Facultad
de
Ciencias

HERRAMIENTA PARA LA GESTIÓN Y VISUALIZACIÓN DE DATOS DE INCIDENTES DE AHOGAMIENTOS EN MEDIOS ACUÁTICOS

(Drowning Incident Data Management and Visualization Tool)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Mario Bosque Avelleira

Director: Diego García Saiz

Septiembre – 2025

Resumen

El origen de este Trabajo de Fin de Grado nace de la preocupación ante el número creciente de ahogamientos en España durante los últimos años y la necesidad de visibilizar esta problemática, facilitando su análisis tanto a profesionales como a la sociedad en general. Esta inquietud surge directamente de la Escuela Segoviana de Socorrismo, entidad que desde 2013 se encarga de recoger y analizar datos sobre ahogamientos en nuestro país.

En la actualidad, la información sobre estos incidentes se encuentra dispersa y poco accesible, perdida entre la gran cantidad de noticias que son publicadas en los diferentes medios de comunicación, lo que dificulta observar realizar análisis sobre tendencias, identificar patrones relevantes, tomar decisiones informadas y concienciar a la ciudadanía para prevenir nuevos casos. Por estos motivos, se propuso la creación de una aplicación para imprimir estos datos.

Por ello, este proyecto propone la creación de una aplicación web que centraliza, organiza y analiza los datos de ahogamientos registrados desde 2013. El sistema permite a los usuarios realizar consultas personalizadas, visualizar diferentes gráficos y mapas interactivos, y explorar cómo evolucionan los incidentes en el tiempo y el espacio. Mediante una interfaz intuitiva, se busca no solo ofrecer herramientas de análisis avanzado con técnicas de minería de datos, sino también contribuir a la sensibilización sobre el riesgo de ahogamiento y fomentar la prevención a través de información clara y comprensible.

Palabras clave: Ahogamientos, Prevención, Tendencias, Aplicación, Gráficos, Mapas

Abstract

This Bachelor's thesis stems from the growing concern regarding the increasing number of drownings in Spain in recent years and the need to raise awareness about this issue, making its analysis accessible to both professionals and the public. This initiative originates from the Escuela Segoviana de Socorrismo, an organization that has been collecting and analyzing data on drownings nationwide since 2013.

Currently, information about these incidents is scattered and difficult to access, lost among a vast array of news stories published in various media outlets. This complicates the analysis of trends, the identification of relevant patterns, informed decision-making, and public awareness to help prevent further cases. To address this, the creation of an application was proposed to organize and present these data.

Consequently, this project introduces a web application that centralizes, organizes, and analyzes drowning data recorded since 2013. The system enables users to perform customized queries, view various interactive charts and maps, and explore the evolution of incidents over time and across different locations. With an intuitive interface, the application aims not only to provide advanced analytical tools using data mining techniques but also to enhance awareness about drowning risks and promote prevention through clear and accessible information.

Keywords: Drownings, Prevention, Trends, Application, Graphs, Maps

Índice

1 Introducción

A continuación, se va a presentar el contexto en el que surge la idea de esta aplicación, los objetivos generales y específicos, la metodología empleada y la estructura de la memoria.

1.1 Contexto

El origen de este proyecto se encuentra vinculado a la iniciativa de Luis Miguel Pascual Gómez, socio fundador y Director Técnico-Docente de la asociación Escuela Segoviana de Socorrismo, quien propuso la creación de esta aplicación. Luis Miguel gestiona el proyecto de AHOGAMIENTO.COM [1], una referencia nacional que recopila y difunde información sobre incidentes de acuáticos en España.

Actualmente, el proyecto AHOGAMIENTOS.COM recopila datos de diversas fuentes, principalmente noticias y reportes oficiales, que incluyen detalles como localización, fecha, características del incidente y víctimas. Estos datos son organizados y publicados en la web de forma accesible para el público general, presentados mediante listados y mapas básicos que permiten conocer la distribución espacial y temporal de los ahogamientos. En la Figura 1, se puede observar el aspecto del blog.

Figura 1 – Interfaz inicial de AHOGAMIENTO.COM

Sin embargo, la variedad de gráficas es bastante limitada y no permite su personalización por parte de los usuarios. Entre otras carencias detectadas en el manejo actual de estos datos está la ausencia de modelos analíticos complejos, como *machine learning*, que permitan identificar causas y factores de riesgo de los ahogamientos.

También se evidencia la falta de una interfaz sencilla y usable para que los administradores puedan actualizar y gestionar los datos de forma eficiente.

1.2 Objetivos

Como se ha comentado, el objetivo principal es el desarrollo de una aplicación interactiva que permita el análisis y la visualización de los datos de ahogamientos en España desde 2013. En cuanto a los objetivos específicos:

- Realizar la transición de la estructura de la base de datos a un modelo más usado, moderno y escalable, como se indica en la Tabla 12.
- Llevar a cabo la depuración de los datos para eliminar errores.
- Implementar funcionalidades de visualización dinámica, personalizables y descargables (gráficos y mapas) que faciliten la interpretación de los datos por parte del público general.
- Incorporar técnicas de *machine learning* cuya ejecución y resultados puedan ser personalizados e interpretados por usuarios no expertos.
- Implementación de un proceso para mejorar la toma y almacenamiento de nuevos casos de ahogamientos por parte de los administradores.

1.3 Estructura de la memoria

A partir del punto 2, se indica la metodología que se ha seguido para desarrollar el proyecto. En el punto 3, se describen las tecnologías y herramientas empleadas en el desarrollo, explicando su funcionalidad y papel dentro del proyecto. En el 4, se presentan los requisitos funcionales y no funcionales.

El punto 5 aborda la arquitectura y diseño del sistema, describiendo sus capas principales y el flujo de datos entre ellas. En el punto 6 se detalla la implementación, desde la gestión de datos hasta el desarrollo de la aplicación y la integración de algoritmos de *machine learning*.

En el punto 7 se exponen las pