fichero CSV tienen la siguiente estructura:

```
monitoring_id,user_id,isAlert,activate,latitude,longitude,UpdateAt,radius,speed,altitude,activity
0018e33ff2dc49deb0d27d464001ac69,268,True,False,40.420628,-3.714041,1598025865249,5.0,0.0,0.0,UNKNOWN,Madrid
0018e33ff2dc49deb0d27d464001ac69,268,True,False,40.420628,-3.714041,1598025845249,5.0,0.0,0.0,UNKNOWN,Madrid
0018e33ff2dc49deb0d27d464001ac69,268,True,False,40.420628,-3.714041,1598025835252,5.0,0.0,0.0,UNKNOWN,Madrid
0018e33ff2dc49deb0d27d464001ac69,268,True,False,40.420628,-3.714041,1598025841013,5.0,0.0,0.0,UNKNOWN,Madrid
0018e33ff2dc49deb0d27d464001ac69,268,True,False,40.420628,-3.714041,1598025854160,5.0,0.0,0.0,UNKNOWN,Madrid
0018e33ff2dc49deb0d27d464001ac69,268,True,False,40.420628,-3.714041,1598025862786,5.0,0.0,0.0,UNKNOWN,Madrid
0018e33ff2dc49deb0d27d464001ac69,268,True,False,40.420628,-3.714041,1598025825254,5.0,0.0,0.0,UNKNOWN,Madrid
0018e33ff2dc49deb0d27d464001ac69,268,True,False,40.420628,-3.714041,1598025855250,5.0,0.0,0.0,UNKNOWN,Madrid
```

Figura 13: Estructura del fichero .csv generado durante el proceso de extracción de datos masivo

El fichero CSV se crea en un directorio temporal dentro de la imagen. Una vez completado el proceso de extracción de datos, se ejecuta la función `upload_file_s3`, la cual sube el fichero CSV desde el directorio temporal al bucket creado en Amazon S3; posteriormente se elimina el fichero del directorio temporal.

3.2.2 Incorporar nuevos datos al modelo

Una vez creado el modelo y realizada la primera extracción de datos, necesitamos un proceso diario que alimente nuestro modelo con datos nuevos, que pueden ser utilizados para validar modelos de Aprendizaje Automático entrenados previamente.

Para este proceso de incorporación de datos nuevos, hemos utilizado el servicio **AWS Lambda**, este servicio permite ejecutar código en una infraestructura de alta disponibilidad. Es ideal para carga de trabajo cortas basada en eventos, debido a que estas funciones tienen un tiempo máximo de ejecución de 15 minutos. (AWS Amazon Lambda, 2022)

Debido a que las funciones lambda tienen una duración máxima de ejecución, el código se ha dividido en dos funciones lambda como se muestra en la Figura 14. La primera función Lambda `GetMonitoringIDfromDate,` extrae los datos de monitorización de un usuario (`monitoring_id`, `user_id`, `isAlert`, `activate`) desde una fecha recibida como parámetro. Debido al volumen de información y para garantizar que la función lambda se ejecute en menos de 15 minutos, dividimos los datos en grupos de 500 registros. Estos registros, son almacenados en ficheros en formato .csv que posteriormente son cargados en un bucket en Amazon S3.

Al subir estos ficheros al **bucket** creado en **Amazon S3**, se activa un trigger[10] o evento que ejecuta la segunda función Lambda `GetlocationsTrigger.` Esta función lambda `GetlocationsTrigger,` procesa cada línea del fichero y extrae todas las localizaciones correspondientes a cada `monitoring_id` registrado. Por cada fichero que genera la primera función lambda, se ejecuta la función `GetlocationsTrigger`, es decir que podemos tener varias

[10] Un trigger o desencadenador es un recurso o configuración que invoca una función de Lambda. Los desencadenadores incluyen los servicios de AWS que puede configurar para invocar una función. (Conceptos de Lambda - AWS Lambda (amazon.com)

funciones ejecutándose en paralelo. Los datos de las localizaciones son almacenados en la base de datos MongoDB, si el parámetro `SAVE_DDBB = True` o en un fichero CSV si el parámetro `SAVE_DDBB = False`.

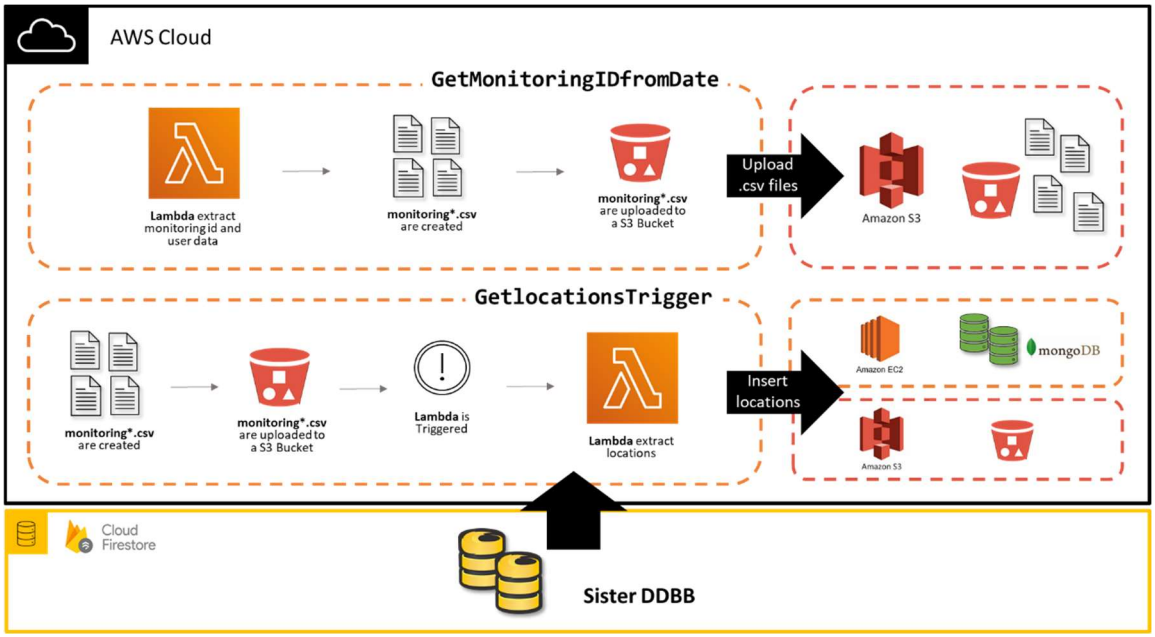


Figura 14: Funciones Lambda para la extracción de nuevos datos

A continuación detallaremos los recursos adicionales, el entorno y módulos que conforman estas funciones lambda:

En AWS Lambda podemos ejecutar código Python, esto nos permite reutilizar todos los módulos Python desarrollados para la aplicación de extracción de datos. En AWS Lambda, el código se ejecuta en un entorno que incluye un SDK para Python, en nuestro caso hemos utilizado el AWSSDK para Python 3.9 (boto3-1.18.55 botocore-1.21.55), en un sistema operativo Amazon Linux 2. El entorno AWSSDK para Python 3.9 utilizado no contiene las librerías de terceros que necesitan nuestras aplicaciones (firebase, geopy, google-cloud, gprcio, y pymongo); por ello es necesario crear una capa con las librerías adicionales.

Para ejecutar ambas funciones lambda hemos creado la capa o layer `PythonLibraries_LambdaFunctions` (Ver Figura 15), que no es más que un recursos adicional de AWS Lambda, que contiene todas las librerías y/o dependencias que necesitan nuestras funciones lambda para ejecutarse. Una capa es un fichero .zip generado de la carpeta "Python", en la cual hemos instalado previamente todas las librerías de terceros necesarias. De esta forma, cuando invoquemos una librería desde una función lambda esta la buscará en el entorno de la aplicación y en las diferentes capas del entorno.

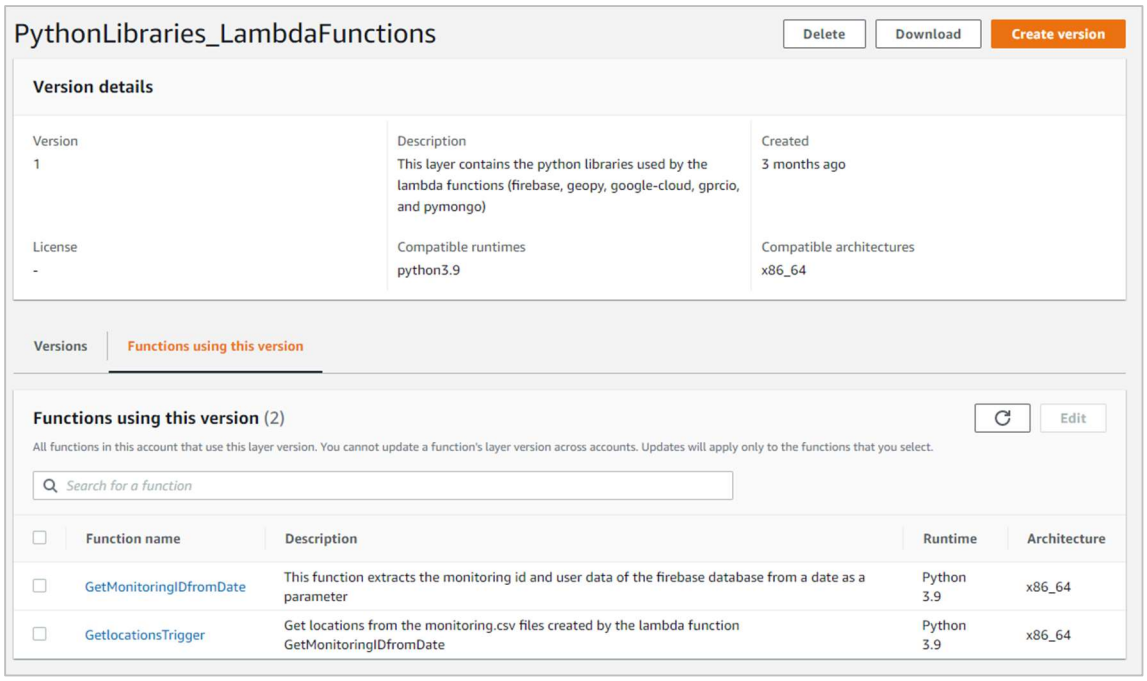


Figura 15: Layer o Capa Python 3.9 creada en AWS Lambda, utilizada por las funciones Lambda GetMonitoringIDfromDate y GetlocationsTrigger

3.2.2.1 Función Lambda "GetMonitoringIDfromDate"

La función lambda `GetMonitoringIDfromDate` como se muestra en la Figura 16, utiliza una capa o entorno con todas las librerías que necesita para conectarse a la BBDD Cloud Firestore de la aplicación Sister. Observamos que no tienen definido ningún Trigger o destino.

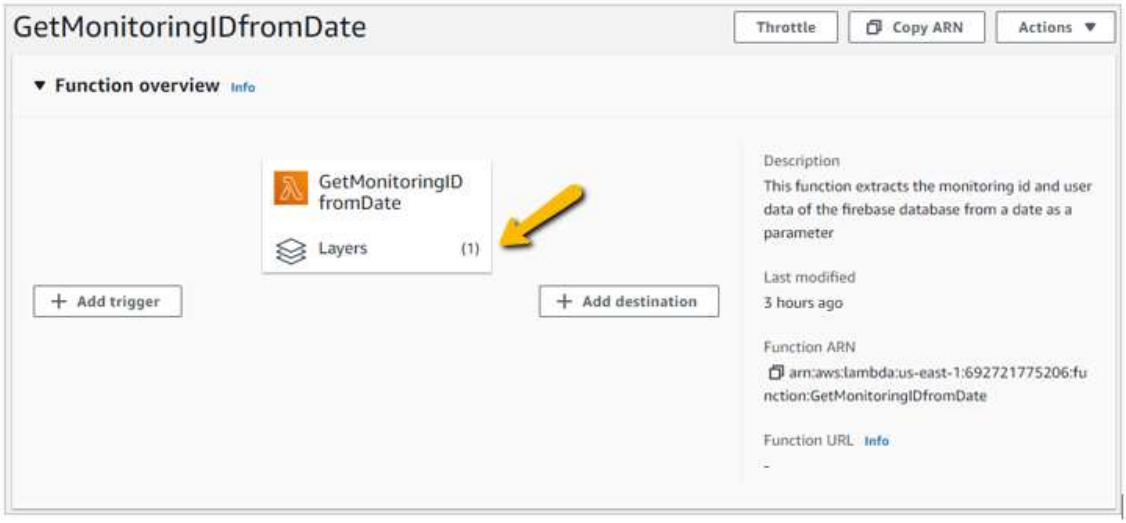


Figura 16: Ventana de información de la función Lambda GetMonitoringIDfromDate en AWS

Como observamos en la Figura 17, la estructura de la función lambda está conformada por un directorio de trabajo con el mismo nombre de la función `GetMonitoringIDfromDate`.

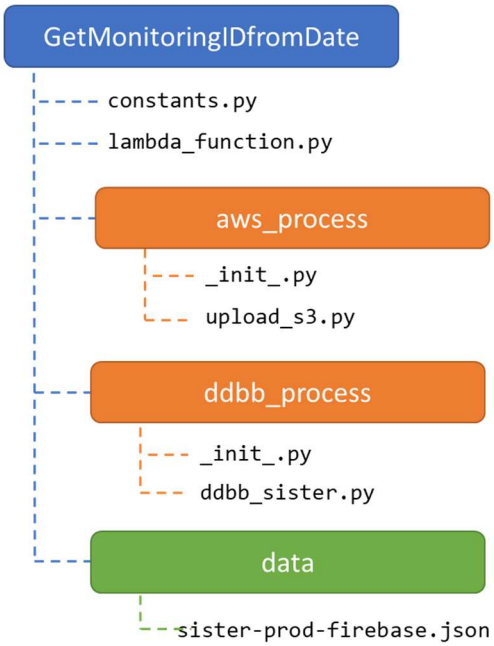


Figura 17: Estructura de la función Lambda GetMonitoringIDfromDate en AWS

En este directorio de trabajo encontramos los paquetes, directorios, módulos y ficheros que necesita la función para ejecutarse. En color naranja observamos los paquetes `aws_process` y `ddbb_process`:

- **aws_process:** El paquete `aws_process` está conformado por el módulo `upload_s3.py`, que contiene la función que permitir subir los ficheros en formato CSV al **bucket** creado en **Amazon S3**).
- **ddbb_process:** Este paquete está conformado por el módulo `ddbb_sister.py` que contiene todas las funciones que permiten extraer y validar los datos desde la Base de Datos de la aplicación Sister en Cloud Firestore.

La carpeta o directorio `data` (color verde) contiene el fichero `sister_prod_firebase.json` con los datos de autentificación necesarios para conectarse a las BBDD Cloud Firestore.

En la raíz del directorio de trabajo encontramos los módulos `constants.py` y `lambda_function.py` ambos desarrollados en Python. El primero contiene todas las constantes y/o parámetros que necesita la función Lambda, el segundo contiene toda la lógica de la función lambda.

lambda_function.py: Es el módulo principal desde donde importamos las librerías y los módulos desarrollados en Python (`constants`, `aws_process`, y `ddbb_process`). Una vez importados los módulos, tenemos acceso a todas las funciones y parámetros necesarios para ejecutar la función.

La función `lambda_handler(event, context)` recibe dos (2) parámetros el evento y el contexto de la función lambda. En esta función se encuentra todo el código de la función, el cual describiremos a continuación:

1. La función se conecta a la BBDD de la aplicación Sister en Cloud Firestore, específicamente a la colección **monitoring**. De donde extraemos entre otros datos, el identificador del usuario (`user_id`) y las geolocalizaciones de los usuarios almacenadas en cada `monitoring_id`.
2. Los datos son almacenados en lotes de 500 registros en un fichero temporal en formato CSV con el prefijo "monitoring" y un consecutivo con el número del lote asignado. Cuando el fichero alcanza los 500 registros almacenados, se cierra, se guarda una copia en el **bucket** creado en **Amazon S3**. Posteriormente se elimina del espacio temporal.

3.2.2.2 Función Lambda "GetlocationsTrigger"

La función lambda `GetlocationsTrigger` como se muestra en la Figura 18, utiliza una capa o entorno con todas las librerías que necesita tanto para conectarse a la BBDD Cloud Firestore de la aplicación Sister, como para conectarse a la base de datos MongoDB. En esta función hemos configurado un trigger de Amazon S3. El trigger que invocará la función lambda se activará cuando un fichero con el prefijo "monitoring" y extensión CSV se cree o se suba en el **bucket** creado en **Amazon S3**. Esta función no tiene ningún destino configurado.

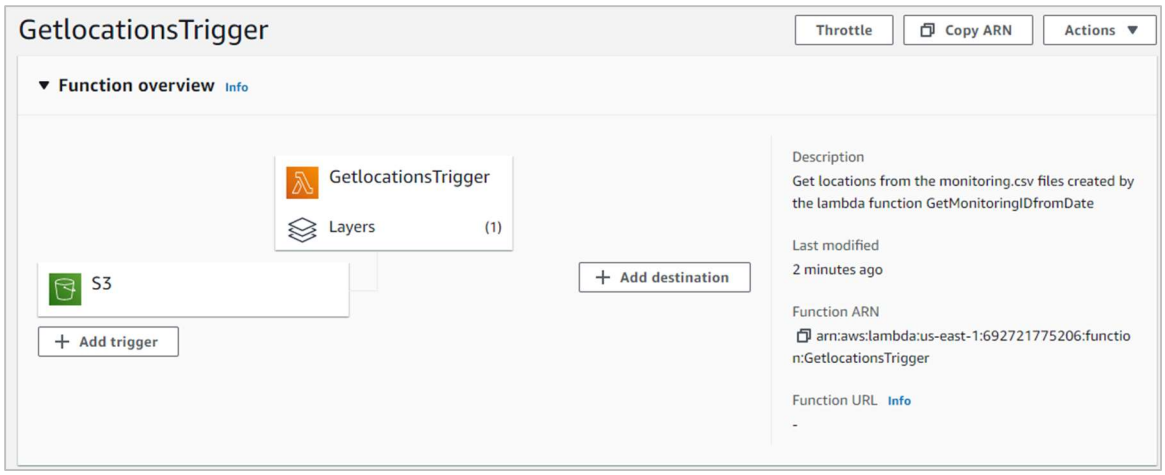


Figura 18: Ventana de información de la función Lambda GetlocationsTrigger en AWS

En la Figura 19, observamos que la estructura de la función lambda está conformada por un directorio de trabajo con el mismo nombre de la función `GetlocationsTrigger`.

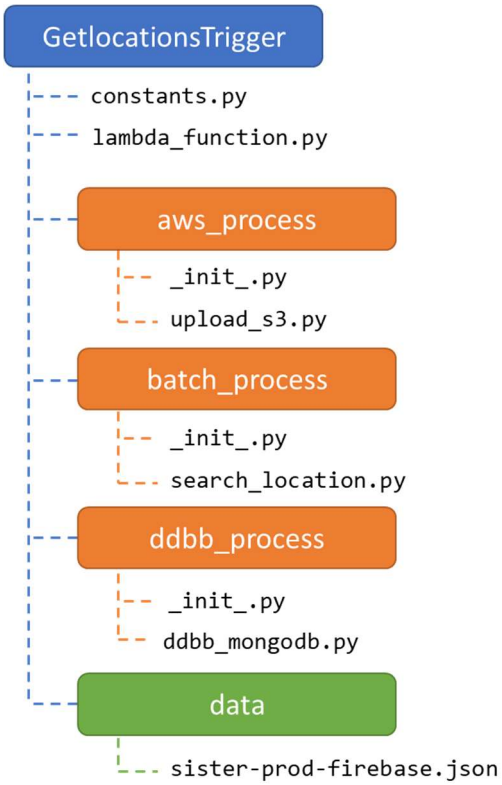


Figura 19: Estructura de la función Lambda GetlocationsTrigger en AWS

El directorio de trabajo contiene los paquetes, directorios, módulos y ficheros que necesita la función para ejecutarse. En color naranja observamos los paquetes `aws_process`, `batch_process` y `ddbb_process`:

- **aws_process:** El paquete `aws_process` está conformado por el módulo `upload_s3.py`, que contiene la función que permitir subir los ficheros en formato CSV al **bucket** creado en **Amazon S3**.
- **ddbb_process:** Este paquete está conformado por el módulo ddbb_sister.py que contiene todas las funciones que permiten extraer y validar los datos desde la Base de Datos de la aplicación Sister en **Cloud Firestore**.
- **batch_process:** Este paquete está conformado por el módulo `search_locations.py` que contiene todas las funciones que permiten extraer los registros de geolocalización de un usuario en la aplicación Sister.

La carpeta o directorio data (color verde) contiene el fichero `sister_prod_firebase.json` con los datos de autentificación necesarios para conectarse a las BBDD **Cloud Firestore**.

En directorio de trabajo encontramos los módulos `constants.py` y `lambda_function.py` ambos desarrollados en Python. El primero contiene todas las constantes y/o parámetros que necesita la función Lambda, el segundo contiene toda la lógica de la función lambda.

lambda_function.py: Es el módulo principal desde donde importamos las librerías y los módulos desarrollados en Python (`constants`, `aws_process`, `batch_process` y `ddbb_process`). Una vez importados los módulos, tenemos acceso a todas las funciones y parámetros necesarios para ejecutar la función.

La función `lambda_handler(event, context)` recibe dos (2) parámetros el evento y el contexto de la función lambda. En esta función se encuentra todo el código de la función, el cual describiremos a continuación:

1. Esta función se ejecuta cuando el fichero con el prefijo "monitoring" y con extensión .csv se sube al **bucket** de **Amazon S3**. Utilizamos el evento[11] `['Records']` para obtener los datos del bucket y del fichero a procesar. Por ejemplo: `monitoring_0.csv`.
2. La función ejecuta una petición GET al **bucket** en **Amazon S3** para abrir el fichero .csv, decodificarlo y extraer la información. Del fichero CSV extraemos la siguiente información `monitoring_id`, `user_id`, `isAlert` y `activate`.
3. Ejecuta la función `ddbb_extract_update_locations` del módulo `batch_process` para cada el `monitoring_id` leído. Esta función se conecta a la BBDD de la aplicación **Sister** en **Cloud Firestore** y obtiene todos los datos de geolocalización registrados en cada `monitoring_id` a partir de la fecha recibida como parámetro (`locationLastUpdate`).
4. Los datos de las localizaciones son almacenados en la base de datos MongoDB, si el parámetro `SAVE_DDBB = True` o en un fichero CSV si el parámetro `SAVE_DDBB = False`.
5. Una vez almacenados los datos, el fichero `monitoring_n.csv` es eliminado del **bucket** de **Amazon S3**.

[11] Un evento es un documento con formato JSON que contiene datos para que una función de Lambda los procese. El tiempo de ejecución convierte el evento en un objeto y lo pasa al código de la función. (Conceptos de Lambda - AWS Lambda (amazon.com))

4. Análisis Exploratorio de los Datos del nuevo modelo

Todos los datos utilizados en el análisis exploratorio de los datos, ha sido extraídos de los notebook de Jupyter desarrollados en Python para este trabajo de fin de Master (Ver anexos: `Analisis_Datos_Comunidad_Madrid.ipynb`, `Analisis_Datos_Monitoring_CM.ipynb`), así como el informe desarrollado en PowerBI `DatosGenerales.pbix` para análisis los datos mundiales.

4.1 Volumen de Datos

La Base de datos documental Sister cuenta con seis (6) colecciones: agents, alerts, assignments, monitoring, notifications y users (Ver Figura 4). **Monitoring** es la colección donde se registran o almacenan, todos los datos que se generan cuando un usuario activa alguna de las funcionalidades disponibles en la aplicación (compartir ubicación, buscar una dirección, activar un alerta o SOS). Cada documento de la colección monitoring, está conformado a su vez por la colección **locations** ubicada en "/public/collections/locations"; donde se registran todas los datos de geolocalización que se generar durante una sesión.

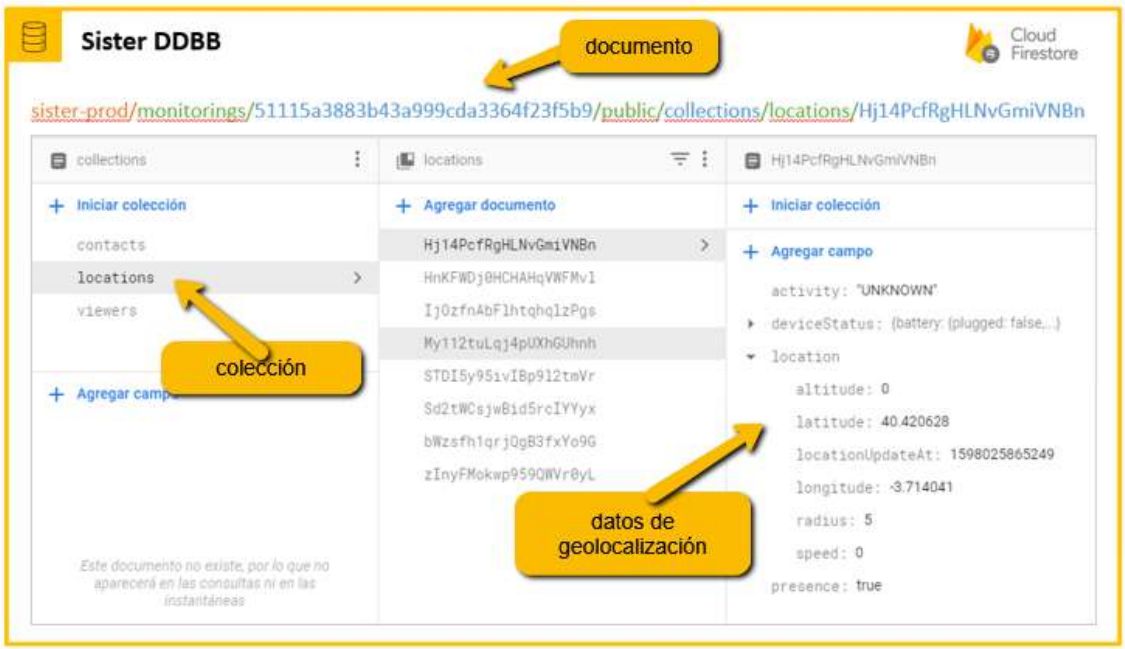
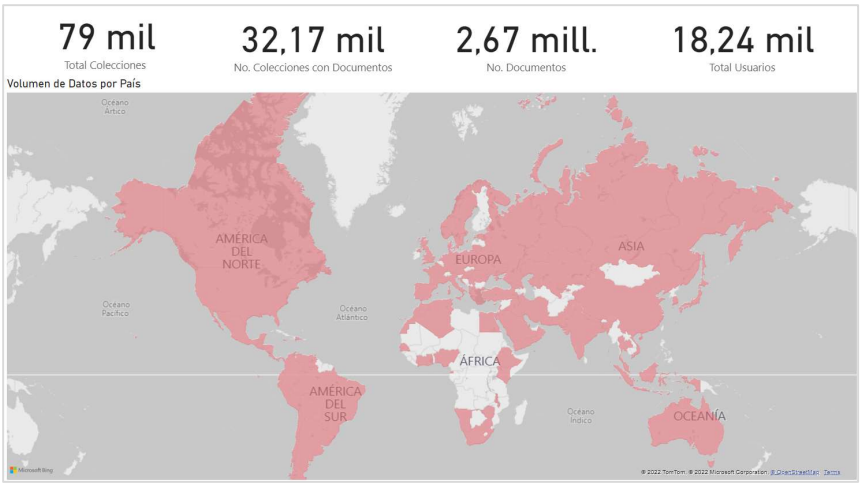


Figura 20: Ejemplo de un documento de la colección Locations donde se almacenan los datos de geolocalización de una sesión.

Como resultado de la ejecución del proceso de extracción masiva de datos en el entorno de producción, obtuvimos datos correspondientes al periodo entre octubre del año 2019 hasta el

27 de mayo del 2022. Un total de 2.666.082 documentos agrupados en 79.294 colecciones o identificadores de monitoreo; de las cuales más de 32.000 contienen 2.67 millones de documentos almacenados. Estos documentos almacenados corresponden a 18.244 usuarios, 116 Países y 4.330 Ciudades (Ver



).

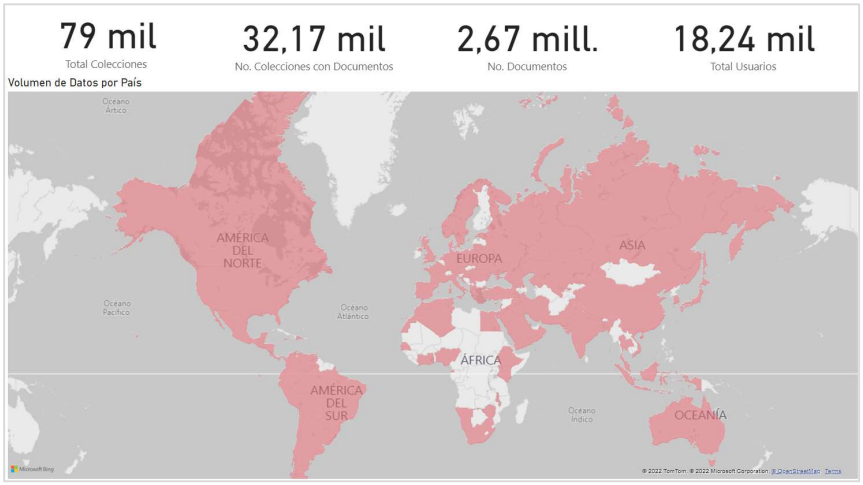


Figura 21: Volumen de datos (DatosGenerales.pbix)

Por otra parte, la ejecución del proceso de extracción de nuevos datos, nos indica que diariamente se generan una media de 90 nuevos monitoreos que generan unos 27.200 documentos o geolocalizaciones.

En la Figura 22, observamos la distribución porcentual de los datos de los 10 principales países. Aproximadamente el 29% de los datos corresponde a México, 20% a España, 13% a Estados Unidos, 11% a Colombia y un 11% a Costa Rica. El 16% restantes corresponde a los 111 piases restantes.

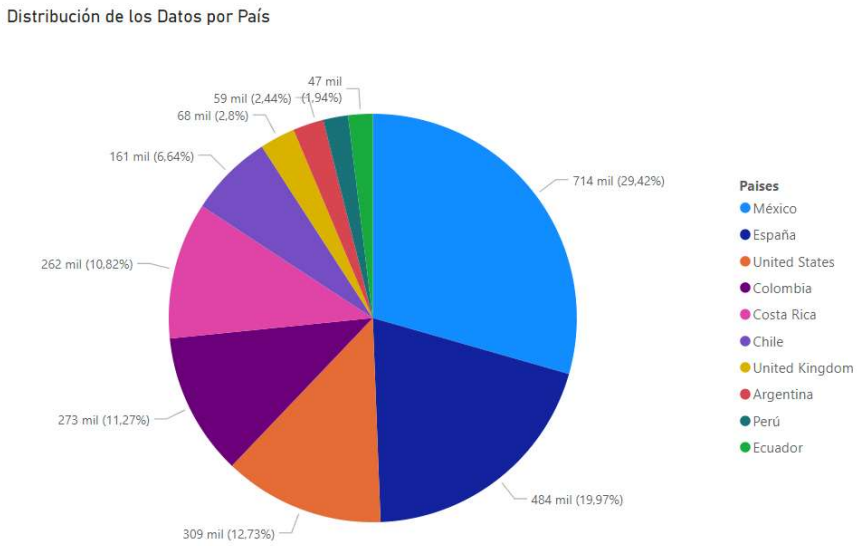


Figura 22: Distribución porcentual del volumen de datos por país. (DatosGenerales.pbix)

España es el segundo país con mayor volumen de datos, con aproximadamente 484.260 documentos, siendo Madrid la ciudad con mayor volumen de datos (18%), seguida de Santiago de Compostela (9%). En la Figura 23, observamos en color rosa las ciudades de España donde se encuentran los usuarios de la aplicación Sister.

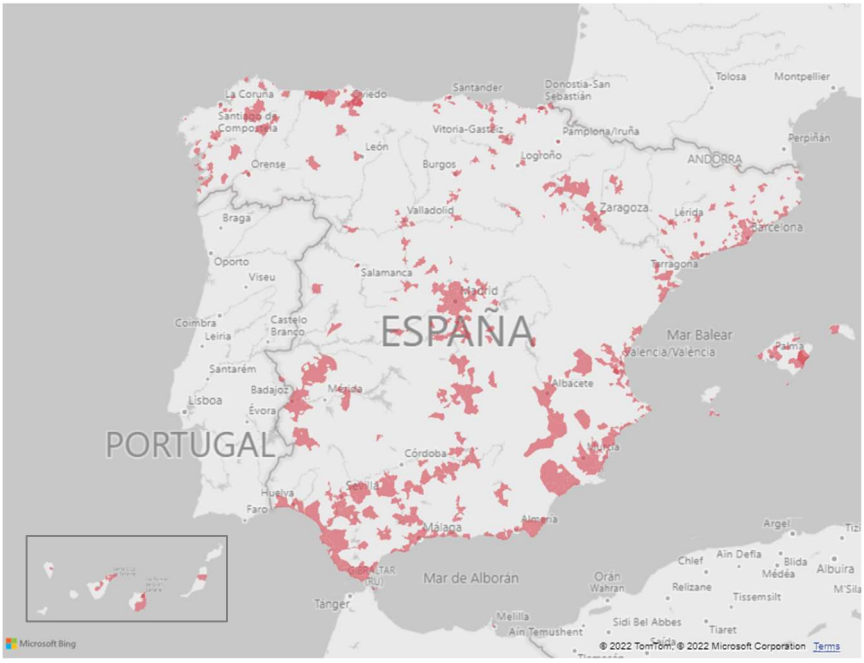


Figura 23: Distribución de los datos en España. (DatosGenerales.pbix)

Considerando el volumen de datos y la distribución geográfica de los mismos, para el resto de las fases utilizaremos los datos correspondientes a las ciudades que conforman la Comunidad Autónoma de Madrid; estos representan el 34% de los datos de España y el 6% del total de datos extraídos.

4.2 Análisis y Visualización de los datos

A partir de los datos extraídos de la Comunidad Autónoma de Madrid, hemos creado una estructura tabular o Dataframe (locations), que en adelante será el conjunto de datos a analizar. Este conjunto de datos está conformado por 166.393 registros o filas y 13 columnas que se describen en la Tabla 4.

Columna	Descripción	Tipo de Dato en origen	Tipo de Dato Dataframe
monitoring_id	identificador único generado cada vez que un usuario selecciona alguna de las funcionalidades que activa el registro de las geolocalizaciones	String	object
user_Id	Número de identificación del usuario	Número entero	int64
isAlert	Es una valor booleano (True/False) que indica si una alerta de SOS está activa. Indica que la usuario tiene una percepción de inseguridad (True) o no (False)	Booleano	bool
activate	Es una valor booleano (True/False) que indica si la alerta continúa o no activa.	Booleano	bool
latitude, longitude:	Son las coordenadas que permiten ubicar geográficamente la posición del usuario.	Número flotante	float64
UpdateAt	Fecha de registro de la localización del usuario	Timestamp	float64
radius	representa la precisión en metros del punto donde se encuentra realmente el usuario. Esta característica es dependiente de la compañía móvil del usuario.	Número flotante	float64
speed	registra la velocidad del usuario	Número flotante	float64
altitude	registra la altitud o la distancia vertical que existe entre la posición del usuario en relación con el nivel del mar.	Número flotante	float64
activity	Representa el tipo de actividad realizada.	String	object
city	Ciudad a la cual corresponde los datos extraídos. Calculada de acuerdo con la latitude y longitude	String	object
country	País a la cual corresponde los datos extraídos. Calculada de acuerdo con la latitude y longitude	String	object

Tabla 4: Conjunto de datos (dataframe)

Cuando importamos datos desde un archivo o una base de datos y los cargamos en un DataFrame, debemos tener en cuenta que estos objetos asignan un tipo de datos que puede esta sobre dimensionado, en consecuencia, podríamos estar usando más espacio en memoria del necesario. Por esta razón el primer proceso que realizaremos es una optimización de este espacio, que consiste en:

1. Eliminar datos duplicados
2. Reducir el tamaño de las columnas (`dawncast`) de tipo numérico:
    a. Convertir las columnas `radius`, `speed` y `altitude` de tipo `Float64` a `Float32`
    b. Convertir la columna `user_id` de un tipo entero `int64` a un entero sin signo `uint`
    c. Convertir columna `UpdateAt` de `float64` a `datetime`

3. Convertir las columnas `activity`, `city` y `country` de tipo `object` a tipo `category`

Antes de ejecutar las tareas de optimización del espacio, el conjunto de datos estaba conformado por 166.393 filas y 13 columnas ocupando un espacio de 67 MB. Una vez realizadas las conversiones de los datos, tenemos un nuevo conjunto de datos conformado con 162.013 filas y 13 columnas; el espacio de memoria utilizada bajo a 21.95 MB, lo cual representa una reducción del 67.27% aproximadamente del espacio utilizado. Observamos una disminución en el número de filas de 166.393 a 162.013, porque se encontraron 4.380 filas duplicadas que fueron eliminadas.

4.2.1 Visualización y Análisis de los Datos Categóricos

Los datos categóricos que analizaremos a continuación e identificados en el conjunto de datos son: `activity`, `city` y `country`.

Activity: Representa el tipo de actividad registrada por el dispositivo móvil utilizado. Se distinguen seis (6) tipos de actividad:

Activity	Descripción	Cantidad
UNKNOWN	Actividad desconocida	110.005
STILL	Inmóvil o quieto	46.449
VEHICLE	Se mueve utilizando un medio de transporte	2.814
WALKING	Caminando	2.712
RUNNING	Corriendo	17
BICYCLE	Se mueve utilizando una bicicleta	16

Tabla 5: Tipos de actividad (Análisis_Datos_ComunidadMadrid.ipynb)

En la Figura 24, observamos que:

- El 67,89% de los datos registran un tipo de actividad (UNKNOWN) o desconocida
- El 28,67% de la actividad registrada corresponde al tipo (STILL) o inmóvil
- El 1,74% de la actividad registrada corresponde al tipo (VEHICLE) o en vehículo
- El 1,67% de la actividad registrada corresponde al tipo (WALKING) o caminando

Mientras que el 0.03% restante corresponde a las actividades (RUNNING) o Corriendo y (BICYCLE) o en bicicleta

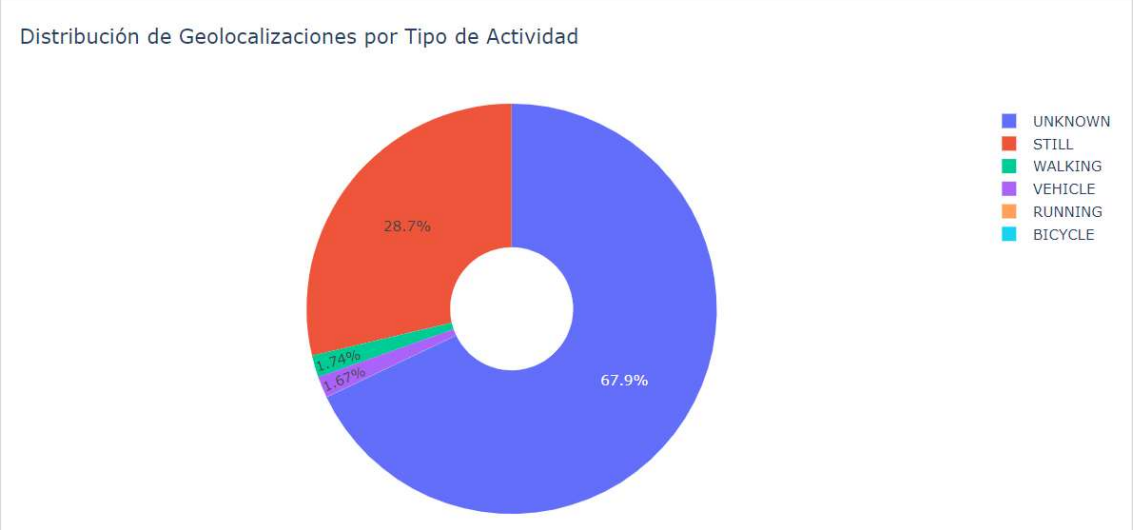


Figura 24: Distribución de geolocalizaciones o registros por tipo de Actividad (Análisis_Datos_ComunidadMadrid.ipynb)

City - Country: Representan la ciudad y el país al cual corresponden los datos extraídos. Estas columnas se calculan en base los valores de latitud y longitud. Para su cálculo utilizamos las función `IsLocatedAt` que utiliza la librería `Geopy-Nominatim` para realizar una geolocalización inversa (reverse), que retorna un diccionario con la dirección correspondiente a la latitud y longitud suministrados. Este diccionario contiene claves como: calle, ciudad, barrio, código postal, comunidad autónoma o estado, país, entre otros valores. Debido a que las claves del diccionario varían, para obtener el valor de la ciudad utilizamos las claves `town`, `city`, `state`, `county` o `country`. Si ninguna de estas claves aparece en el diccionario, se le asignará el valor de "`None_CITY`". El nombre del país lo obtenemos en el mismo proceso, utilizando el valor de la clave `country`, si esta clave no existe, se le asigna el valor `"None_COUNTRY"`. En la Figura 25, observamos la ubicación en Google Maps de las coordenadas (34.58278083041984, 17.106670774519447) , donde los valores retornados son `None_CITY` y `None_COUNTRY` porque corresponden a una ubicación en Mar Mediterráneo.

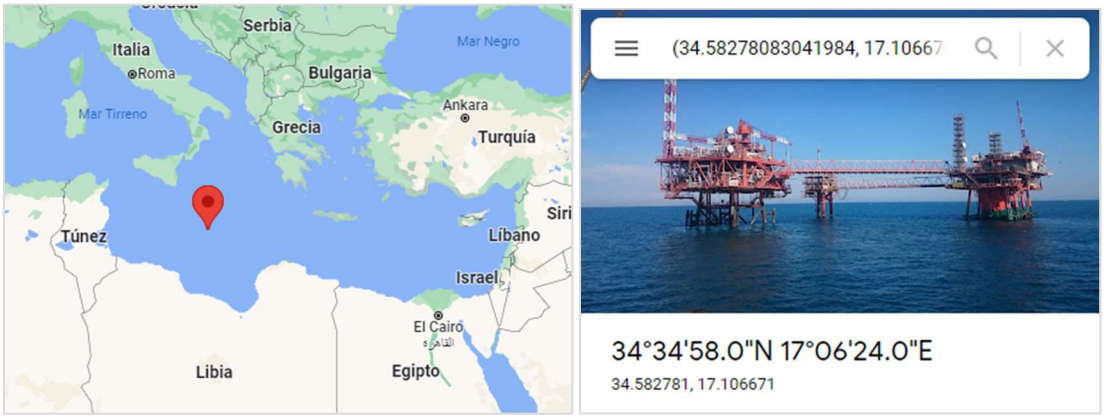


Figura 25: Google Maps coordenadas latitud = 34.582781 y longitud = 17.106671

Para filtrar los datos de las ciudades que conforman la Comunidad Autónoma de Madrid utilizamos la lista de municipios publicada en Wikipedia (Wikipedia Municipios Comunidad de Madrid, 2022). Utilizando la misma fuente de datos, obtuvimos un listado de los municipios que conforman el Área Metropolitana de Madrid publicada Wikipedia (Wikipedia - Área Metropolitana de Madrid, 2022). Esta clasificación nos facilitará la visualización de los datos.

Zona	Cantidad	%
Área Metropolitana de Madrid	124.961	77.13
Comunidad de Madrid	37.052	22.87

Tabla 6: Volumen de datos de la Comunidad Autónoma de Madrid

Los datos obtenidos corresponden a 46 ciudades que forman parte de la Comunidad Autónoma de Madrid. El 77% de los datos corresponden al Área Metropolitana de Madrid y 23% restante a otras ciudades de la Comunidad de Madrid. Dentro del Área Metropolitana, Madrid es la ciudad con mayor número de datos (86.596) que representa el 53.45%. No encontramos valores de tipo `None_CITY` en el conjunto de datos.

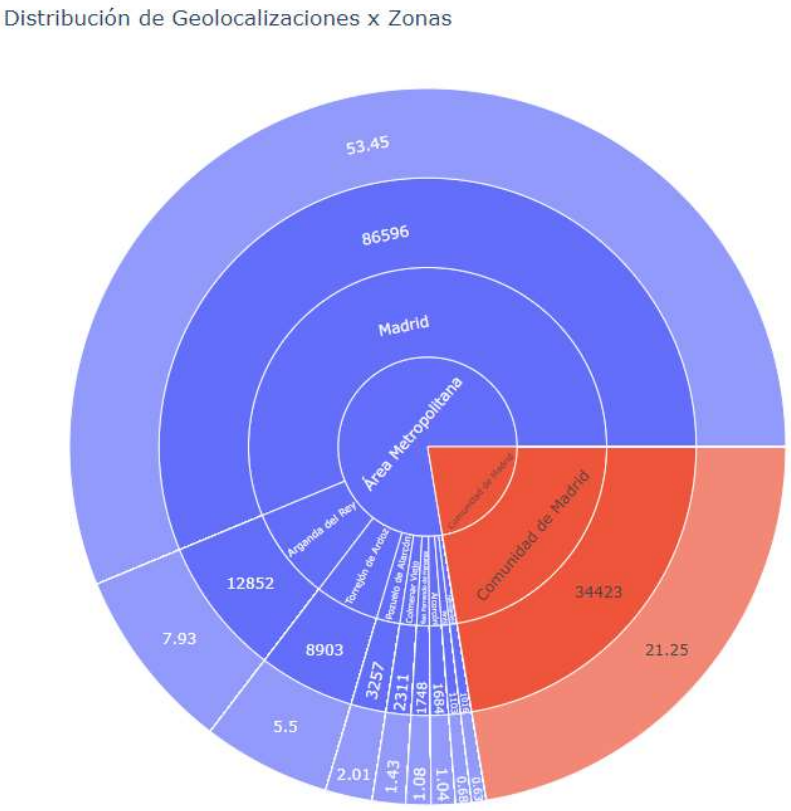


Figura 26: Distribución de Geolocalizaciones por Zonas y Ciudades (Top 10)

4.2.2 Visualización y Análisis de los Datos Numéricos

En el conjunto de datos identificamos seis (6) columnas de tipo numérico (enteros y decimales) entre los que tenemos son: `user_id`, `latitude`, `longitude`, `altitude`, `radius` y `speed`.

user_id: Es un identificador único que distingue al usuario de forma inequívoca. Considerando que los usuarios de la aplicación Sister, pertenece en su mayoría a un colectivo vulnerable, no tenemos acceso a otros datos relacionados con los usuarios.

En el conjunto de datos, observamos un total de 569 usuarios, el 70% de los usuarios (398) se ubican o registran su actividad en la ciudad de Madrid.

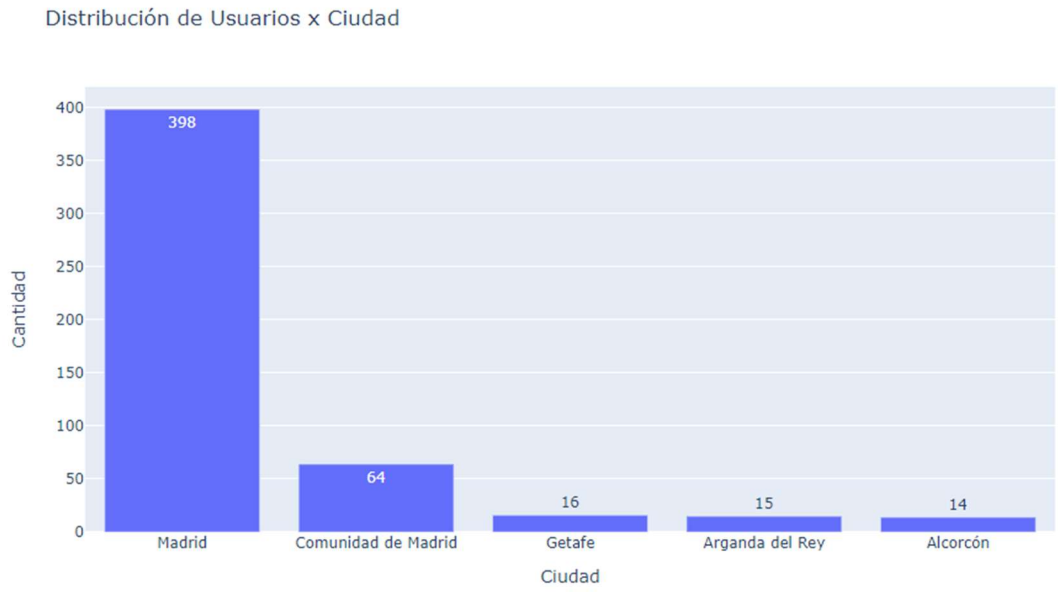


Figura 27: Distribución de Usuario por Ciudad

En la Figura 28, observamos que los usuarios que más utilizan la aplicación, tomando en cuenta la cantidad de localizaciones registradas son los usuarios: 15371 (8,5%), 2154 (8%) y 10 (6%) con más de 10.000 registros.

Figura 28: Total de Geolocalizaciones por Usuario

latitude y longitude: Son las coordenadas que permiten ubicar geográficamente la posición de un usuario. Para analizar estos los datos de geolocalización, hemos considerado que el centro de la Comunidad Autónoma de Madrid se encuentra en la latitud 40.4167 y longitud -3.7167. Para obtener estas coordenadas, hemos utilizado la función `geo_cc.get_city_geolocation`, la cual extrae del conjunto de datos **worlcities.csv** disponible en Kaggle (Kaggle - World Cities, 2022) las coordenadas de una ciudad; en nuestro caso de la ciudad de Madrid. Estos valores los hemos almacenado en las constantes: CITY_LAT, CITY_LNG que utilizaremos para resaltar o acercar esta zona en los mapas.

Utilizando los datos suministrado en la página **Openstretmap** obtuvimos los límites de la Comunidad Autónoma de Madrid, donde tenemos que la latitud se encuentra entre las coordenadas 41.4427 y 39.6015 y la longitud entre las coordenadas -5.9711 y -1.6589 (Ver Figura 29). Estas coordenadas nos permitirán validar si los valores de latitud y longitud están dentro de estos límites.

Figura 29: Límites de la Comunidad de Madrid.
Fuente https://nominatim.openstreetmap.org/ui/details.html?osmtype=R&osmid=6426653&class=boundary

En pruebas realizadas encontramos algunas coordenadas que no se correspondían con la ciudad asignada o tenían un valor "None_CITY", porque en el proceso de geolocalización inversa no se encontraron valores asignados en las claves `town`, `city`, `state`, `county` y `country` que utilizamos para identificar la ciudad. Luego de analizar algunos de los datos que presentaban este error, incluimos una nueva columna `state_district` que nos permite identificar las islas; por ejemplo: La Palma y Canarias.

Al representar en un mapa de calor todas los valores de latitude y longitude del conjunto de datos de la Comunidad Autónoma de Madrid (Ver Figura 30) , observamos algunas coordenadas fuera de los limites:

1. Las coordenadas (34.58278083041984, 17.106670774519447) que corresponde a un punto en el Mar Mediterráneo.
2. Las coordenadas (37.330583, -122.029247) que corresponde a Mariani Ave, Cupertino, CA 95014, EE.UU.
3. Las coordenadas (37.253447, -6.939273 y 37.212340, -7.051987) que corresponde a Huelva, España.

4. Las coordenadas (4.722579,-74.258621 y 4.735678, -74.255079) corresponde a la ciudad de Madrid en Cundinamarca Colombia.

También observamos coordenadas de localidades de Jaén y Sevilla, que han sido clasificadas como localidades de Madrid.

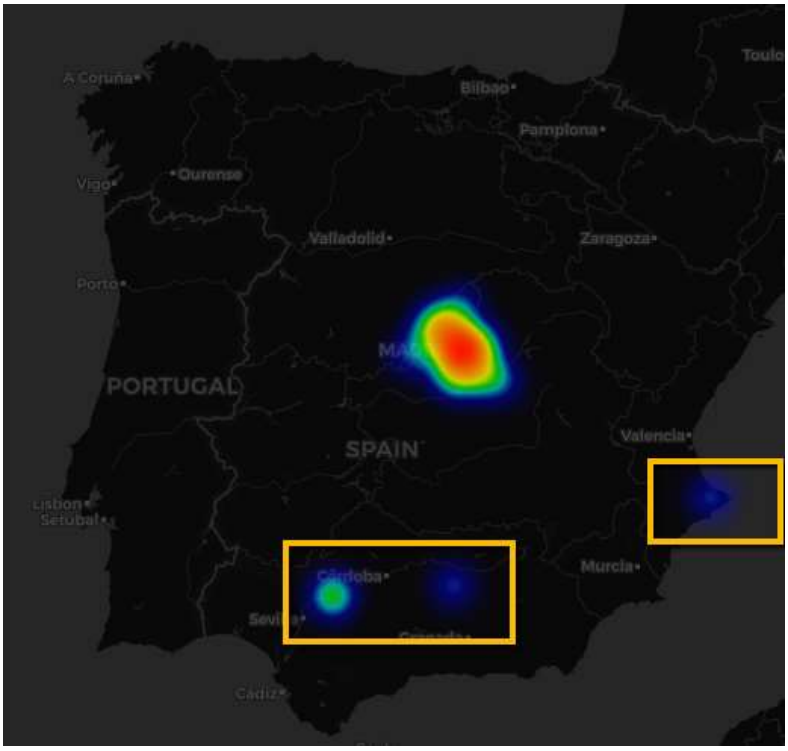


Figura 30: Mapa de calor valores latitud y longitud de los datos de la Comunidad Autónoma de Madrid. Las zonas marcadas en amarillo, corresponde a valores fuera de rango.

Considerando que hemos detectado coordenadas que no corresponde a ciudades de la Comunidad Autónoma de Madrid, fue necesario ejecutar un proceso para eliminar los datos fuera de estos límites. Para ello, utilizando la función `selDataset` que nos permite extraer los datos que corresponda a la Comunidad Autónoma de Madrid, es decir que la latitude está entre las coordenadas `41.4427` y `39.6015` y la longitud entre las coordenadas `-5.9711` y `-1.6589`.

```
locations = geo.selDataset(locations, 41.4427, -5.9711, 39.6015, -
1.6589)
```

Luego de aplicar la función `selDataset`, observamos que se han descartado 86 registros. El conjunto de datos tiene 161.927 filas y 15 columnas; porque el proceso incorpora las columnas Geometry y InZone. La columna Geometry almacena un objeto con las coordenadas de latitud y longitud para utilizar métodos de GeoPandas con el sistema estándar de coordenadas WGS

84[12]. La columna InZone es una columna de tipo entero donde 1 indica que pertenece a una ciudad de la Comunidad Autónoma de Madrid y 0 que no pertenece.

Una vez eliminados estos datos, generamos nuevamente el mapa de calor donde no se observan localizaciones fuera de los límites (Ver Figura 31).

Figura 31: Mapa de calor valores de latitud y longitud de los datos de la Comunidad Autónoma de Madrid, luego de eliminar los datos fuera de los limites

altitude: Representa la altitud o la distancia vertical que existe entre la posición del usuario en relación con el nivel del mar. Esta característica es dependiente de la compañía móvil del usuario. La Figura 32**¡Error! No se encuentra el origen de la referencia.**, nos muestra algunos valores estadísticos de la columna altitude:

[12] El WGS 84 (World Geodetic System 1984) es un sistema geodésico de coordenadas geográficas usado mundialmente, que permite localizar cualquier punto de la Tierra (sin necesitar otro de referencia) por medio de tres unidades dadas (x,y,z). (WGS84 - Wikipedia, la enciclopedia libre)

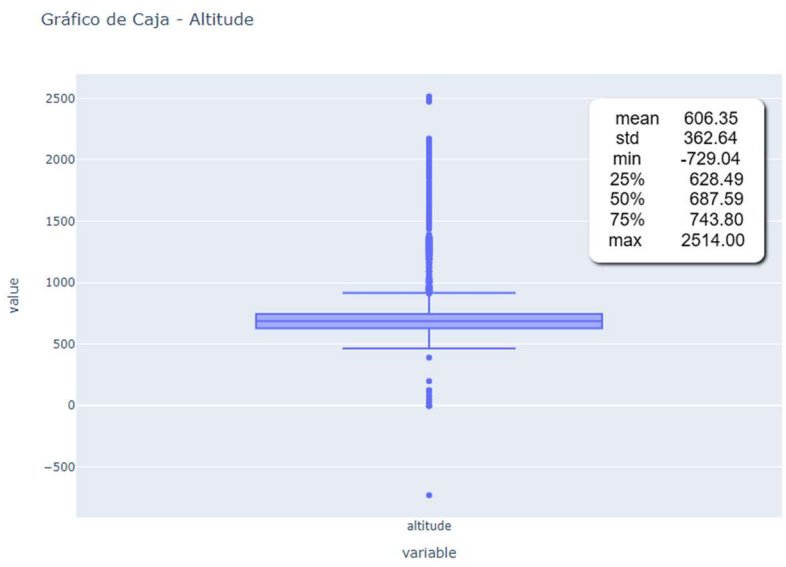


Figura 32: Valores estadísticos de la columna altitude

De acuerdo con la información publicada en Topographic Map (Topographic Map, 2022) en Madrid tanto en el Área Metropolitana y el Corredor del Henares los valores de altitud van desde los 519 m hasta los 1.829 m

```
Nombre: Mapa topográfico Madrid, altitud, relieve.
Coordenadas: 40.31198 -3.88895 40.64373 -3.51792

        Altitud mínima:   519 m
        Altitud media:    715 m
        Altitud máxima: 1.829 m
```

Tabla 7: datos Topográficos de Madrid. Fuente: https://es-es.topographic-map.com/maps/6o29/Madrid/

Si comparamos los valores estadísticos mínimo (-729 m) y máximo (2.514 m) de la columna altitud con los datos de la Tabla 7; observamos que existen valores fuera de los rangos del mapa topográfico. Con la finalidad de evitar que estos valores atípicos distorsiones los resultados, actualizaremos los valores de las altitudes que están fuera de rango utilizando la función `get_elevation()`, esta función realiza un petición (Get) al sitio web `https://api.open-elevation.com/api/v1/lookup'` enviando como parámetros la latitud y la longitud y retorna la altitud correspondiente esas coordenadas.

Luego de actualizar las altitudes fuera de rango, observamos que el valor mínimo es de 501 m, el máximo 1.818 m y la altitud media es de 755,97 metros de altura:

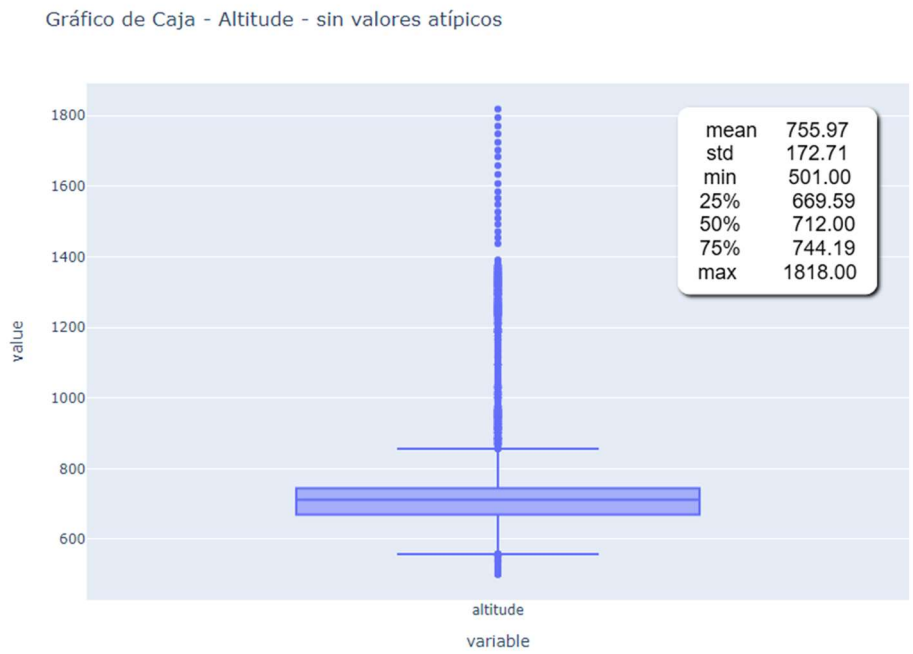


Figura 33: Gráfico de Caja de la columna altitude, después de eliminar los datos atípicos.

Aun cuando los valores han mejorado, esta columna puede tener valores que no son confiables, una posible mejora es actualizar este valor al momento de realizar la extracción de los datos o aplicar la función `get_elevation` a todo el conjunto de datos.

radius: Esta columna representa la precisión en metros del punto donde se encuentra realmente el usuario. Esta característica es dependiente del dispositivo utilizado y la compañía móvil del usuario. La Tabla 8**¡Error! No se encuentra el origen de la referencia.**, nos muestra algunos valores estadísticos de la columna radius:

mean	6.961189e+01
std	9.967138e+03
min	-2.147472e+06
25%	6.097288e+00
50%	1.164000e+01
75%	1.768000e+01
max	1.089531e+06

Tabla 8: Valores estadísticos de la columna radius

Observando los resultados, vemos que los valores mínimo y máximo de la columna son valores muy extremo (-2.147.472 y 1.089.531), lo cual distorsionan la muestra, porque el 75% de los valores se encuentran muy por debajo de la media. Es importante destacar que al realizar un filtrado de estos valores atípicos, observamos que corresponden a un único usuario el 10671.

Al calcular estos valores estadísticos excluyendo al usuario 10671, obtenemos una media de 32 metros de precisión; el valor mínimo cambia de -2.147.435,75 a 1 y el máximo de 1.068.749,25 a 4.334,58. Si bien es cierto, que una precisión del 100% en geoposicionamiento

no existe y que la misma depende del dispositivo utilizado y de los métodos de cálculo del geoposicionamiento que estos utilicen (GPS, Antenas de Telefonía móvil o Wifi). Los valores de la columna radius para este usuario parecen ser muy altos (> a 1 km), si lo comparamos con la última media y desviación estándar obtenida. En esta fase de análisis y visualización no ejecutaremos ninguna otra acción correctiva sobre esta columna.

speed: registra la velocidad del usuario, al momento de registrar la geolocalización. Para analizar la columna speed nos hemos basado en un estudio del Reino Unido de 2011 en el cual rastrearon a 358 participantes de varias edades utilizando acelerómetros para determinar sus velocidades medias de caminata. Este estudio indica que las velocidades dependen de variable como el sexo, la edad y la altura. (Hippocrates Guild, 2020)

Tomando como base las velocidades medias de caminata confortable por sexo del estudio. determinamos que una persona puede caminar de forma confortable a una velocidad media de 4.90 km/h si es mujer y de 5.04 km/h si es hombre. Por otra parte, podemos considerar que la velocidad máxima de carrera es la correspondiente al récord de velocidad de un humano que lo tiene Usain Bolt y es de 44.72 km/h.

La Tabla 9, muestra los límites máximos de velocidad por tipo de vehículo. Para determinar estos límites máximos, nos basamos en los datos publicados en la página web de la Dirección General de Tráfico (DGT) que hacen referencia al RD 1514/2018 o articulo 48 RCC. (Revista DGT, 2022)

Tipo de vehículo	Máxima
Vehículo Turismo	50 - 120
Bicicleta	45

Tabla 9: Límites de velocidad por tipo de vehículo.

Los datos estadísticos de la columna speed (Ver Tabla 10), nos muestran que la velocidad mínima es de 0 km/h y la máxima 87.50 km/h; si consideramos que esta columna está asociada a actividades como caminar, correr e inclusive trasladarse en un medio de transporte (coche o bicicleta) y tomando como base las velocidades medias de una caminata confortable (4.90 km/h si es mujer y de 5.04 km/h si es hombre) o los límites de velocidad de un vehículo en una autovía (80 - 120 km/h). Podemos concluir que los valores mínimos y máximos de la columna speed indican velocidades dentro de los límites, es decir, no observamos valores atípicos.

También observamos que el 75% de los datos registran velocidades menores a 2 km/h, lo cual puede indicar que el 75% de los usuarios prefiere caminar.

```
mean          2.626555
std           5.988488
min           0.000000
25%           0.003047
50%           0.569104
75%           1.388889
max          87.500008
```

Tabla 10: Valores estadísticos de la columna speed

Para contrastar esta información veamos en la Tabla 11, las velocidades registradas por tipo de actividad:

Tipo de Actividad	min	mean	max
VEHICLE	0.0	11.856548	39.411674
UNKNOWN	0.0	2.711101	87.500008
STILL	0.0	2.051866	47.580002
BICYCLE	0.0	0.000000	0.000000
RUNNING	0.0	0.000000	0.000000
WALKING	0.0	0.000000	0.0000

Tabla 11: Velocidades registradas por tipo de actividad

Observamos que las velocidades no concuerdan con el tipo de actividad registrada. Por ejemplo: la velocidad media y máxima para los tipos de actividad BICYCLE, WALKING y RUNNING es cero. Por otra parte, observamos que las velocidades registradas en el tipo de actividad STILL no corresponden a una persona que está parada.

4.2.3 Análisis de Datos Booleanos (True/False)

isAlert: La columna isAlert indica si el usuario de la aplicación ha activado la funcionalidad SOS. Si el valor es True indica que el usuario puede estar en peligro o tiene la percepción de peligro. Está opción graba videos que se guardan y luego pueden ser solicitados como pruebas.

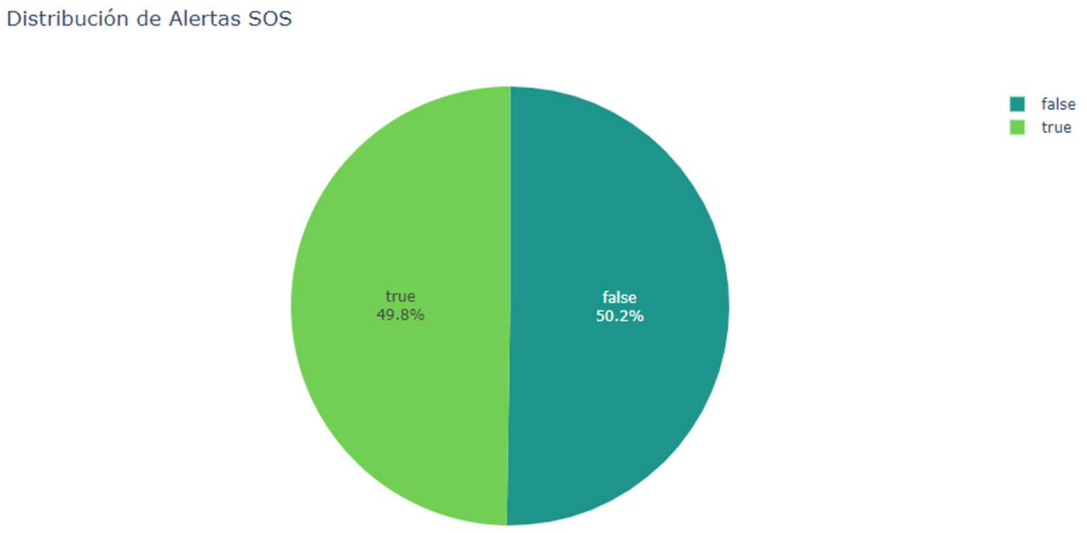


Figura 34: Distribución de Alertas

El 49.80% de los datos (80.576) corresponde a geolocalizaciones donde los usuarios han activado un alerta SOS, estos datos seguramente nos ayudarán a identificar o clasificar una zona como insegura. En la Figura 35, observamos con marcadores rojos las zonas o coordenadas donde un usuario activó un alerta SOS.

Figura 35: Mapa de Madrid. Los marcadores rojos indican la activación de un alerta SOS.

4.2.4 Análisis de Datos Datetime

UpdateAt: Fecha de registro de la localización del usuario. Que corresponde a la hora del meridiano de Greenwich (UTC – Tiempo Universal Coordinado) que se utiliza como hora estándar. Esta columna en los datos originales es un timestamp, que registra el número de segundos que han transcurrido desde las 0 horas del 1 de enero de 1970 GMT. En la fase de optimización del espacio la convertimos a fecha, para facilitar el análisis de los datos.

Si verificaremos los valores mínimos y máximos de la columna UpdateAt, observamos que los datos de la Comunidad Autónoma de Madrid, corresponde al periodo entre el 02 de octubre del 2022 hasta el 27 de mayo del 2022.

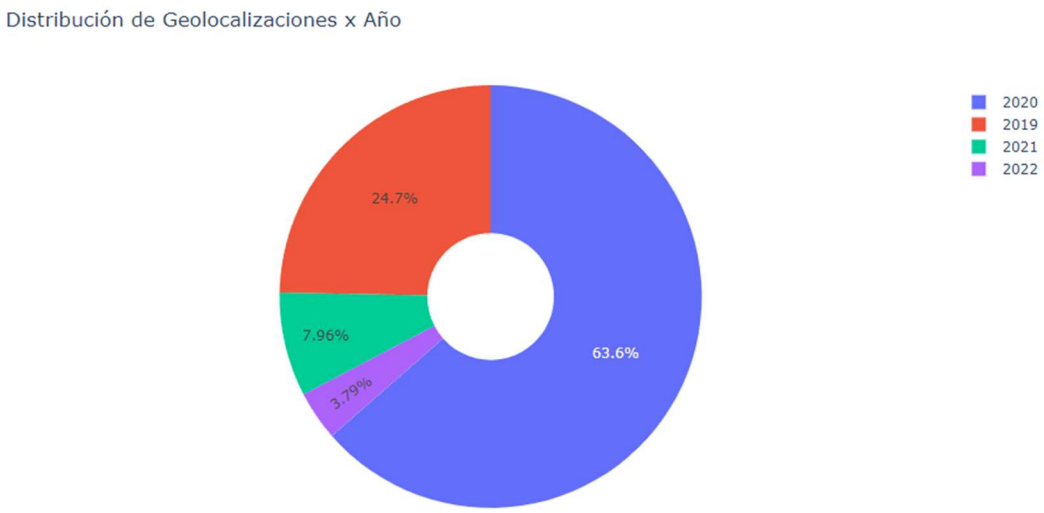


Figura 36: Distribución de los datos por Año

En la Figura 36, observamos que el 63.60% de los datos corresponden al año 2020, un 24.70% al año 2019 y el 11.70% restante a los años 2021 (7.96%) y 2022 (3.79%).

En los meses de febrero y septiembre del año 2020 y en octubre del 2019 observamos un alto volumen de geolocalizaciones. En los dos (2) últimos años, observamos una disminución en el volumen de geolocalizaciones (ver Figura 37).

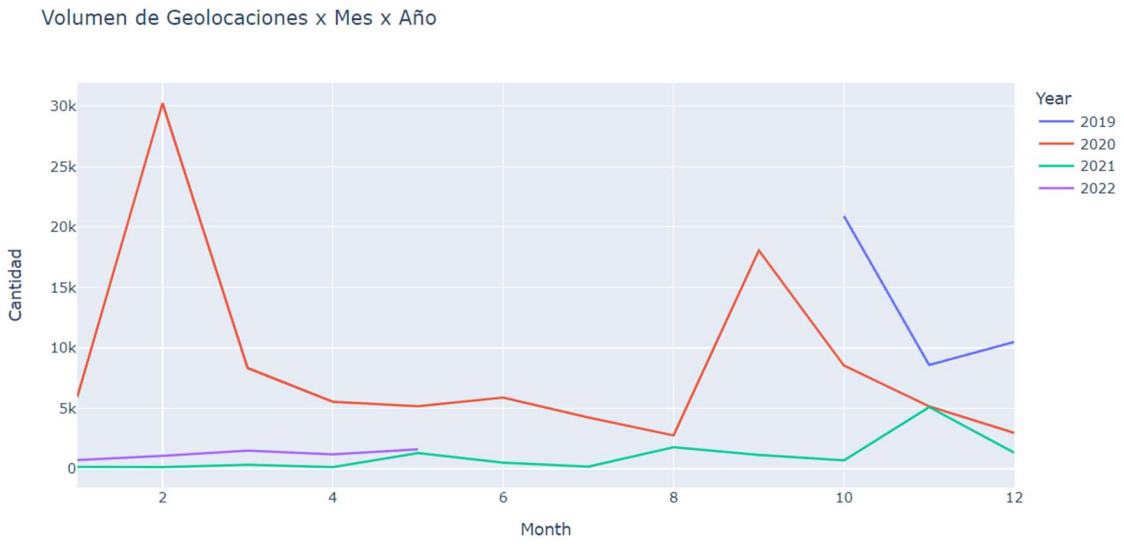


Figura 37: Volumen de Geolocalizaciones x Mes x Año

4.2.5 Análisis de Datos tipo Object

monitoring_id: Es un identificador único creado cada vez que un usuario selecciona alguna de las siguientes funcionalidades: compartir su ubicación, pulsar el botón SOS, activar la localización que es compartida con sus usuarios de confianza, entre otras.

Al activar una funcionalidad y/o un alerta SOS, la aplicación comienza a registrar datos como: usuario, latitud, longitud, altitud, radio, velocidad, fecha y hora, actividad de cada registro. Esta información se sigue almacenando, hasta que el usuario detiene la opción de compartir indicando que ha llegado bien, desactiva el alerta o alcanza el tiempo máximo (20 minutos en el plan gratuito y 1 día en el plan premium).

En resumen un monitoring_id contiene todos los movimientos del usuario durante el tiempo que este mantiene activa un alerta y/o funcionalidad permitida por la aplicación de acuerdo con el plan contratado. En nuestro conjunto de datos existen 1.643 identificadores únicos de monitoreo y 161.927 registros de localizaciones.

La Tabla 12, muestra todos los registros generados al activar una funcionalidad, en este ejemplo el usuario no activo un alerta SOS. Los registros fueron almacenados bajo el id "a9e7a8fe95b34765920344707c760f6a":

user_id	isAlert	latitude	longitude	altitude	radius	UpdateAt	activity	Speed
59883	False	40.393805	-3.658126	646.0	128.0	2021-11-20 20:09:25.250	UNKNOWN	0.0
59883	False	40.393427	-3.660121	640.0	25.0	2021-11-20 20:10:05.258	WALKING	0.0
59883	False	40.393427	-3.660121	640.0	25.0	2021-11-20 20:10:06.171	WALKING	0.0
59883	False	40.393327	-3.660194	640.0	25.0	2021-11-20 20:10:15.265	WALKING	0.0
59883	False	40.393263	-3.660230	640.0	25.0	2021-11-20 20:10:23.218	WALKING	0.0
59883	False	40.393229	-3.660219	640.0	25.0	2021-11-20 20:10:25.651	WALKING	0.0
59883	False	40.393229	-3.660219	640.0	25.0	2021-11-20 20:10:26.189	WALKING	0.0
59883	False	40.393229	-3.660219	640.0	25.0	2021-11-20 20:10:26.191	WALKING	0.0
59883	False	40.393150	-3.660243	640.0	25.0	2021-11-20 20:10:34.011	WALKING	0.0
59883	False	40.393120	-3.660262	640.0	25.0	2021-11-20 20:10:35.258	WALKING	0.0
59883	False	40.393036	-3.660310	640.0	25.0	2021-11-20 20:10:45.259	WALKING	0.0
59883	False	40.393033	-3.660322	640.0	25.0	2021-11-20 20:10:55.254	WALKING	0.0
59883	False	40.393019	-3.660314	640.0	25.0	2021-11-20 20:11:05.261	WALKING	0.0

Tabla 12: Datos registrados para el monitoring_id "a9e7a8fe95b34765920344707c760f6a"

De acuerdo con los datos de la Tabla 12, el usuario 59883 parte del punto A (40.393805, -3.658126) el 20 de noviembre del 2021 a las 20:09 y llega al punto (40.393019, -3.660314) el mismo día a las 20:11. Recorre caminando una distancia estimada de 224,47 metros en aproximadamente 2 minutos y una velocidad estimada de 8 km/h. La Figura 38 nos muestra la ruta creada utilizando la librería Folium de Python (izquierda) y la ruta generada con Google maps.

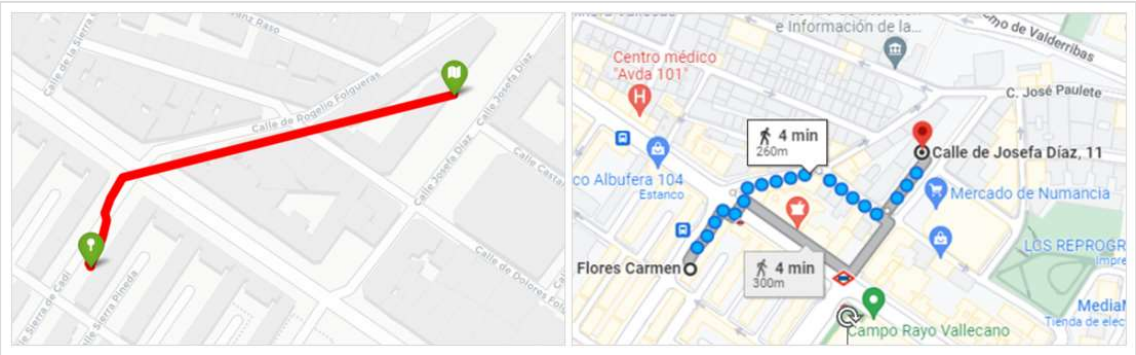


Figura 38: Ruta del usuario 59883: A la izquierda la ruta calculada a partir de los datos recopilados utilizando la librería Folium de Python, a la derecha la misma ruta utilizando Google maps.

4.3 Correlación de Variables

La matriz de correlación es una estructura tabular que representa las "correlaciones" entre dos variables. La utilizamos para comprender la relación entre la diversas variables y toma decisiones en base a esa relación. El número que aparece en cada cuadro de nuestra matriz representa la fuerza de la relación entre dos variables. Aunque existen diversos métodos de calcular la matriz de correlación (coeficiente de Spearman o el coeficiente de correlación de Kendall Tau), en este caso hemos utilizado la correlación de Pearson.

Un valor positivo cercano a 1.0 indica una fuerte correlación positiva, es decir que si el valor de una variable aumenta la otra también. Mientras que un valor negativo cercano a -1.0 indica una fuerte correlación negativa, es decir que si el valor de una variable disminuye la otra también. Por otra parte, un valor cercano a 0, ya sea positivo o negativo indica la ausencia de correlación entre las dos variables y por tanto podemos decir que esas variables son independientes entre sí.

En la matriz de correlación (ver Figura 39), no se observan correlaciones positivas fuertes (> 0.5), existen correlaciones positivas débiles (< 0.5) entre las variables `latitude` y `altitude` y entre `isAlert` y `longitude`, `latitude` y `altitude`. También se observa un correlación negativa fuerte entre las variables `usuario` y `radius`. Estas correlaciones tienen sentido, porque cuando el usuario activa un alerta, la aplicación registra las coordenadas de geolocalización del usuario (latitud, longitud, altitud y radio representa la precisión en metros del punto donde realmente se encuentra el usuario).

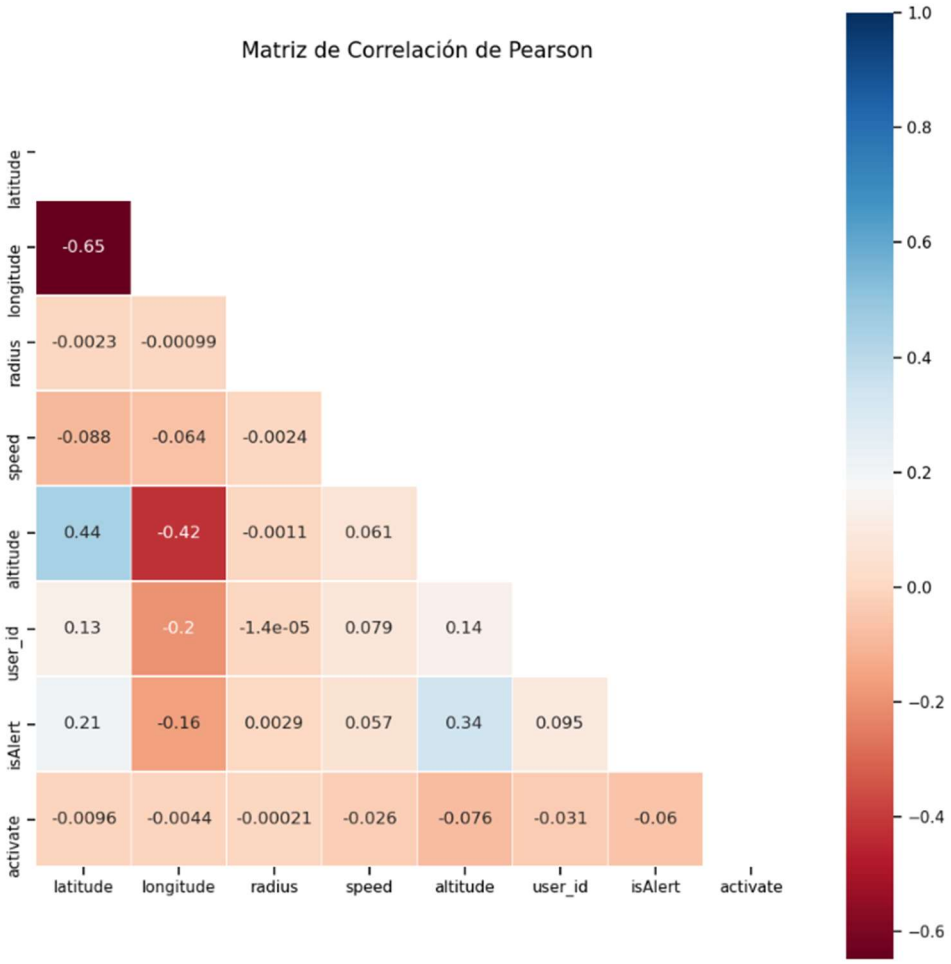


Figura 39: Matriz de Correlación de Pearson

5. Modelo de Trayectoria de Usuario

Cuando un usuario hace uso de la aplicación Sister, se registran bajo un único identificador todas las geolocalizaciones del usuario durante 20 minutos (plan básico), durante un día (plan premium) o hasta que el usuario desactive la funcionalidad validando que ha llegado a su destino o la percepción de peligro desaparece.

En resumen, un identificador de monitoreo (monitoring_id) contiene todos los movimientos del usuario durante el tiempo que mantuvo activa un alerta SOS y/o otra funcionalidad. Cuando se activa un alerta SOS el valor de la columna isAlert es igual a True., de lo contrario es False.

Funcionalidad	isAlert
Alerta SOS	True
Funcionalidad**	False
Funcionalidad + Alerta SOS**	True

Tabla 13: Valores de la columna isALert

El nuevo modelo de datos se crear luego de agrupar todas las localizaciones por el identificador de monitoreo. El nuevo conjunto de datos tiene un total de 1.643 filas o registros y 25 columnas (Ver Tabla 1Tabla 14).

Columna	Descripción	Tipo de Dato Origen	Tipo de Dato Dataframe
monitoring_id	identificador único generado cada vez que un usuario selecciona alguna de las funcionalidades que activa el registro de las geolocalizaciones	object	object
user_Id	Número de identificación del usuario	uint32	uint32
isAlert	Es una valor booleano (True/False) que indica si una alerta de SOS está activa.	bool	bool
activate	Es una valor booleano (True/False) que indica si la alerta continúa o no activa.	bool	bool
lat_sp, long_sp	Calculado a partir de las coordenadas (latitude, longitude) del primer punto (inicio del trayecto).	-	float32
lat_fp, long_fp	Calculado a partir de las coordenadas (latitude, longitude) del último punto (fin del trayecto).	-	float32
distance	Se calcula en base a la distancia entre cada dos puntos y esta expresada en km	-	float32
start_date	Calculado a partir de la fecha (UpdateAt) del primer punto de la localización del usuario	-	datetime64
finish_date	Calculado a partir de la fecha (UpdateAt del último punto de la localización del usuario	-	datetime64
radius	representa la precisión en metros del punto donde se encuentra realmente el usuario.	float32	float32
time	Se calcula en base a la suma de las diferencias entre la fecha de inicio y fin de dos puntos.	-	deltatime64
t_hours	Calculado a partir de la columna time, la cual es convertido en horas.	-	float32
estimated_speed	Calculado utilizando la formula distance/t_hours. La velocidad está expresada en km/h	-	float32
altitude	registra la altitud o la distancia vertical que existe entre la posición del usuario en relación con el nivel del mar.	float32	float32
activity	Se asigna la actividad más frecuente de todos los registros de un identificador de monitoreo	category	category

Columna	Descripción	Tipo de Dato Origen	Tipo de Dato Dataframe
city	Ciudad a la cual corresponde los datos extraídos.	category	category
country	País a la cual corresponde los datos extraídos.	category	category
isMetropolitan	Indica si la ciudad pertenece al Área Metropolitana de Madrid	bool	Bool
Year	Año al cual corresponde el registro	uint32	uint32
Month	Mes al cual corresponde el registro	uint32	uint32
dayWeek	Día de la semana, calculado a partir de la columna start_date	-	category
isWeekend	Indica si el registro ocurrió en fin de semana (True), calculado a partir de la columna start_date	-	bool
isDay	Indica si el registro ocurrió durante día (True) o luego de la puesta del sol o Noche (False). Calculado a partir de la columna start_date, start_point y end_point	-	Bool
geo_points	Contiene las coordenadas desde el punto de inicio y fin sin repeticiones. Esta columna la utilizaremos para dibujar la trayectoria.	-	object
polygon	Es un objeto tipo POLYGON con todas las coordenadas. Esta columna la utilizaremos para dibujar un polígono con todas las coordenadas de la trayectoria.	-	object

Tabla 14: Conjunto de datos Trayectoria de los Usuarios (dataframe)

En la Figura 40, observamos algunas trayectorias dibujadas a partir de los datos del nuevo modelo. A la izquierda la trayectoria de los usuarios utilizando vectores, a partir de la lista de puntos de la columna `geo_points`. A la derecha las trayectorias utilizando polígonos a partir de la columna `polygon`.

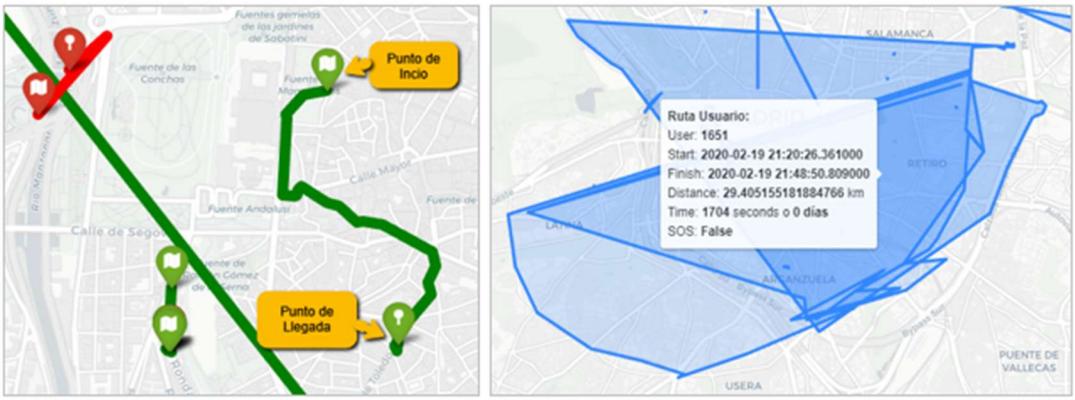


Figura 40: Trayectorias de usuario: A la izquierda utilizando vectores. A la derecha utilizando polígonos

Una vez creado el modelo debemos validar la calidad de los datos, para ello analizaremos los datos estadísticos de las columnas de tipo numérico:

	Distance	Time	Radius	Estimated Speed	Altitude	t_hours
mean	13,74	1 days	-1,24E+09	7,026780	705,554510	31,436765
std	453,12	11 days	5,30E+10	37,629459	81,833079	282,840005
min	0,00	0 days	-2,15E+12	0,000000	518,000000	0,000000
25%	0,00	0 days	1,28E+07	0,041675	659,204834	0,005556
50%	0,026951	0 days	1,60E+07	0,858456	699,000000	0,030278
75%	0,256158	0 days	6,50E+07	3,205919	734,000000	0,221806
max	18355,741	225 days	3,24E+09	873,278811	1313,824585	5418,482778

Tabla 15: Valores estadísticos de los datos numéricos

En la Tabla 15, podemos ver que la columna `altitude` se mantiene dentro de los rangos topográficos de la Comunidad Autónoma de Madrid. Sin embargo, como era de esperarse se observan valores atípicos en las columnas `distance`, `time`, `estimated_speed`, `t_hours` y `radius`.

- La columna `distance` muestra valores atípicos como distancias de 18.355 km
- La columna `time` muestra tiempos superiores a un día (entre 3 y 225 días), cuando la aplicación limita la monitorización a 20 min a usuarios del plan básico y 1 día a usuario de plan premium.
- La columna `estimated_speed` tiene valores superiores a 120 km/h , observamos velocidad muy altas entre 157 y 873 km/h.
- La columna `radius` muestra valores negativos y valores superiores a 3 km.
- La columna t_hours muestra valores atípicos como tiempo de 5.418 horas.

Considerando que la aplicación guardar los datos de máximo un día, procederemos a eliminar todos los registros con tiempos (time) superiores a un día, porque a priori parecieran un error de la aplicación y además muestran valores incongruentes con la realidad. Por ejemplo: una persona no puede realizar un trayecto de 42 km en 3 minutos, porque necesitaría trasladarse en un vehículo a una velocidad superior a los 800 km/h.

	Distance	Time	Radius	Estimated Speed	Altitude	t_hours
count	1.553	1.553	1.553	1.553	1.553	1.553
mean	1,43	0 days	-1,320198e+03	5,10	705,12	1,01
std	5,66	0 days	5,449425e+04	14,16	80,31	4,81
min	0,00	0 days	-2,147436e+06	0,00	518,00	0,00
25%	0,00	0 days	1,282200e+01	0,08	660,00	0,00
50%	0,02	0 days	1,600000e+01	0,97	699,69	0,02
75%	0,19	0 days	6,500000e+01	3,41	734,00	0,16
max	85,89	1 days	3,239598e+03	157,50	1313,82	47,93

Tabla 16: Valores estadísticos de los datos numéricos, luego de eliminar los datos atípicos

Una vez eliminados estos datos atípicos (90 registros), observamos que la mayoría de las columnas muestran valores dentro de rangos aceptables (Ver Tabla 16). Excepto las columnas `estimated_speed` y `radius.` En la columna `estimated_speed` observamos que el valor máximo 157,50 km/h; supera los límites de velocidad permitidos para un vehículo en una autovía de 120 km/h, pero lo consideraremos como aceptable. La columna `radius` presenta valores atípicos extremadamente bajos y altos, eliminaremos aquellos registros con valores que están fuera de los limites mínimos (Q1) y máximos (Q3). La Figura 41 nos muestra un diagrama de caja o bigotes para visualizar los valores antes y después de eliminar los valores atípicos.

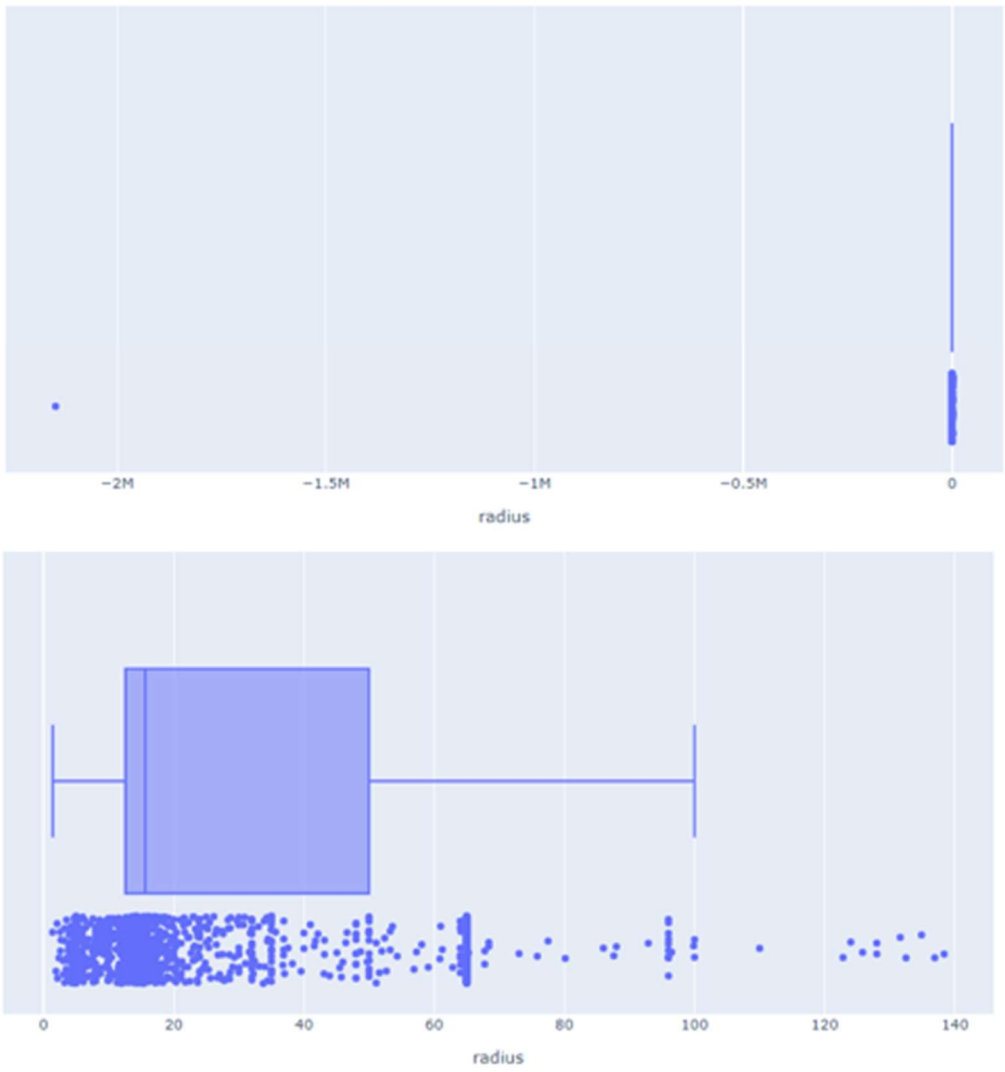


Figura 41: Diagrama de caja de la columna radius. Arriba: Muestra los datos con los valores atípicos o fuera de los límites Q1 y Q3. Abajo: Muestra los datos una vez eliminados los valores fuera de rango.

5.1 Análisis y Visualización de los Datos del Modelo

El modelo de datos de las trayectorias del usuario tiene un total de 1.553 registros y 25 columnas. El 92% de los datos corresponden al Área metropolitana de Madrid, siendo Madrid la ciudad con mayor número de registro el 71,53% de los datos.

Los registros muestran que la aplicación muestra una utilización del 70% en el día y un 30% en la noche 30%. El uso de la aplicación de lunes a viernes es de un 81% y un 19% durante el fin de semana. Mientras que los días de la semana en los cuales se utiliza más la aplicación son los viernes (20,90%), jueves (20,70%) y miércoles (14,40%).

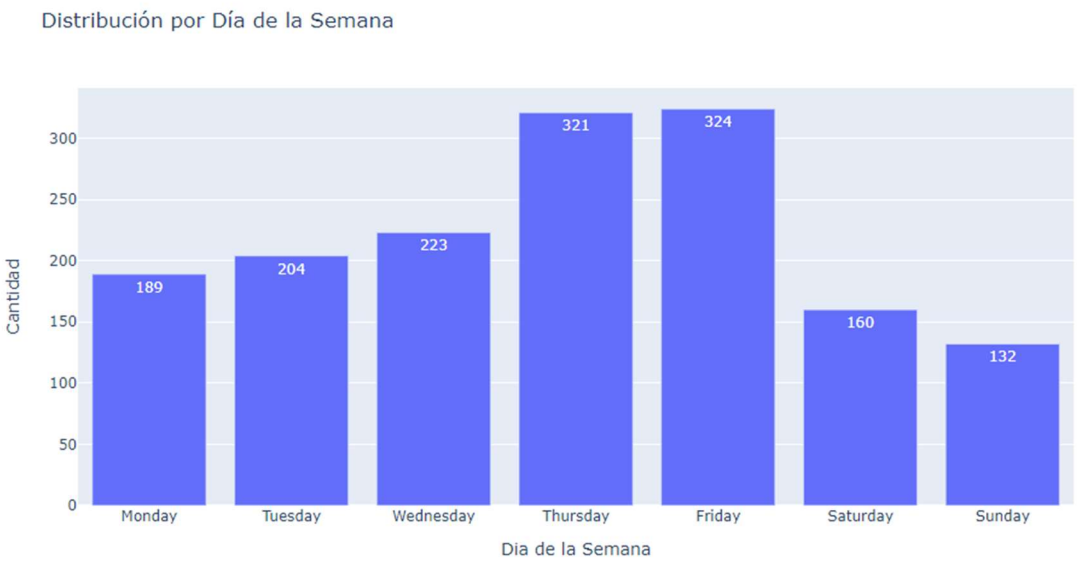


Figura 42: Distribución los registros x día de la semana

Si clasificamos los registros de acuerdo con el tipo de funcionalidad, considerando dos tipos de funcionalidades: el primero cuando se activar un alerta SOS y el segundo cuando se utiliza cualquiera de las funcionalidades sin activar un alerta SOS. De acuerdo con esta clasificación, el 56% de los registros corresponden al uso de otras funcionalidades (sin alarma), mientras que el 44% corresponden a los registros generados por alertas SOS.

Los siguientes gráficos (ver Figura 43), nos muestran las distribución de los datos de acuerdo con el tipo de funcionalidad utilizada, el día de la semana y el área geográfica donde se registran.

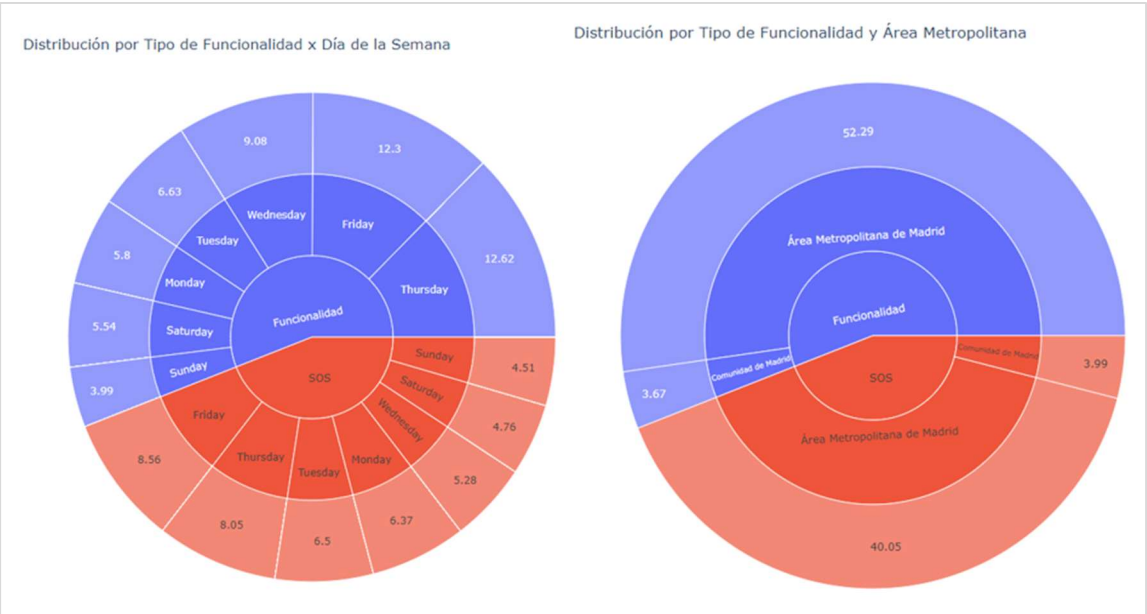


Figura 43: Izquierda: Distribución de los datos por Funcionalidad y día de la semana. Derecha: Distribución de los datos por funcionalidad y Área Metropolitana de Madrid.

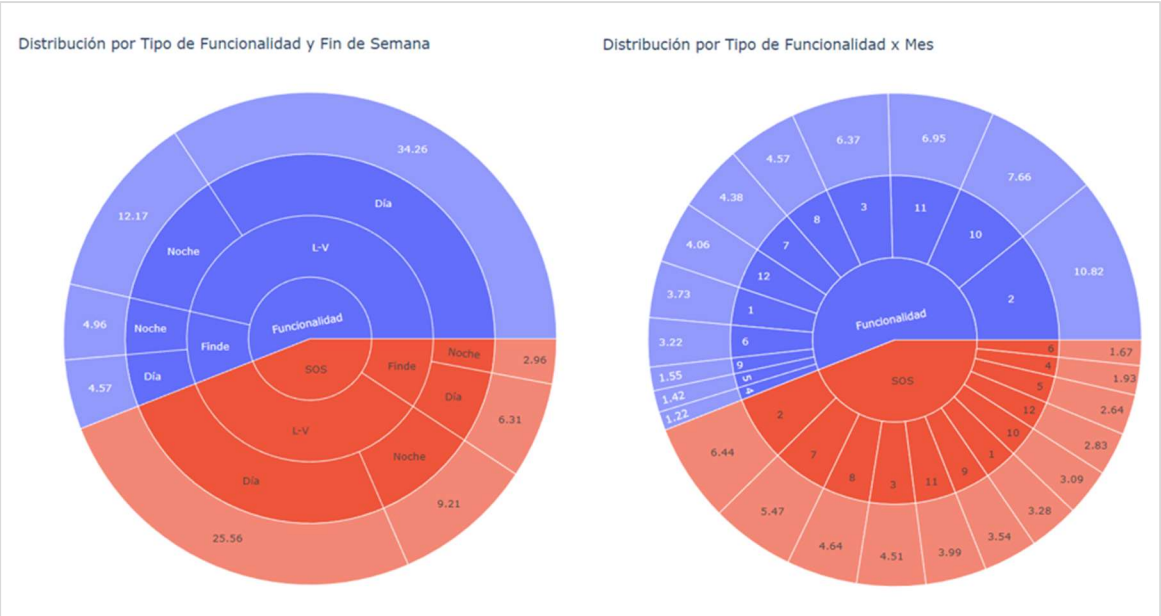


Figura 44: Izquierda: Distribución de los datos por Funcionalidad y momento del día (día o Noche). Derecha: Distribución de los datos por funcionalidad x Mes.

Si analizamos los resultados relacionados con la funcionalidad "Activar Alerta SOS", podemos decir que:

1. El 40% de los alertas se generan en ciudades o barrios del Área Metropolitana de Madrid.
2. Los días de la semana que más alertas se registran son los viernes (8.56%) y los jueves (8%). No se evidencia que la funcionalidad de alerta SOS se active más en las noches.
3. La aplicación se utiliza más durante el día, tanto de lunes a viernes (25,56%), como los fines de semana (6%) .
4. En los meses de febrero, julio y agosto se observa un mayor número de alertas SOS

5.2 Matriz de Correlación

En la matriz de correlación (ver Figura 45), se observa una correlación fuerte positiva (0.95) entre la latitud y la longitud de los puntos de inicio y fin (`lat_sp` y `long_sp`). Y entre las columnas `Year` y `user_id` (0.79) y las latitudes del punto de inicio y fin y la columna `altitude` (0.63).

Mientras que las columnas `distance` y `estimated_speed` (0,37) y `t_hours` y `distance` (0,31) presenta correlaciones positivas débiles. También observamos correlaciones débilmente negativas entre las columnas entre las columnas `longitud` de los puntos de inicio y fin con la `altitude` y las columnas `Year-Month`.

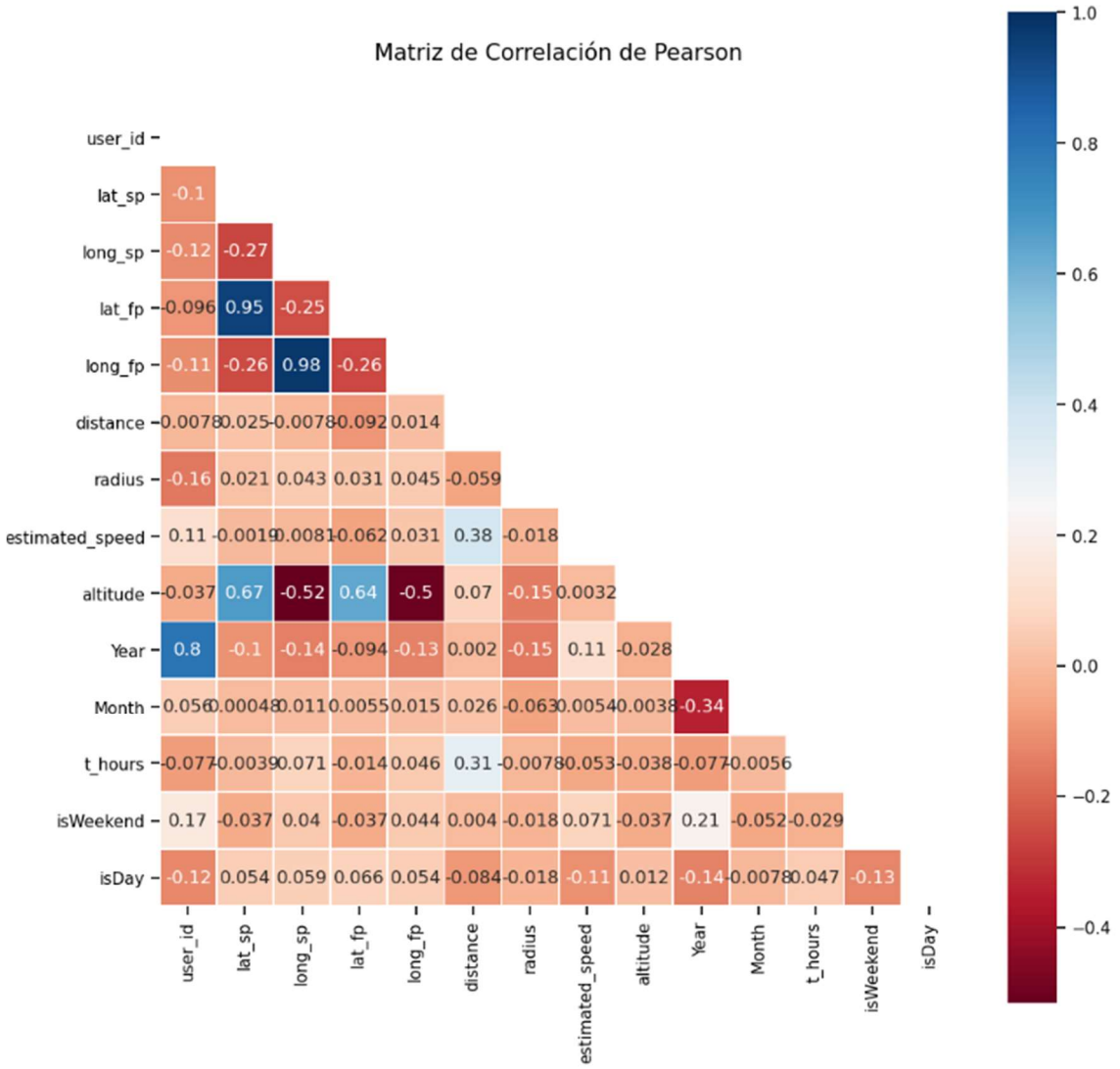


Figura 45: Matriz de Correlación del Modelo de trayectoria de los usuarios

6. Métodos de agrupamiento en Geolocalización

Como resultados de los pasos previos, hemos obtenido un conjunto de datos con el registro de las trayectorias de los usuarios de la aplicación Sister, cada trayectoria tiene un objeto `polygon` con todas las geolocalizaciones registradas cuando han activado alguna de las funcionalidades de la aplicación o cuando han activado una alerta SOS, porque han tenido una percepción de "inseguridad". Podemos decir que nuestros datos se dividen en trayectorias o geolocalizaciones de zonas "inseguras" y trayectorias o geolocalizaciones de zonas "seguras".

Con el objetivo de descubrir agrupaciones naturales o innatas en nuestro conjunto de datos no etiquetado e identificar valores atípicos en los mismos. Aplicaremos los algoritmos de clustering o agrupamiento K-Means y DBSCAN y analizaremos los resultados.

Como sabemos los algoritmos de aprendizaje automático nos ofrecen mayor precisión si los datos están estandarizados o normalizados. La estandarización de los datos es una técnica de que utilizamos para escalar características de forma todos los datos estén en un mismo intervalo, la mayoría de las veces entre 0 y 1. Pero antes, debemos aplicar una serie de transformaciones a los datos categóricos, tipo texto y booleanos para convertirlos a numéricos. Los datos categóricos son necesarios transformarlos, porque son características o etiquetas cuyos valores tienen un número finito de categorías. Para la estandarización de los datos hemos utilizado la clase `MinMaxScaler` de la API Scikit-learn, que traduce cada característica individualmente de modo que esté en el rango entre cero y uno.

Aun cuando nuestro conjunto de datos no tiene muchas características y considerando que algunas de ellas han sido calculadas a partir de otras, nos hace pensar que es innecesario incluir todas las características, entonces nos surge la duda ¿Cuáles son realmente importantes para incluir en el modelo?. Para responder a esta pregunta, utilizaremos la clase `SelectKBest` de la API Scikit-learn para extraer las mejores características de nuestro conjunto de datos. Este método selecciona las características de acuerdo con la puntuación más alta de `k`. Para aplicar los métodos de esta clase debemos dividir el conjunto de datos en `train` y `test set`, para reducir los tiempos de entrenamiento. Seleccionaremos aquellas las características estén por encima del umbral de 0.02.

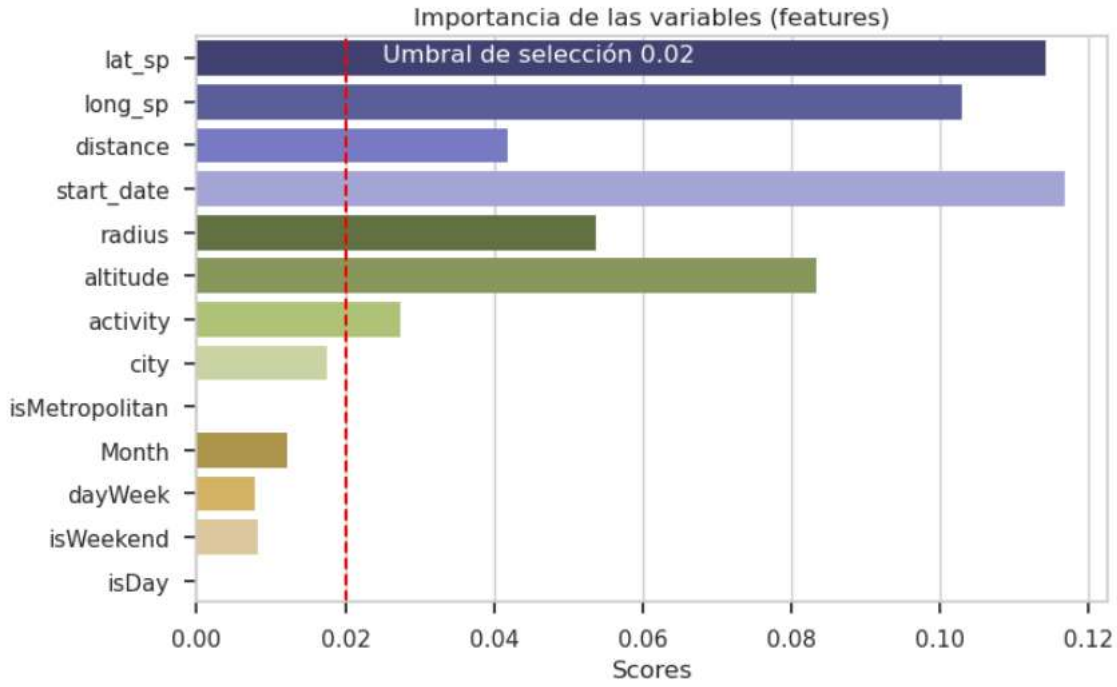


Figura 46: Gráfico de importancia de las variables utilizando la clase SelectKBest de la API Scikit-learn

La Figura 46 nos muestra que las características más importes son: `start_date`, `lat_sp`, `long_sp`, `altitude`, `distance` y `activity`. A continuación extraeremos del conjunto de datos las otras características y dividiremos nuestro conjunto de datos en dos (2) subconjuntos:

1. Zonas "Inseguras" (isAlert = True). Este conjunto de datos está conformado por un total de 660 registros.
2. Zonas "Seguras" (isAlert = False). Este conjunto de datos está conformado por un total de 827 registros.

Cada su conjunto los dividiremos a su vez en `train` y `test set`:

1. El conjunto de entrenamiento (`train set`) lo utilizaremos para entrenar o ajustar el modelo. Para este conjunto seleccionaremos el 80% de los datos.
2. El conjunto de prueba (`test set`) es necesario para la evaluación del modelo. Para este conjunto seleccionaremos el 20% de los datos

6.1 Conjunto de datos Zonas “Inseguras”

6.1.1 Selección del número óptimo del clústeres

Una vez tenemos nuestros datos preparados, comenzaremos aplicando el algoritmos de agrupación de K-Means a nuestros conjuntos de datos con los datos correspondientes a las zonas de percepción de inseguridad. Como comentamos, el algoritmo K-Means necesita que le suministremos el número del clústeres que se crearan a partir de los datos. Como no conocemos la cantidad de grupos o clústeres, necesitamos utilizar algún método que nos permita encontrar la cantidad óptima de clústeres. Utilizaremos el método de Elbow que utiliza los valores de la inercia obtenidos tras aplicar el algoritmo K-Means a diferentes números de clústeres (desde 1 a N clústeres). Además utilizaremos las métricas de Silhouette y Davis Bouldin para determinar el número de clúster. Veamos las gráficas que obtenemos de los diferentes métodos.

Figura 47: Gráficas del coeficiente de Silhoette y Davis Bouldin para obtener el número óptimo de clústeres. Para el índice de DB escogemos el valor más bajo y para el coeficiente de Silhouette escogemos el valor más cercano a 1. En ambos caso el número óptimo de clústeres es 9.

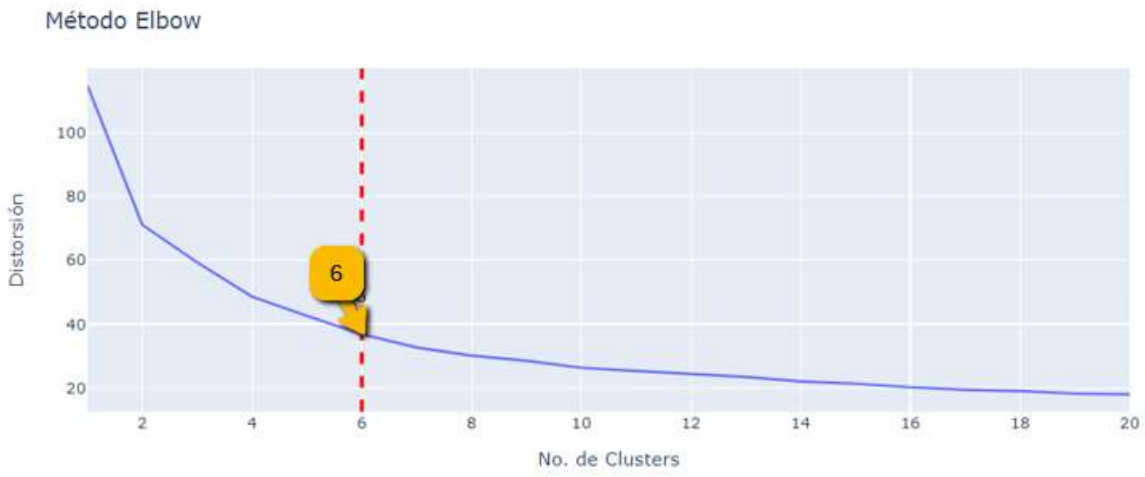


Figura 48: Método de Elbow indica que el número óptimo de clústeres es 6.

K-Means		
No. Cluster	**Davies Bouldin**	**Silhouette**
2	1,17816	0,51691
3	1,19309	0,47457
4	1,26028	0,47086
5	1,15579	0,49217
6	1,06392	0,49564
7	0,98944	0,51599
8	0,99602	0,53540
9	0,95065	0,54918
10	1,01233	0,49484
11	1,04286	0,50407
12	1,15934	0,42631

Tabla 17: Resultado de las métricas de Davis Bouldin y Silhouette para el algoritmo KMeans

En la Tabla 17, observamos los valores obtenidos para cada una de las métricas. Para el índice de Davies Bouldin, buscamos el valor más bajo (0.950646) y para el coeficiente de Silhoutte buscamos el valor más cercano a 1 (0.54918); ambos métodos coinciden en que el número óptimo de clústeres óptimo de 9. Mientras que el método Elbow, nos indica que el número óptimo de k es 6.

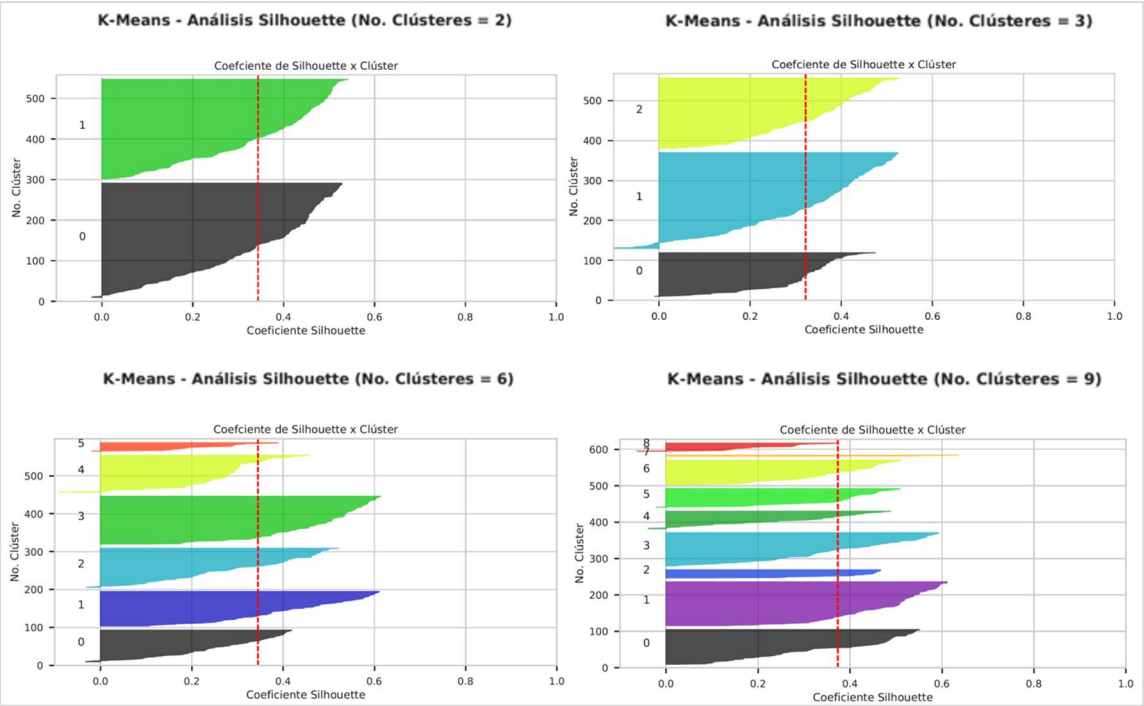


Figura 49: Algoritmos K-Means Análisis de Silhouette para 2, 3, 6 y 9 Clústeres. La línea roja representa la media del coeficiente de Silhouette

Debido a la diferencias encontradas en dos (2) de los métodos utilizados. Vamos a realizar un análisis del coeficiente de Silhouette para los diferentes número de clústeres obtenidos. Los gráficos de la Figura 49, nos muestran que los clústeres 6 y 9 pueden ser una mala elección para los datos proporcionados, debido a la presencia de clústeres con coeficientes por debajo de la media (línea vertical roja), además observamos en algunos de los casos amplias fluctuaciones en el tamaño de los clústeres (grosor del gráfico de Silhouette). El análisis de Silhouette nos indica que los mejores clústeres pueden estar entre 2 y 3.

Hemos comprobado lo complicado que puede resultar encontrar el valor óptimo de k, para el algoritmo K-Means,. Aun cuando el análisis de Silhouette nos indica que el mejor valor para k es 2, ejecutaremos el algoritmo K-Means para diferente número del clústeres, utilizando los mismos parámetros. Para minimizar los problemas de inicialización utilizaremos el algoritmo `KMeans++`, que aun cuando no elimina por completo la aleatoriedad, facilita una inicialización óptima.:

```
kmeans_kwargs = {
     "init": "k-means++",
     "n_init": 10,
     "max_iter": 300,
     "random_state": 42 }
```

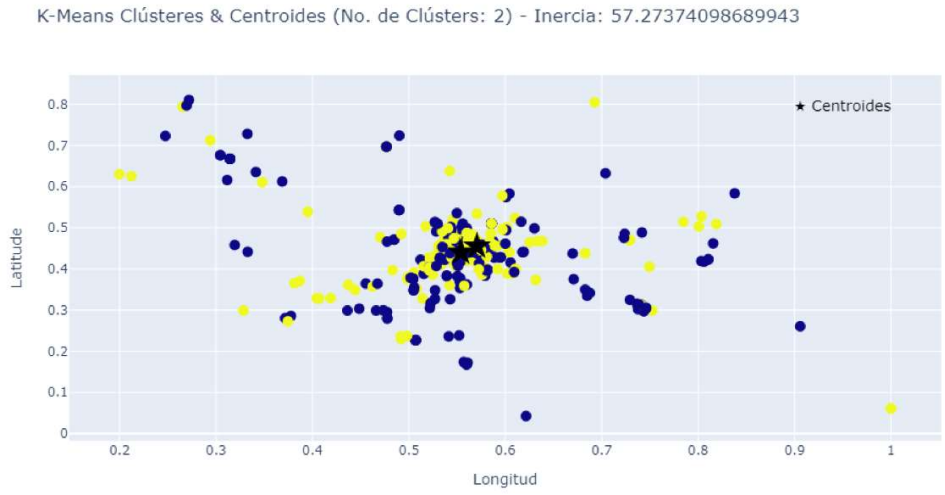
6.1.2 Modelo de Agrupación K-Means (Zonas “Inseguras”)

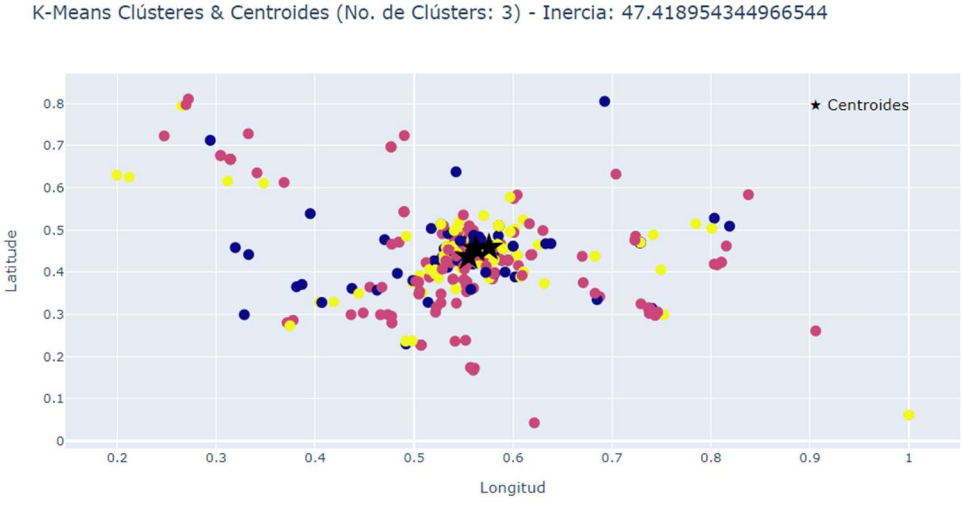
Una vez realizadas las pruebas del algoritmo K-Means para los diferentes valores de k = 2, 3, 5 y 6 en el conjunto de datos que hemos definido como de Zonas “inseguras”. Podemos revisar cómo influye el hiperparámetro k en el algoritmo K-Means, para ello repasemos los cambios de la inercia para cada valor de k.

No. Clúster	Iteraciones	Inercia
2	8	57,27374
3	7	47,41895
5	9	34,00227
6	5	29,74901

Tabla 18: Resultados de la inercia del Algoritmos K-Means para los valores de k = 2, 3, 5 y 6.

De acuerdo con los datos de la Tabla 18, el agrupamiento con mejor inercia es el k=6. Esto se debe a que al haber menos puntos por clúster, están más compactos y la inercia baja. En el caso de k=2 pasa lo contrario, clústeres más grandes generan inercias más grandes.





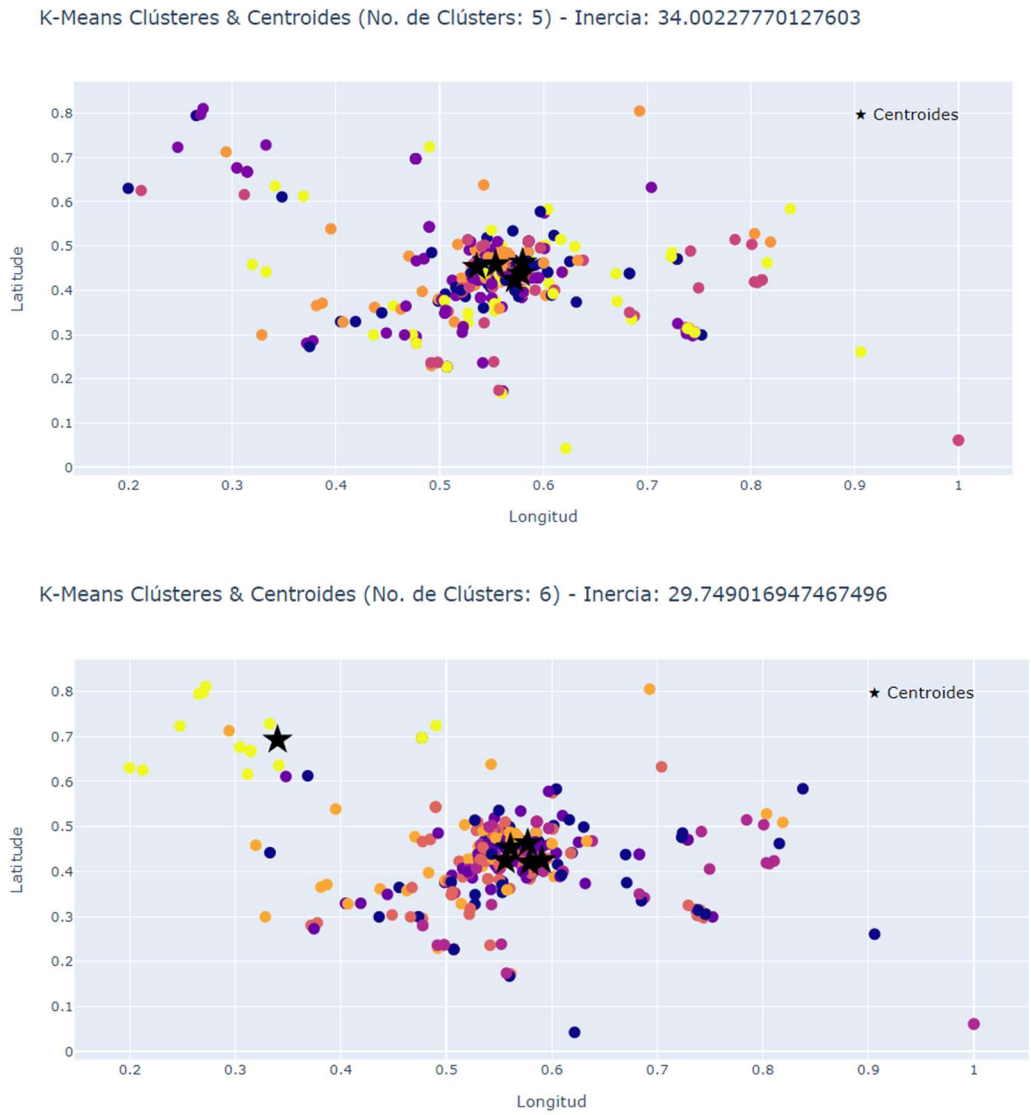


Figura 50: Distribución de los datos para los valores de k = 2, 3, 5 y 6.

Los gráficos de la Figura 50, nos muestra la distribución de los elementos en cada uno de los clústeres y los centroides. Aun cuando a simple vista, se observar que los elementos están superpuestos, utilizaremos algunas métricas de medición interna para comprobarlo.

El coeficiente de Silhouette nos permite medir cuán densos y bien separados están los grupos. Este coeficiente se encuentra dentro del rango [-1,1].

- Un coeficiente de 1 significa que los clústeres o grupos son muy densos y bien separados.
- Un coeficiente de 0 significa que los clústeres se superponen.
- Un coeficiente inferior a 0 significa que los datos pertenecientes a los grupos o clústeres son incorrectos.

El índice de Davies-Bouldin es la medida de similitud promedio de cada grupo con su grupo más similar, donde la similitud es la relación entre las distancias dentro del grupo y las distancias entre grupos. Por lo tanto, los grupos que están más separados y menos dispersos darán como resultado una mejor puntuación.

- Este índice varía entre 0 y + ∞
- Los valores más bajos indican una mejor agrupación.
- Los valores más altos indican una peor agrupación.

K-Means		
No. Clúster	**Davies Bouldin**	**Silhouette**
2	1,18476	0,34372
3	1,19455	0,32116
5	1,16327	0,32625
6	1,07045	0,34470

Tabla 19: Resultados de las métricas de valoración interna

Los valores del coeficiente de Silhouette para los diferentes valores de k, nos indican que los clústeres o grupos se superponen. Mientras que el índice de Davis Bouldin, nos indica que los grupos no están bien separados y los elementos están muy dispersos.

6.1.3 Modelo de Agrupación DBSCAN (Zonas "Inseguras")

Como hemos comentado, DBSCAN es un algoritmo de agrupación espacial basado en la densidad que supone que los clústeres son regiones densas, que están separadas por regiones con una menor densidad. Es un algoritmo capaz de detectar grupos complejos, o de forma y tamaño aleatorio. Otra característica única de este algoritmo es su resistencia a los valores atípicos.

El algoritmo DBSCAN necesita dos parámetros:

- `min_samples` o número mínimo de puntos que deben existir en un clúster, el valor mínimo recomendado para este parámetro es 3.
- `épsilon` especifica lo cerca que deben estar los puntos para ser considerados parte de un grupo o clúster.

Para calcular el valor óptimo de épsilon utilizamos el método de detección del punto de codo. Para ello, calcularemos las distancias medias entre cada punto y sus `n` vecinos más cercanos utilizando el algoritmo de NearestNeighbors de la API Scikit-learn. Luego trazamos las `n-distancias` promedio en orden ascendente en un gráfico, para determinar el valor óptimo para

épsilon identificamos en el gráfico el punto con la máxima curvatura, es decir, con la mayor pendiente. En la Figura 51, observamos que el valor óptimo de épsilon es: 0.2773.

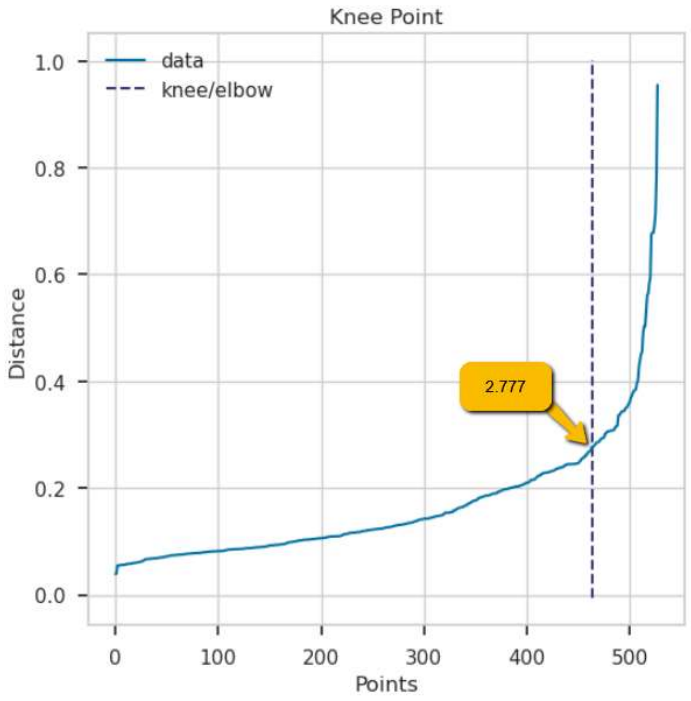


Figura 51: Gráfico Elbow para el cálculo del valor óptimo de épsilon

Una vez obtenido el valor óptimo de épsilon, ejecutamos el algoritmo DBSCAN. Este ha agrupado los datos en dos (2) grupos o clústeres, además ha detectado 19 puntos atípicos o ruido en nuestro conjunto de datos de entrenamiento, lo cual representa un 3,6% de los datos de entrenamiento. (ver Figura 52)

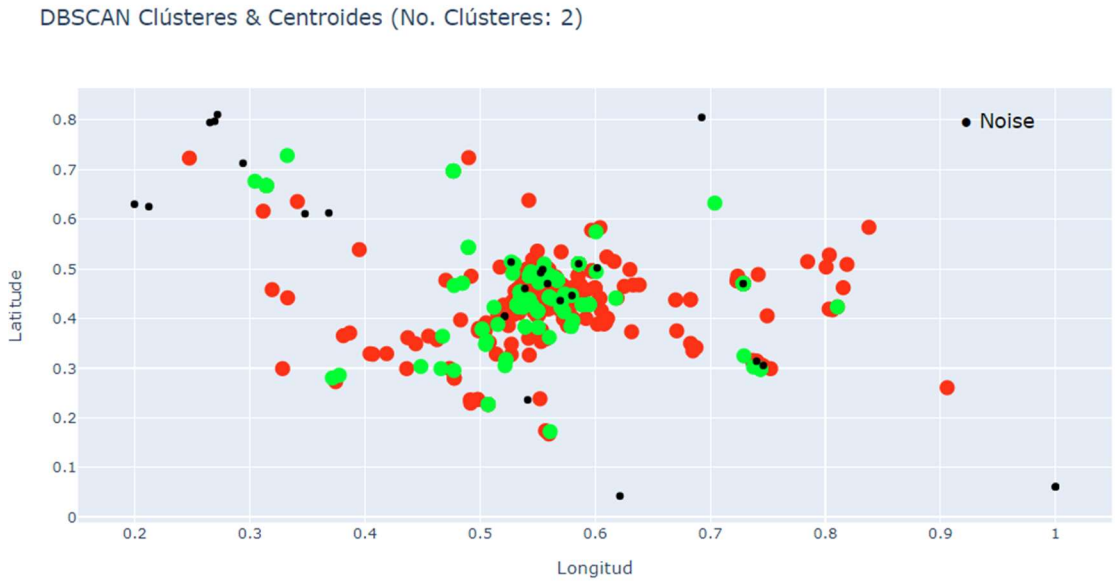


Figura 52: Distribución de los datos (train). Algoritmo DBSCAN con eps=0.277 y min_samples =4.

Las métricas de valoración nos indican que los clústeres o grupos se superponen con un coeficiente de Silhouette (0,23873) muy cercano a 0. Mientras que el índice de Davis Bouldin (3,04492), nos indica que los grupos no están bien separados y los elementos están más dispersos.

Si aplicamos el algoritmos sobre el conjunto de datos de test, los agrupa en tres (3) clústeres y detectan 11 puntos de ruido (8%).

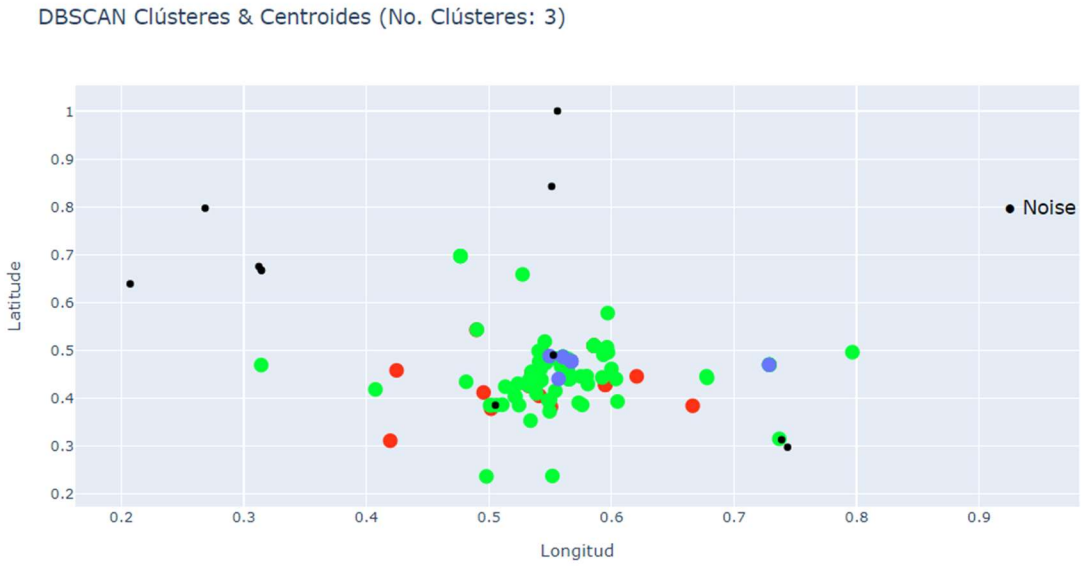


Figura 53: Distribución de los datos (test). Algoritmo DBSCAN con eps=0.277 y min_samples =4.

Si aplicamos el algoritmos sobre todo el conjunto de datos los agrupa en dos (2) clústeres y detectan 21 puntos de ruido (3%).

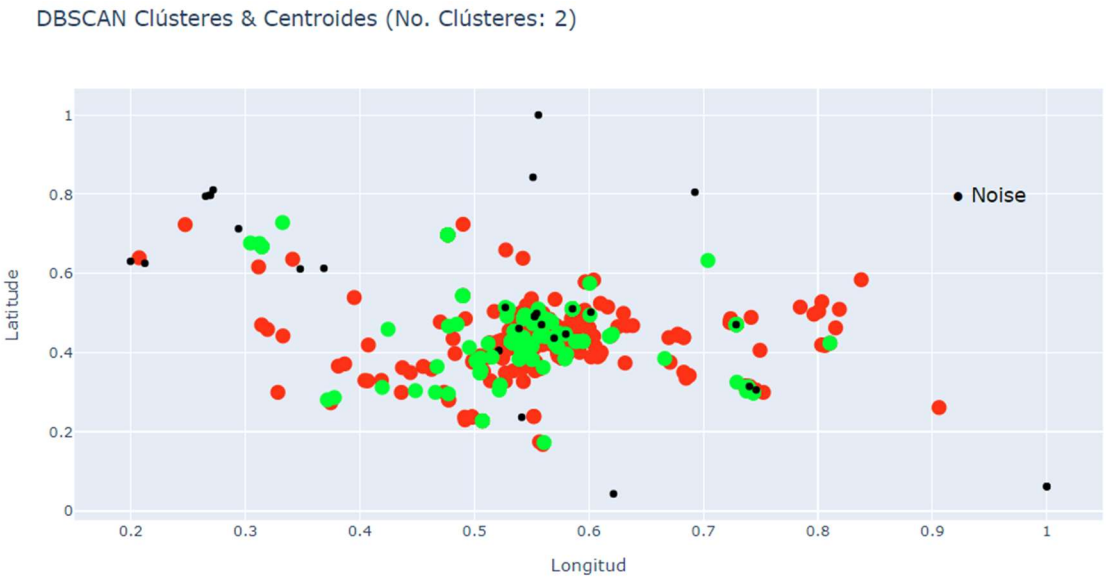


Figura 54: Distribución de todo el conjunto de datos. Algoritmo DBSCAN con eps=0.277 y min_samples =4.

6.2 Conjunto de datos Zonas “Seguras”

6.2.1 Selección del número óptimo del clústeres

Una vez tenemos nuestros datos preparados, comenzaremos aplicando el algoritmos de agrupación de K-Means a nuestros conjuntos de datos con los datos correspondientes a las zonas que definiremos como seguras. Como no conocemos la cantidad de grupos o clústeres, utilizaremos al igual que en el ejercicio anterior el método de Elbow y las métricas de Silhouette y Davis Bouldin para determinar el número de clúster. Veamos las gráficas que obtenemos de los diferentes métodos.

Figura 55: Gráficas del coeficiente de Silhoette y Davis Bouldin para obtener el número óptimo de clústeres. Para el índice de DB escogemos el valor más bajo y para el coeficiente de Silhouette escogemos el valor más cercano a 1. En ambos caso el número óptimo de clústeres es 12, en el gráfico del coeficiente de Silhouette hay un valor muy cercano al más alto que indica 3 clústeres.

Figura 56: Método de Elbow indica que el número óptimo de clústeres es 5.

K-Means		
No. Cluster	**Davies Bouldin**	**Silhouette**
2	1,35509	0,45181
3	1,07054	0,52223
4	1,06089	0,47613
5	1,20464	0,47215
6	1,08965	0,44676
7	1,12128	0,47532
8	1,09410	0,47046
9	1,06950	0,49616
10	1,03085	0,52083
11	0,99239	0,51870
12	0,93719	0,52596
13	1,04500	0,47724
14	0,99453	0,47520

Tabla 20: Resultado de las métricas de Davis Bouldin y Silhouette para el algoritmo KMeans

En la Tabla 20, observamos los valores obtenidos para cada una de las métricas. Para el índice de Davies Bouldin, buscamos el valor más bajo (0.93719) y para el coeficiente de Silhoutte buscamos el valor más cercano a 1 (0.52596); ambos métodos coinciden en que el número óptimo de clústeres óptimo de 12. Observamos otros valores del coeficiente de Silhouette muy cercanos al valor más alto, como es el caso del clúster 3. Mientras que el método Elbow, nos indica que el número óptimo de k es 5. Debido a la diferencias encontradas en dos (2) de los métodos utilizados. Vamos a realizar un análisis del coeficiente de Silhouette para los diferentes número de clústeres obtenidos.

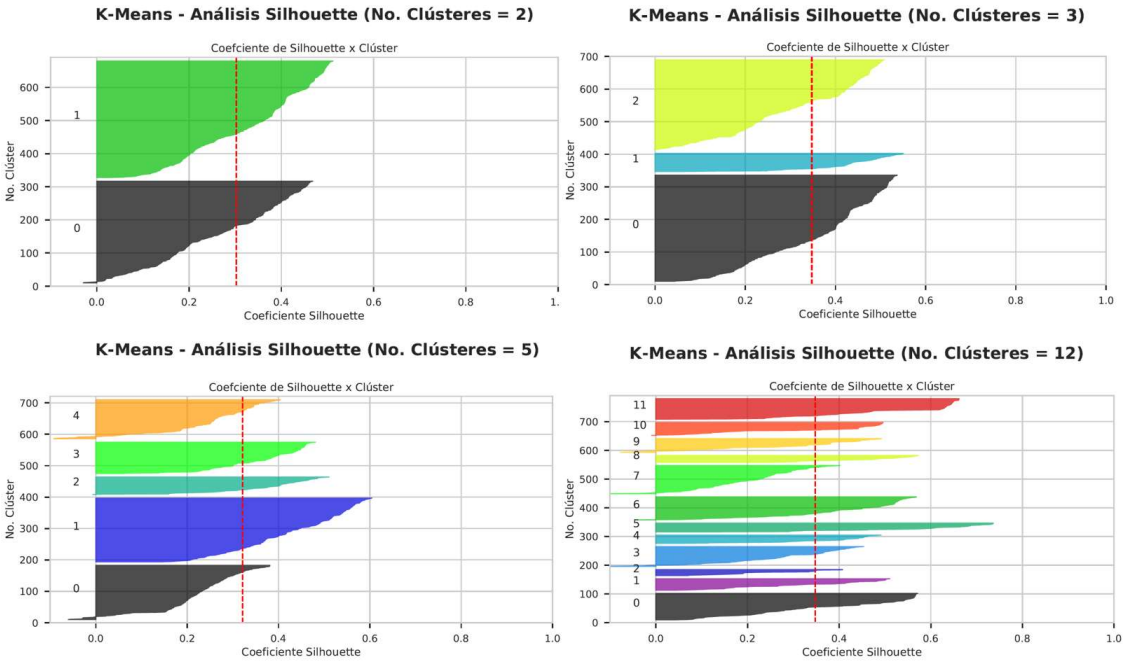


Figura 57: Algoritmos K-Means Análisis de Silhouette para 2, 3, 5 y 12 Clústeres. La línea roja representa la media del coeficiente de Silhouette

En los gráficos de la Figura 49, no se detectan clústeres con coeficientes por debajo de la media (línea vertical roja). Pero si observamos fluctuaciones en el tamaño de los clústeres (grosor del gráfico de Silhouette). El análisis de Silhouette nos indica que la mejor selección son 2 clústeres.

Aun cuando el análisis de Silhouette nos indica que el mejor valor para k es 2, ejecutaremos el algoritmo K-Means para diferente número del clústeres, utilizando los mismos parámetros que utilizamos en el ejercicio anterior.

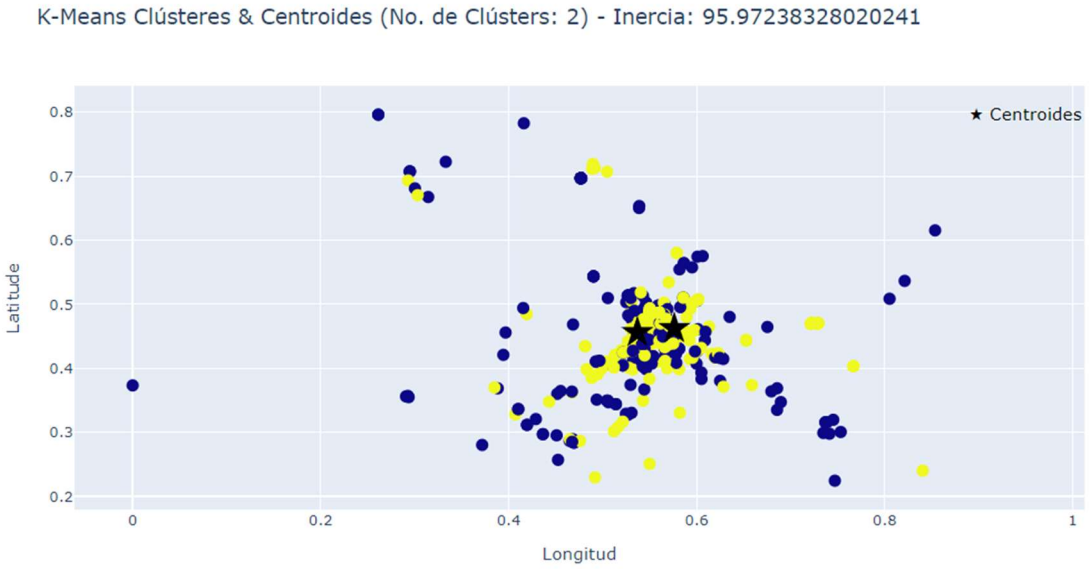
6.2.2 Modelo de Agrupación K-Means (Zonas “Seguras”)

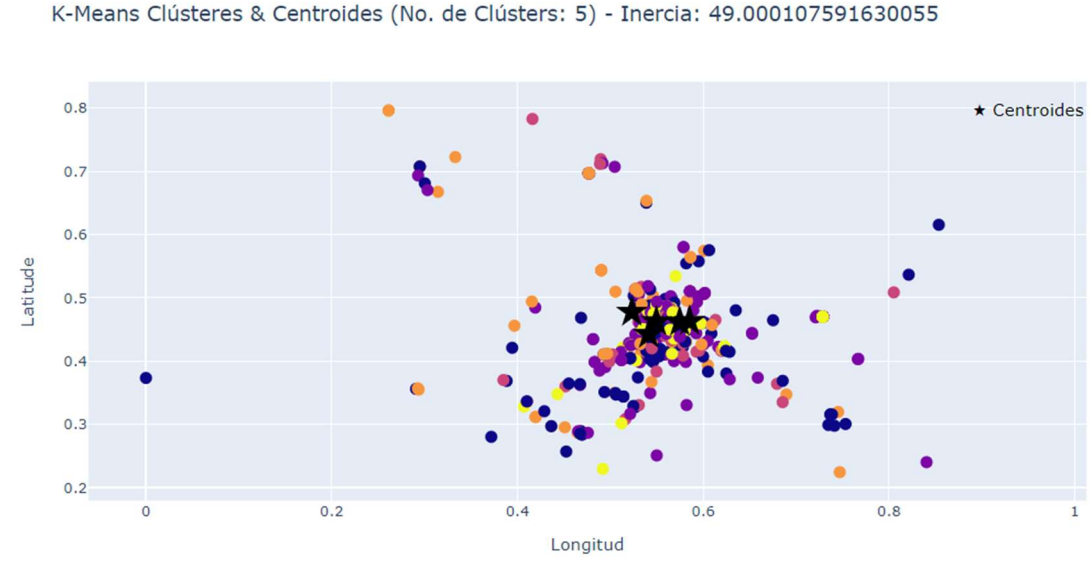
Una vez realizadas las pruebas del algoritmo K-Means para los diferentes valores de k = 2, 5 y 12 en el conjunto de datos que hemos definido como de Zonas “seguras”. Podemos revisar cómo influye el hiperparámetro k en el algoritmo K-Means, para ello repasemos los cambios de la inercia para cada valor de k.

No. Clúster	Iteraciones	Inercia
2	6	95,97238
5	6	49,00010
12	9	26,48071

Tabla 21: Resultados de la inercia del Algoritmos K-Means para los valores de k = 2, 5 y 12.

De acuerdo con los datos de la Tabla 18, el agrupamiento con mejor inercia es el k=12. Esto se debe a que al haber menos puntos por clúster, están más compactos y la inercia baja. En el caso de k=2 pasa lo contrario, clústeres más grandes generan inercias más grandes.





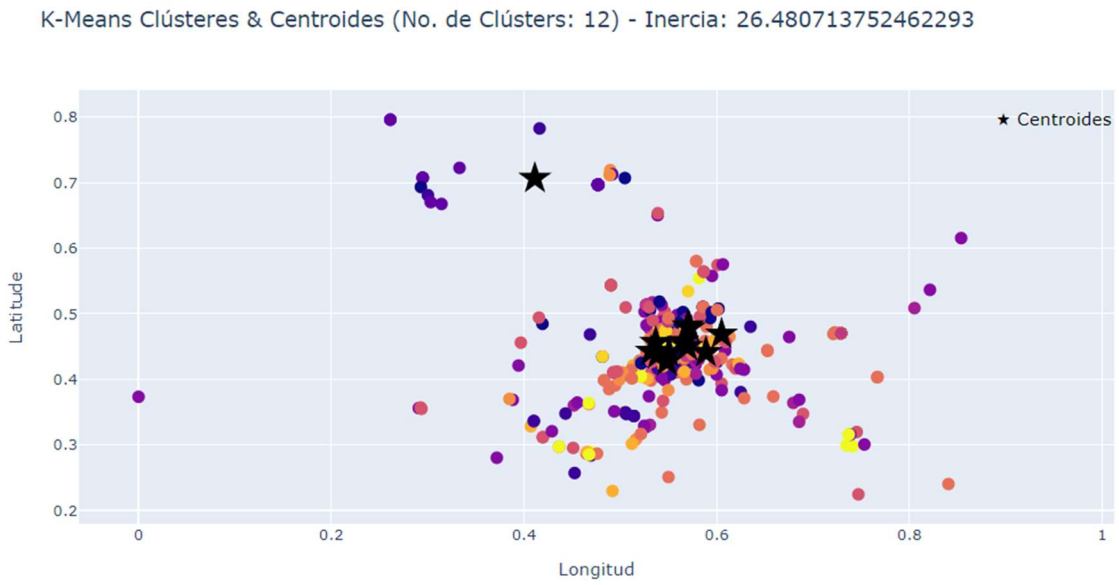


Figura 58: Distribución de los datos para los valores de k = 2, 5 y 12.

Los gráficos de la Figura 50, nos muestra la distribución de los elementos en cada uno de los clústeres y los centroides. Aun cuando a simple vista, se observar que los elementos están superpuestos, utilizaremos algunas métricas de medición interna para comprobarlo.

Los valores del coeficiente de Silhouette para los diferentes valores de k, nos indican que los clústeres o grupos se superponen. Mientras que el índice de Davis Bouldin, nos indica que los grupos no están bien separados y los elementos están muy dispersos.

K-Means		
No. Clúster	**Davies Bouldin**	**Silhouette**
2	1,35213	0,30241
5	1,20337	0,32134
12	1,00598	0,34781

Tabla 22: Resultados de las métricas de valoración interna

6.2.3 Modelo de Agrupación DBSCAN (Zonas “Seguras”)

Para ejecutar el algoritmo de DBSCAN sobre el conjunto de datos que hemos identificado como zonas “Seguras”, comenzaremos calculando el valor óptimo del parámetro épsilon. La Figura 51, nos muestra el gráfico de Elbow indicando que el punto con la máxima curvatura es: 0.2632.

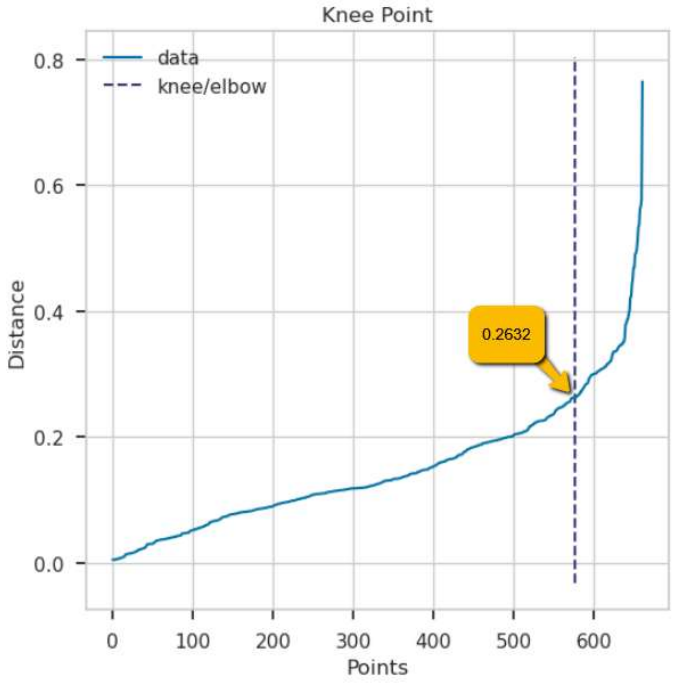


Figura 59: Gráfico Elbow para el cálculo del valor óptimo de épsilon

Una vez obtenido el valor óptimo de épsilon, ejecutamos el algoritmo DBSCAN. Este ha agrupado los datos en siete (7) grupos o clústeres, además ha detectado 16 puntos atípicos o ruido en nuestro conjunto de datos de entrenamiento, lo cual representa un 2,4% de los datos de entrenamiento. (ver Figura 52)

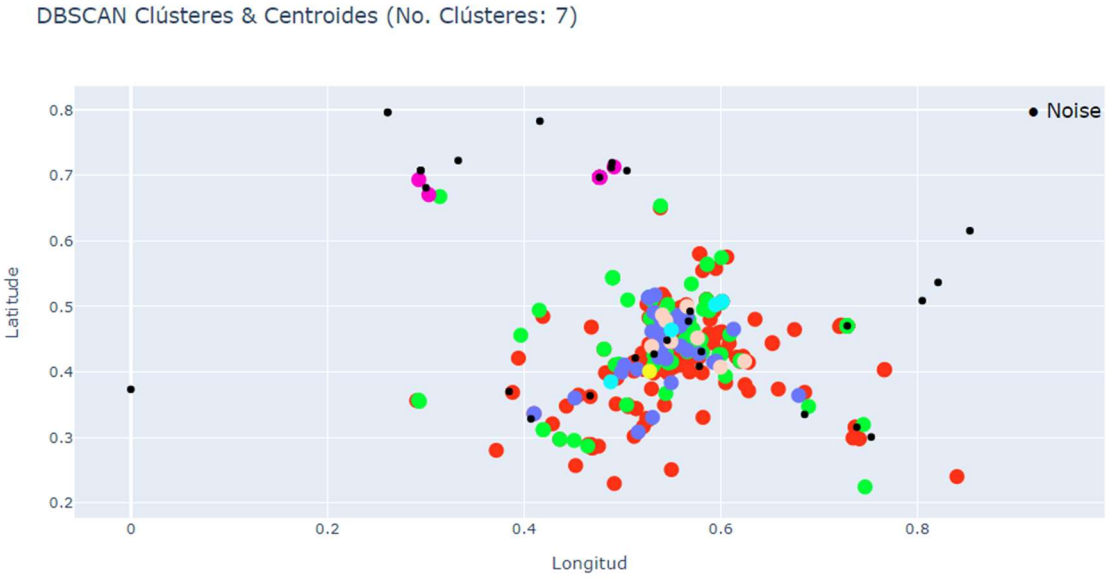


Figura 60: Distribución de los datos (train). Algoritmo DBSCAN con eps=0.26322 y min_samples =4.

Las métricas de valoración nos indican que los clústeres o grupos se superponen con un coeficiente de Silhouette (0,0959) muy cercano a 0. Mientras que el índice de Davis Bouldin (1,7827), nos indica que los grupos no están bien separados y los elementos están más dispersos.

Si aplicamos el algoritmos sobre el conjunto de datos de test, los agrupa en cuatro (4) clústeres y detectan 25 puntos de ruido (15%).

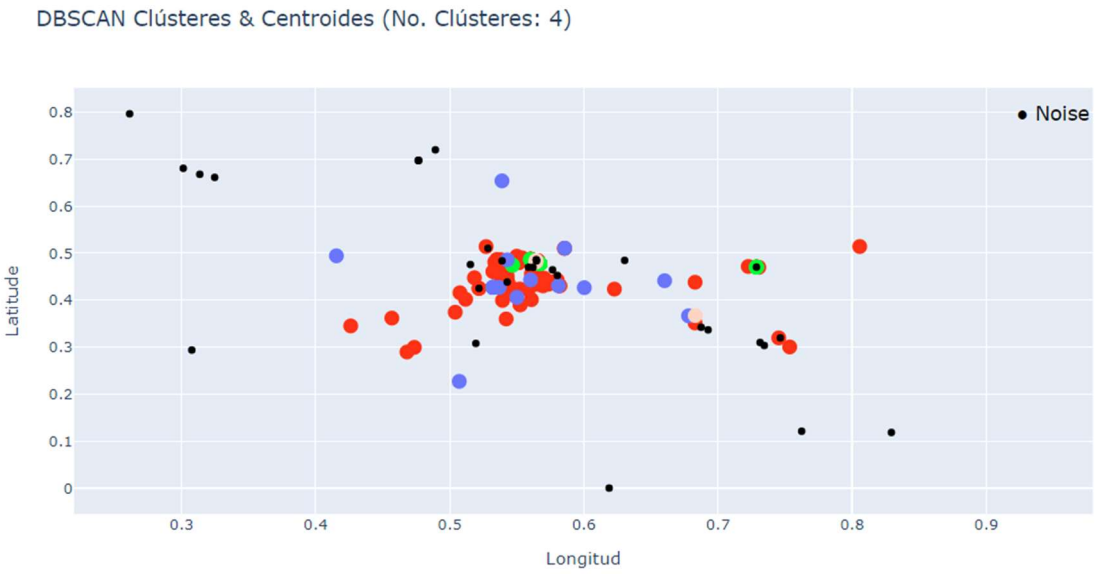


Figura 61: Distribución de los datos (test). Algoritmo DBSCAN con eps=0.2632 y min_samples =4.

Si aplicamos el algoritmos sobre todo el conjunto de datos los agrupa en siete (7) clústeres y detectan 25 puntos de ruido (3%).

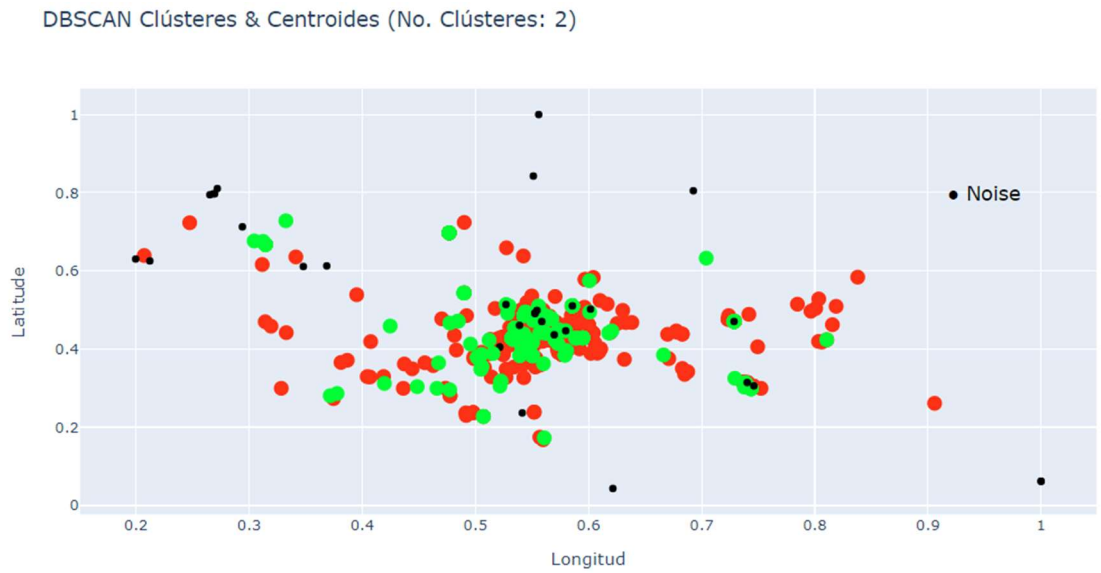


Figura 62: Distribución de todo el conjunto de datos. Algoritmo DBSCAN con eps=0.2632 y min_samples =4.

7. Conclusiones y trabajos futuros

7.1 Conclusiones

- Una de las ventajas detectadas al utilizar los servicios del proveedor en la nube es una disminución de los tiempos de ejecución. Si consideramos que la media del tiempo de ejecución del contenedor Docker con la aplicación en un entorno local es superior a las 5 horas; mientras que si ejecutamos el mismo contendor en la nube el tiempo medio de ejecución baja a 3 horas. Un disminución de aproximadamente el 40% del tiempo de ejecución.
- La limitación de los tiempos de ejecución máximos de las funciones Lambda en AWS, puede ocasionar que al aumentar el volumen de datos, aumenten los tiempos de ejecución y la función lambda no ejecute su totalidad. Al ejecutarlas en el entorno de producción debido al volumen de datos, las funciones Lambda terminaban por límites de tiempos, por lo cual fue necesario reducir el tamaño de los datos a procesar.
- Los procesos de extracción de datos pueden ser implementados en diferentes plataformas u otros proveedores servicios en la nube, que permitan ejecutar aplicaciones en contendores. También pueden ser adaptados a otros modelos de geoposicionamiento porque están implementados por módulos desarrollados en lenguaje de programación Python.
- El análisis de coeficiente de Silhouette, nos facilita la estimación el número óptimo de clústeres, porque nos ofrece información adicional sobre el tamaño de los grupos y cuales se encuentra por debajo de la media.
- Las métricas de validación internas como el coeficiente de Silhouette y el índice de Davies-Bouldin, nos indica el conjunto de datos presenta superposición y dispersión, lo cual afecta la detección de los grupos o clústeres.
- El algoritmo de DBSCAN nos permite deducir que existe aproximadamente un 3% de ruido en el conjunto de datos.
- En el algoritmo de K-Means la mayor dificultad encontrada es la determinar el número óptimo de clúster.
- El algoritmo DBSCAN es muy simple de utilizar, capaz de identificar cualquier número de grupos. Sin embargo, una variación significativa de la densidad entre los clústeres puede afectar la selección correcta de los clústeres.

7.2 Trabajos Futuros

- Evaluar la facilidad o dificulta de implementación y el rendimiento de la aplicación de extracción masiva de datos utilizando el servicio de Elastic Kubernetes Service (EKS).
- Incorporar funcionalidad de almacenamiento en cache al proceso geolocalización inversa, para la asignación de la ciudad y país, a fin de evitar los errores generados por superar el tiempo de conexión con la API.
- Incluir un logging file, que registre todas las transacciones realizadas, a fin de identificar los errores generados en el proceso de extracción de datos, la solución actual muestra los resultados en la consola.
- Aplicar algoritmos de aprendizaje profundo al conjunto de datos creado, para detectar zonas de “riesgos” o “inseguras”.
- Crear una aplicación basada en un modelo de detección de zonas de “riesgo” o “inseguras”, que permita informar a los usuarios de la aplicación.

8. Bibliografía

Aaron Ploetz, D. K. (2018). Seven NoSQL Databases in a Week. En D. K. Aaron Ploetz, *Seven NoSQL Databases in a Week* (pág. 308). Packt Publishing.

AWS Amazon Batch. (27 de 03 de 2022). *Amazon - AWS Batch*. Obtenido de https://aws.amazon.com/es/batch/

AWS Amazon Docker. (04 de enero de 2022). *aws.amazon.com*. Obtenido de aws.amazon.com: https://aws.amazon.com/es/docker/

AWS Amazon ECR. (04 de enero de 2022). *AWS - Amazon ECR*. Obtenido de https://docs.aws.amazon.com/es_es/AmazonECR/latest/userguide/what-is-ecr.html

AWS Amazon ECS. (27 de 03 de 2022). *AWS - Amazon ECS*. Obtenido de https://aws.amazon.com/es/ecs/?c=cn&sec=srv

AWS Amazon EKS. (04 de enero de 2022). *AWS Amazon EKS*. Obtenido de https://aws.amazon.com/es/eks/

AWS Amazon Fargate. (27 de 03 de 2022). *AWS Fargate*. Obtenido de https://aws.amazon.com/es/fargate/?c=cn&sec=srv

AWS Amazon IAM. (15 de 04 de 2022). *AWS Identity and Access Management*. Obtenido de https://docs.aws.amazon.com/es_es/es_es/IAM/latest/UserGuide/introduction.html

AWS Amazon Lambda. (10 de abril de 2022). *AWS Lambda*. Obtenido de https://docs.aws.amazon.com/es_es/es_es/lambda/latest/dg/welcome.html

AWS Amazon Pods. (04 de enero de 2022). *AWS Amazon Pods*. Obtenido de https://aws.amazon.com/es/blogs/aws-spanish/uso-de-multi-container-pods-para-logs-de-aplicaciones-stateful-en-amazon-eks/

AWS Amazon S3. (27 de 03 de 2022). *AWS Amazon S3*. Obtenido de https://aws.amazon.com/es/s3/?nc2=h_ql_prod_fs_s3

AWS Amazon SageMaker. (15 de 04 de 2022). *Documentación de Amazon SageMaker*. Obtenido de https://docs.aws.amazon.com/es_es/sagemaker/?id=docs_gateway

AWS Amazon Web Services . (23 de marzo de 2022). *Amazon Web Services (AWS)*. Obtenido de https://aws.amazon.com/es/what-is-aws/?nc1=f_cc

Bisht, J. (12 de Febrero de 2018). *Geeksforgeeks*. Obtenido de https://www.geeksforgeeks.org/eigenvector-centrality-centrality-measure/#:~:text=In%20graph%20theory%2C%20eigenvector%20centrality%20%28also%20called%20eigencentrality%29,in%20question%20than%20equal%20connections%20to%20low-scoring%20nodes.

Cherven, K. (2015). Mastering Gephi Network Visualization. En K. Cherven, *Mastering Gephi Network Visualization.* Packt Publishing.

Durán, J. (12 de 10 de 2021). *Somos Binarios - Tutorial Docker*. Obtenido de https://www.somosbinarios.es/docker-conceptos-tutorial/

Firebase - Cloud Firestore. (11 de 04 de 2022). *Firebase Google*. Obtenido de https://firebase.google.com/docs/firestore?hl=es-419

Folium - Documentation. (14 de 04 de 2022). *Folium 0.12.1 documentation*. Obtenido de https://python-visualization.github.io/folium/

Geopy. (15 de 04 de 2022). *Welcome to GeoPy's documentation!* Obtenido de https://geopy.readthedocs.io/en/stable/#

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition.* United States of America: O'Reilly Media, Inc.

Google Developers. (13 de 01 de 2021). *Google Developers.* Obtenido de https://developers.google.com/machine-learning/clustering/algorithm/advantages-disadvantages

Guzmán, E. L. (Julio de 2019). *Métricas para la validación de CLustering.* Obtenido de Universidad Nacional de Colombia: https://disi.unal.edu.co/~eleonguz/cursos/mda/presentaciones/validacion_Clustering.pdf

Hippocrates Guild. (19 de 08 de 2020). *Hippocrates Guild*. Obtenido de https://hippocratesguild.com/es/qu%C3%A9-tan-r%C3%A1pido-es-la-velocidad-promedio-de-caminar-ritmo-de-caminar-r%C3%A1pido/

Instituto Nacional de Estadística - EVDVG. (2020). *Estadística de Violencia Doméstica y Violencia de Género (EVDVG).* INE.

Instituto Nacional de Estadistica (INE). (13 de 04 de 2022). *Instituto Nacional de Estadistica (INE)*. Obtenido de https://www.ine.es/ss/Satellite?L=es_ES&c=INESeccion_C&cid=1259926144037&p=1254735110672&pagename=ProductosYServicios%2FPYSLayout&param1=PYSDetalle&param3=1259924822888

Kaggle - World Cities. (30 de 01 de 2022). *Kaggle*. Obtenido de https://www.kaggle.com/realankit/world-cities

Machine Learning Knowledge. (10 de 12 de 2021). *Tutorial for DBSCAN Clustering in Python Sklearn.* Obtenido de https://machinelearningknowledge.ai/tutorial-for-dbscan-clustering-in-python-sklearn/

Mujeres de la Ciudad de Madrid. (11 de 02 de 2022). Estudio sobre la seguridad femenina en la calle. (Sister, Entrevistador)

Newman, M. E. (2006). *Modularity and community structure in networks.*

Niño, J. O., & Berzal, F. (Mayo 2019). *Evaluation Metrics for Unsupervised Learning https://arxiv.org/pdf/1905.05667.pdf.* Granada: Universidad de Granada.

Nominatim. (14 de 04 de 2022). *Open-source geocoding*. Obtenido de https://nominatim.org/

Pandata Web. (10 de 06 de 2022). *Pandata Web.* Obtenido de https://pandataweb.com/algoritmo-kmeans/#:~:text=Con%20t%C3%A9rminos%20estad%C3%ADsticos%2C%20esto%20quiere%20decir%20que%20KMeans,y%20haremos%20experimentos%20con%20ella%20durante%20el%20ejemplo.

R. Zafarani, M. A. (2014). *Social Media Mining: An Introduction.* USA: Cambridge University Press,.

Revista DGT. (31 de 05 de 2022). *Revista DGT*. Obtenido de https://revista.dgt.es/es/sabia-que/normas/2018/0103velocidad-a-la-que-debe-circular.shtml

Shannon Bradshaw, E. B. (2019). *MongoDB: The Definitive Guide, 3rd Edition.* O'Reilly Media.

Shaw, A. (31 de Diciembre de 2019). *Strategic Planet*. Obtenido de https://www.strategic-planet.com/2019/12/how-a-page-rank-is-calculated-in-gephi/

Singh, H. (2020). *Practical Machine Learning with AWS : Process, Build, Deploy, and Productionize Your Models Using AWS.* Apress.

Stackoverflow, @. (28 de febrero de 2021). *Stackoverflow*. Obtenido de https://stackoverflow.com/questions/66291208/firebase-admin-with-aws-lambda-python

Topographic Map. (01 de 05 de 2022). *Topographic Map*. Obtenido de https://es-es.topographic-map.com/maps/6o29/Madrid/

Unknown. (2011). *Social Network Analysis Theory and Applications.* Creative Commons.

Wave Location Technologies. (11 de abril de 2022). *Join Sister*. Obtenido de https://joinsister.com/es/

Wikimedia - Open Container Initiative. (12 de octubre de 2021). *Wikipedia - Open Container Initiative*. Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Open_Container_Initiative

Wikipedia - Área Metropolitana de Madrid. (29 de 05 de 2022). *Área Metropolitana de Madrid*. Obtenido de https://es.wikipedia.org/wiki/%C3%81rea_metropolitana_de_Madrid

Wikipedia - DBSCAN. (9 de 5 de 2022). *Wikipedia - DBSCAN.* Obtenido de https://es.wikipedia.org/wiki/DBSCAN

Wikipedia - Docker. (14 de 04 de 2022). *Docker (software)*. Obtenido de https://es.wikipedia.org/wiki/Docker_(software)#:~:text=Docker%20es%20un%20proyecto%20de%20c%C3%B3digo%20abierto%20que,de%20virtualizaci%C3%B3n%20de%20aplicaciones%20en%20m%C3%BAltiples%20sistemas%20operativos.

Wikipedia - Geolocalización. (14 de 04 de 2022). *Geolocalización*. Obtenido de https://es.wikipedia.org/wiki/Geolocalizaci%C3%B3n#:~:text=La%20geolocalizaci%C3%B3n%20es%20la%20capacidad%20para%20obtener%20la,bien%20para%20la%20consulta%20real%20de%20la%20ubicaci%C3%B3n.

Wikipedia - Modularity (networks). (25 de 05 de 2022). *Modularity (networks)*. Obtenido de https://en.wikipedia.org/wiki/Modularity_(networks)

Wikipedia - Municipios Comunidad de Madrid. (29 de 05 de 2022). *Wikipedia Municipios Comunidad de Madrid*. Obtenido de Anexo:Municipios de la Comunidad de Madrid - Wikipedia, la enciclopedia libre

9. Anexos

9.1 Programas AWS ECS y ECR

9.1.1 Programas para la extracción masiva de datos (Docker)

Dockerfile

```
FROM amazonlinux:latest
RUN yum -y install wich unzip python3 pip3
RUN pip3 install boto3
RUN pip3 install firebase_admin
RUN pip3 install geopandas
RUN pip3 install geopy
RUN pip3 install google-cloud-core
RUN pip3 install google-cloud-firestore
RUN pip3 install grpcio
RUN pip3 install setuptools
RUN pip3 install wheel
RUN pip3 install pymongo

WORKDIR /app_batch_docker
COPY ./aws_process ./aws_process
COPY ./batch_process ./batch_process
COPY ./ddbb_process ./ddbb_process
COPY ./data/sister-prod_firebase.json ./data/sister-prod_firebase.json
COPY constants.py .
COPY main.py .
CMD ["python3", "main.py"]
```

Programa principal (main.py)

```
# Import libraries
from time import time

import pathlib

import csv

# Import modules
from batch_process import search_location
from ddbb_process import ddbb_mongodb
from aws_process import upload_s3
from ddbb_process import ddbb_sister

# Import constants
import constants as constant

BASE_DIR = pathlib.Path(__file__).parent.resolve()

if __name__ == '__main__':
    print("Executing batch process version: 14/03/2022")
    db = ddbb_sister.ddbb_conexion(constant.PATH_CRED)
    collection_ref = ddbb_sister.ddbb_collection(db, 'monitorings')
    if not constant.SAVE_DDBB:
        f = open(constant.PATH_FILE, 'w', newline='')
        csv_writer = csv.writer(f, delimiter=',')
        csv_writer.writerow(constant.ENCABEZADO)  # escribe el encabezado
    max_monitorings = constant.MAX_MONITORING
    start = time()
    n_reg_process = 0
    for document_ref in collection_ref.list_documents():
        # Get monitoring id
        monitoring_id = document_ref.id
        n_reg_process += 1
        # Get user id
```

```
            user_id, isAlert, activate = ddbb_sister.ddbb_obtein_user_id(document_ref)
            # Get locations and devices status from public - collection - locations
            public_ref = document_ref.collection("public")
            collections_ref = public_ref.document("collections")
            locations_ref = collections_ref.collection("locations")
            first_point = True
            target_city_ok = False
            geo_points = 0
            # Loop for locations (points)
            for location_ref in locations_ref.list_documents():
                # TODO Crear una función extraer location
                location_ref = location_ref.get().to_dict()
                if 'location' in location_ref:
                    data = location_ref['location']
                    latitude = data['latitude']
                    longitude = data['longitude']
                    timestamp = data.get('endDate', None)
                    UpdateAt = data['locationUpdateAt']
                    radius = data['radius']
                    speed = data['speed']
                    altitude = data['altitude']
                    activity = location ref['activity'].upper()
                    # Check only if first point of locations in city (review)
                    if first_point:
                        target_city_ok = search_location.isLocatedAt(latitude,
                        longitude, constant.TARGET_ZONE, 1000)
                        first_point = False
                    if target_city_ok:
                        geo_points = geo_points + 1

                    else:

                        break
                    # Limit points
                    if geo_points > max_monitorings:  # 100
                        break
                    if constant.SAVE_DDBB:
                        ddbb_mongodb.insert_locations(monitoring_id, user_id,
                        isAlert, activate, latitude, longitude, UpdateAt, radius,
                        speed, altitude, activity)
                    else:
                        row = [monitoring_id, user_id, isAlert, activate, latitude,
                               longitude, UpdateAt, radius, speed, altitude, activity,
                               constant.TARGET_ZONE]
                        csv writer.writerow(row)
                print("Monitoring '" + monitoring_id + "' focused in " +
                      constant.TARGET_ZONE + " with " + str(geo_points) + " locations")
        # Update last date
        if constant.SAVE_DDBB:
            ddbb_mongodb.update_lastbatchdate()  # update last date
        else:
            upload_s3.upload_file_s3(f"{BASE_DIR}{constant.PATH_FILE[1:]}",
                                     constant.BUCKET_NAME, constant.PATH_FILE[2:])
        end = time()
        print(f'{n_reg_process} records processed. Execution time {end - start}
                 seconds!')

```

9.1.2 Módulos Principales

Modulo batch_process - search_location

```
from random import randint
import logging

from geopy.geocoders import Nominatim
from geopy.exc import GeocoderTimedOut, GeocoderServiceError


def reverse_geocode(geolocator, latlon, sleep_sec):
    """
    param geolocator:
    param latlon:
    param sleep_sec:
```

```
        return an object geolocator.reverse(latlon) or None
        """
        try:
            return geolocator.reverse(latlon)
        except GeocoderTimedOut:
            logging.info('TIMED OUT: GeocoderTimedOut: Retrying...')
            sleep(randint(1 * 100, sleep_sec * 100) / 100)
            return reverse_geocode(geolocator, latlon, sleep_sec)
        except GeocoderServiceError as e:
            logging.info('CONNECTION REFUSED: GeocoderServiceError encountered.')
            logging.error(e)
            return None
        except Exception as e:
            logging.info('ERROR: Terminating due to exception {}'.format(e))
            return None


    def isLocatedAt(lat, long, city, resolution):
        """
        :param lat:
        :param long:
        :param city:
        :param resolution:
        :return: Returns True if the city corresponds to the coordinates
                 (latitude and longitude) supplied or False, otherwise returns False
        """
        user_agent = 'user_me_{}'.format(randint(10000, 99999))
        print("Buscando location", user_agent)
        locator = Nominatim(user_agent='http', timeout=300)
        coordinates = str(lat) + "," + str(long)

        location = reverse_geocode(locator, coordinates, 1)

        try:
            geo_located_address = location.raw['address']
            # geo_located_address = {'amenity': 'Colegio Arturo Soria',
              'road': 'Calle Duque de Tamames', 'quarter': 'San Juan Bautista',
            # 'suburb': 'Ciudad Lineal', 'city': 'Madrid',
              'county': 'Área metropolitana de Madrid y Corredor del Henares',
            # 'state': 'Comunidad de Madrid', 'postcode': '28001',
              'country': 'España', 'country_code': 'es'}
            #
            if 'town' in geo_located_address:
                geo_located_city = geo_located_address['town']
            elif 'city' in geo located address:
                geo_located_city = geo_located_address['city']
            elif 'county' in geo_located_address:
                geo_located_city = geo_located_address['county']
            else:
                geo_located_city = "None_CITY"

            return city.lower() == geo_located_city.lower()
        except Exception as e:
            logging.info('ERROR: Terminating due to exception {}'.format(e))
            geo_located_city = "None_CITY"
            return city.lower() == geo_located_city.lower()

    def createLocationPoint(monitoring_id, user_id, isAlert, data):
        """
        :param monitoring_id:
        :param user_id:
        :param isAlert:
        :param data:
        :return: Returns a dictionary with the information the activity,
                 location and status of the mobile device
        """
        latitude = data['latitude']
        longitude = data['longitude']
        timestamp = data['locationUpdateAt']
        radius = data['radius']
        speed = data['speed']
        altitude = data['altitude']
        location_data = {'timestamp': timestamp,
                         'radius': radius,
                         'speed': speed,
                         'altitude': altitude,
                         'isSafe': isAlert,
```

```
                        'monitoring': monitoring_id,
                        'userId': user_id,
                        'latitude': latitude,
                        'longitude': longitude
                        }
        return location_data

    def GetLocated(lat, long, city, resolution):
        """
        :param lat:
        :param long:
        :param city:
        :param resolution:
        :return: Returns True if the city corresponds to the coordinates
                 (latitude and longitude) supplied or False, otherwise returns False
        """
        user_agent = 'user_me_{}'.format(randint(10000, 99999))

        locator = Nominatim(user_agent='http', timeout=300)
        coordinates = str(lat) + "," + str(long)
        location = reverse_geocode(locator, coordinates, 1)

        try:
            geo_located_address = location.raw['address']
            print(geo_located_address)
            if 'town' in geo_located_address:
                geo_located_city = geo_located_address['town']
            elif 'city' in geo_located_address:
                geo_located_city = geo_located_address['city']
            elif 'state' in geo_located_address:
                geo_located_city = geo_located_address['state']
            elif 'state_district' in geo_located_address:
                geo located city = geo located address['state district']
            elif 'county' in geo_located_address:
                geo_located_city = geo_located_address['county']
            elif 'country' in geo_located_address:
                geo_located_city = geo_located_address['country']
            else:
                geo located city = "None CITY"
            geo_located_country = geo_located_address['country']
            return  geo_located_city, geo_located_country
        except Exception as e:
            logging.info('ERROR: Terminating due to exception {}'.format(e))
            geo_located_city = "None_CITY"
            return geo_located_city, geo_located_country
```

Módulo ddbb_process - ddbb_mongodb

```
# Import libraries
from bson.objectid import ObjectId
import pymongo
from datetime import datetime

# Import constants file
import constants as constant


def ddbb conexion(servidor):
    client = pymongo.MongoClient(f'mongodb://{servidor}:27017/')
    db = client.sisterlocations
    return db


def insert_locations(monitoring_id, user_id, isalert, activate, latitude,
                     longitude, updateat, radius, speed, altitude, activity,
                     city=constant.TARGET_ZONE):

    client = pymongo.MongoClient(f'mongodb://{constant.MONGODB_SERVER}:27017/')

    db = client.sisterlocations
    location = {
        "monitoring_id": monitoring_id,
        "user_id": user_id,
        "isAlert": isalert,
        "activate": activate,
        "latitude": latitude,
```

```
            "longitude": longitude,
            "UpdateAt": updateat,
            "radius": radius,
            "speed": speed,
            "altitude": altitude,
            "activity": activity,
            "city": city
        }
        location_id = db.locations.insert_one(location)
        client.close()
        return location_id.inserted_id


    def get_lastbatchdate():
        db = ddbb_conexion(constant.MONGODB_SERVER)
        cursor = db.lastbatchdate.aggregate([
            {"$project": {"_id": 1, "Lastupdate": 1}}
        ])

        for data in cursor:
            id_ld = data["_id"]
            lastdate = data["Lastupdate"]

        return id_ld, lastdate


    def update_lastbatchdate():
        db = ddbb_conexion(constant.MONGODB_SERVER)
        id_ld, last_dbdate = get_lastbatchdate()
        lastdate = datetime.timestamp(datetime.now())
        try:
            db.lastbatchdate.update_one({"_id": ObjectId(id_ld)},
                                        {"$set": {"Lastupdate": lastdate}})
        except:
            print("Error updating lastdate...")
```

Módulo ddbb_process - ddbb_sister

```
    # Import libraries
    import firebase admin
    from firebase_admin import credentials
    from firebase_admin import firestore

    def ddbb_conexion(file):
        cred = credentials.Certificate(file)
        firebase admin.initialize app(cred)
        db = firestore.client()
        print(db)
        return db

    def ddbb_collection(db, collection_name):
        collection = db.collection(collection_name)
        return collection

    def ddbb_extract_update_locations(monitoring_doc, last_date):
        locations_ref = monitoring_doc.collection("public").document("collections")
            .collection("locations") \
            .where("location.locationUpdateAt", ">", last_date) \
            .order_by("location.locationUpdateAt",
                      direction=firestore.Query.ASCENDING) \
            .stream()
        return locations_ref

    def ddbb_obtein_user_id(document_ref):
        try:
            user_owner = document_ref.collection("private").document("data")
                         .get().to_dict()
            user_data = document_ref.collection("public").document("data")
                         .get().to dict()
            if (user_owner is not None) and ('ownerUID' in user_owner):
                user_id = user_owner['ownerUID']
            else:
                user_id = "Unknown"
            if user_data is not None:
                # isAlert: True
                isAlert = user_data.get('isAlert')
```

```
36              activate = user_data.get('active')
37          else:
38              isAlert = False
39              activate = False
40          return user_id, isAlert, activate
41      except:
42          return "Unknown", False, False
```

Módulo aws_process – upload_s3

```
1   import pathlib
2   import boto3
3
4   from botocore.exceptions import ClientError
5
6   import constants as constant
7
8   BASE_DIR = pathlib.Path(__file__).parent.resolve()
9
10  AWS_REGION = constant.AWS_REGION
11  S3_BUCKET_NAME = constant.BUCKET_NAME
12  s3_client = boto3.client("s3", region_name=AWS_REGION)
13
14
15  def upload_file_s3(file_name, bucket, object_name=None):
16      print(file_name, bucket)
17      s3 = boto3.resource('s3',
18                          aws_access_key_id="accesskey",
19                          aws_secret_access_key="secretkey")
20      if object_name is None:
21          object_name = file_name
22      try:
23          response = s3.Object(bucket, file_name).upload_file(object_name)
24          print(f"'{file_name}' has been uploaded to '{S3_BUCKET_NAME}'")
25          return True
26      except ClientError as e:
27          print(f"Error: The file not has been uploaded ({e})")
28          return False
```

9.2 Anexos Funciones AWS Lambda

9.2.1 Función AWS Lambda GetMonitoringIDfromDate

```
# Import libraries
    import os
    import json
    import csv
    import pathlib

    from time import time

    # Import firebase libraries (layer)
    import firebase_admin
    from firebase_admin import credentials
    from firebase_admin import firestore

    # Import modules and constants
    import constants as constant
    from ddbb_process import ddbb_sister
    from aws_process import upload_s3 as up_s3

    S3_BUCKET_NAME = constant.bucketname

    BASE_DIR = pathlib.Path(__file__).parent.resolve()

    # Get last update date
    locationLastUpdate = constant.LAST_DATE


    def lambda_handler(event, context):
        # Connect to sister database
        cred = credentials.Certificate(constant.PATH_CRED)
        firebase_admin.initialize_app(cred)
        db = firestore.client()

        # Connect Monitoring collection
        collection = db.collection("monitorings")

        contador = 0
        start = time()
        f_contador = 0

        filename = "/tmp/monitoring_" + str(f_contador) + ".csv"

        # Create File monitoring<contador>.csv
        f = open(filename, 'w', newline='')

        csv_writer = csv.writer(f, delimiter=',')
        csv_writer.writerow(constant.M_HEADER)

        for document_ref in collection.list_documents():

            # Get monitoring id
            monitoring_id = document_ref.id

            # Get user data
            user_id, isAlert, activate = ddbb_sister.ddbb_obtein_user_id(document_ref)

            # Insert user data
            row = [monitoring_id, user_id, isAlert, activate]
            csv_writer.writerow(row)
            contador = contador + 1

            # Save Data in a new file monitoring<contador>.csv
            if contador == 500:
                # Close file monitoring<contador>.csv
                f.close()

                # Upload file to S3
                up_s3.upload_file_s3(filename, constant.bucketname)

```

```
            # Delete file monitoring>contador>.csv
            os.remove(filename)

            # Create a new file monitoring<contador>.csv
            contador = 0
            f_contador = f_contador + 1

            filename = "/tmp/monitoring_" + str(f_contador) + ".csv"
            f = open(filename, 'w', newline='')
            csv_writer = csv.writer(f, delimiter=',')
            csv_writer.writerow(constant.M_HEADER)

    db.close()

    end = time()

    return {
        'statusCode': 200,
        'body': json.dumps(
            f"Execution time {end - start} seconds! {(f_contador - 1) * 500 +
            contador} id's and {f_contador} files upload")
    }
```

9.2.2 Función AWS Lambda GetlocationsTrigger

```
# Import python libraries
import json
import os
import boto3
import csv

from datetime import datetime

# Import firebase libraries (layer)
import firebase_admin
from firebase_admin import credentials
from firebase_admin import firestore

# Import modules
import constants as constant
from batch_process import search_location
from aws_process import upload_s3 as up_s3
from ddbb_process import ddbb_mongodb

# Initializing variables

locationLastUpdate = constant.LAST DATE  # search last date
SAVE_DDBB = constant.SAVE_DDBB

s3 = boto3.client('s3',
                  aws_access_key_id="AccessKey",
                  aws secret access key="SecretId")

cred = credentials.Certificate(constant.PATH_CRED)

firebase_admin.initialize_app(cred)
db = firestore.client()

input_archive_folder = "input_archive"
to_process_folder = "to_process"


def lambda_handler(event, context):
    print("Received event: \n" + str(event))
    for record in event['Records']:
        # Assign some variables of event record.
        bucket = record['s3']['bucket']['name']  # tfm_safe_bucket
        key = record['s3']['object']['key']
        archive_path = os.path.join(bucket, input_archive_folder,
        os.path.basename(key))
        folder = os.path.split(key)[0]
        response = s3.get_object(Bucket=bucket, Key=key)
        text = response["Body"].read().decode()
```

```
48          count = 0
49
50          if not SAVE_DDBB:
51              # create csv file
52              filename = "/tmp/locations" +
                           str(datetime.now().strftime("%d%m%Y%H%M%S")) + ".csv"
53
54              f = open(filename, 'w', newline='')
55              csv_writer = csv.writer(f, delimiter=',')
56              csv_writer.writerow(constant.HEADER)
57
58          for reg in text.split("\r\n"):
59              count += 1
60              if len(reg) > 0:
61                  monitoring_id, user_id, isAlert, activate = reg.split(",")
62                  document ref = db.collection(u'monitorings')
                                 .document(monitoring_id)
63
64                  # Get location
65                  locations_ref = search_location
                                    .ddbb_extract_update_locations(document_ref,
                                    locationLastUpdate)
66                  last_lat = 0
67                  last_lon = 0
68                  for location in locations_ref:
69                      location = location.to_dict()
70                      if 'location' in location:
71                          data = location['location']
72                          latitude = data['latitude']
73                          longitude = data['longitude']
74                          UpdateAt = data['locationUpdateAt']
75                          radius = data['radius']
76                          speed = data['speed']
77                          altitude = data['altitude']
78                          activity = location['activity']
79                          if last_lat != latitude or last_lon != longitude:
80                              target_city_ok = search_location
                                .isLocatedAt(latitude, longitude, constant.TARGET_ZONE)
81                              last lat = latitude
82                              last_lon = longitude
83                          if target_city_ok:
84                              if SAVE_DDBB:
85                                  ddbb_mongodb.insert_locations(monitoring_id,
                                    user_id, isAlert, activate, latitude,
86                                    longitude, UpdateAt,
                                    radius, speed, altitude, activity,
87                                                              city="Madrid")
88                              else:
89                                  row = [monitoring_id, user_id, isAlert, activate,
                                           latitude, longitude, UpdateAt, radius,
90                                         speed, altitude, activity.upper()]
91                                  csv_writer.writerow(row)
92
93          if not SAVE_DDBB:
94              # Close file locations<date>.csv
95              f.close()
96
97              # Upload file to S3
98              up_s3.upload_file_s3(filename, constant.bucketname,
                                     object_name=None, args=None)
99
100         # Delete file monitoring_contador.csv
101         os.remove(filename)
102
103         return {
104             'statusCode': 200,
105             'body': json.dumps('Process completed!' + str(count))
106         }
```

9.3 Anexos Funciones de Geolocalización

Modulo geolocation_functions - get_geolocation.py

```
import pandas as pd
import requests
import json

def get_country_polygon(country):
    c_json = json.load(open("data/world-countries.json", "r"))
    for i in range(len(c_json['features'])):
        if (c_json['features'][i]['properties']['name'] == country):
            return c_json['features'][i]

def get_city_polygon(city):
    c_json = json.load(open("data/citiesgeojson.json", "r"))
    for i in range(len(c_json['features'])):
        if (c_json['features'][i]['properties']['NAME'] == city.upper()):
            return c_json['features'][i]

def get country geolocation(country):
    countries = pd.read_csv("data/countries_locations.csv")
    country_lat = countries.loc[countries["name"] == country,
                  ("latitude", "longitude")].values[0][0]
    country_lng = countries.loc[countries["name"] == country,
                  ("latitude", "longitude")].values[0][1]
    return country_lat, country_lng

def get_city_geolocation(country, city):
    cities = pd.read_csv("data/worldcities.csv")
    city_lat = cities.loc[(cities["country"] == country)
                          & (cities["city"] == city),("lat", "lng")].values[0][0]
    city_lng = cities.loc[(cities["country"] == country)
                          & (cities["city"] == city),("lat", "lng")].values[0][1]
    return city_lat, city_lng

def get_elevation(lat, long):
    # script for returning elevation from lat, long, based on open elevation data
    # which in turn is based on SRTM
    # Fuente: https://stackoverflow.com/questions/19513212/can-i-get-the-altitude-
              with-geopy-in-python-with-longitude-latitude

    query = ('https://api.open-elevation.com/api/v1/lookup'
                        f'?locations={lat},{long}')
    try:
        r = requests.get(query, stream=True).json()

        elevation = pd.io.json.json_normalize(r, 'results')['elevation'].values[0]
        r.close()
        return elevation

    except ConnectionError as error:
        print(error)
```

Modulo geolocation_functions – geometry_functions.py

```
import geopandas as gpd
from shapely.geometry import Polygon
from pyproj import CRS

def create polygon geometry(lat points, lon points):
    # Create a polygon objetct
    # parameters: lat_points, lon_points are list
    # Fuente: https://gis.stackexchange.com/questions/294206
      /how-to-create-a-simple-polygon-from-coordinates-in-geopandas-with-python

    polygon geom = Polygon(zip(lon points, lat points))
    crs = CRS('epsg:4326')

    polygon = gpd.GeoDataFrame(index=[0], crs=crs, geometry=[polygon_geom])
    return polygon.geometry
```

9.4 Anexos Funciones de Calidad de los Datos

Modulo data_clean_process – space_optimization.py

```python
import pandas as pd

def get_memory_used(df):
    memoria = df.memory_usage(deep=True).sum()/1024**2
    print(f"Total memoria utilizada: {memoria:0.2f} MB\n")
    for column type in ['float', 'int', 'object', "bool", "category", "datetime"]:
        columns_type = df.select_dtypes(include=[column_type])
        memory_use = columns_type.memory_usage(deep=True).mean()
        memory_use_MB = memory_use / 1024 ** 2
        print(f"El tipo de dato {column_type} utiliza {memory_use_MB:0.5f} MB
                de memoria")
    return memoria, len(df), len(df.columns)

def delete_notnumerical_data(df, column, string):
    error_id = df[df[column].str.contains(string, na=False, case=True, regex=True)]
    if len(error_id) != 0:
        df = df.drop(error id.index)
    return df

def float_downcast(df, *columnas):
    for column in columnas:
        df[column] = pd.to_numeric(df[column], downcast='float')
    return df

def convert_float_datetime(df, column):
    df[column] = df[column].astype('datetime64[ms]')
    return df

def convert_object(df, *columnas, new_type):
    for column in columnas:
        df[column] = df[column].astype(new_type)
    return df
```

Modulo data_clean_process – remove_outliers.py

```python
# Calculo de límites y Eliminar valores atípicos

def calcular_limites(df, columna):
    q1 = df[columna].quantile(.25)
    q2 = df[columna].quantile(.50)
    q3 = df[columna].quantile(.75)
    iqr = q3-q1 #Interquartile range
    li_min = q1-1.5*iqr
    li_max = q3+1.5*iqr
    le_min = q1-3*iqr
    le_max = q3+3*iqr
    return (q1,q2,q3,li_min,li_max,le_min,le_max)

def imprimir_limites(valores):
    print("Q1: ", valores[0])
    print("Q2: ", valores[1])
    print("Q3: ", valores[2])
    print("Limite interno min: ", valores[3])
    print("Limite interno max: ", valores[4])
    print("Limite externo min: ", valores[5])
    print("Limite externo max: ", valores[6])

def remove_outlier(df, columna):
    q1 = df[columna].quantile(.25)
    q3 = df[columna].quantile(.75)
    iqr = q3-q1
    li_min = q1-1.5*iqr
    li_max = q3+1.5*iqr
    df_out = df.loc[(df[columna] > li_min) & (df[columna] < li_max)]
    return df_out
```

Modulo data_clean_process – new_data.py

```
1    import ephem
2
3    def isMetropolitanM(df):
4
5        """ Función: Calcula si una ciudad forma parte de la Zona Metropolitana de
Madrid
6            Args:    Dataframe
7            Returns: True si la ciudad pertenece al Área Metropolitana
8                     False si la ciudad NO pertenece al Área Metropolitana
9            Autor: María del Carmen Sierra Fernández"""
10
11       metropolitan = ['Madrid', 'Alcobendas', 'Alcorcón', 'Boadilla del Monte',
                        'Brunete', 'Colmenar Viejo',
12                       'Coslada', 'Getafe', 'Leganés', 'Majadahonda',
                         'Mejorada del Campo', 'Paracuellos de Jarama',
13                       'Pinto', 'Pozuelo de Alarcón', 'Rivas-Vaciamadrid',
                         'Las Rozas de Madrid',
14                       'San Fernando de Henares', 'San Sebastián de los Reyes',
                         'Torrejón de Ardoz',
15                       'Velilla de San Antonio', 'Villanueva de la Cañada',
                         'Villanueva del Pardillo',
16                       'Villaviciosa de Odón', 'Móstoles', 'Parla', 'Tres Cantos',
                         'Colmenar',
17                       'Alcalá de Henares', 'Fuenlabrada', 'Collado Villalba',
                         'Torrelodones', 'Arganda del Rey']
18
19       city = df["city"]
20
21       if city in metropolitan:
22           return True
23       else:
24           return False
25
26
27   def isWeekend(df):
28
29       """ Función: se calcula si fue fin de semana (1) o entre semana (0)
30           Args: el dataframe sin los registros que han pasado los filtros
31           Returns: si fue fin de semana (1) o entre semana(0)
32           Autor: José Luis García Fuentes
33           TFG: Geoposicionamiento de objetos móviles mediante
                      plataformas GIS en la nube """
34
35       day_of_week = df["start_date"].day_name()
36
37       if day_of_week == 'Sunday' or day_of_week == 'Saturday':
38           return (True)
39       else:
40           return (False)
41
42   def isDay(df):
43
44       """ Función: se calcula si un día a una hora en una latitud y longitud
             concreta fue de día o de noche
45           Args: el dataframe sin los registros que han pasado los filtros
46           Returns: True es de día o False si es noche
47           Autor: José Luis García Fuentes
48           TFG: Geoposicionamiento de objetos móviles mediante plataformas
                      GIS en la nube"""
49
50       o = ephem.Observer()
51       o.long = df["lat_sp"]
52       o.lat = df["long_sp"]
53       o.date = df["start_date"]
54       s = ephem.Sun()
55       s.compute(o)
56
57       if s.alt > 0:
58           return (True)
59       else:
60           return (False)
```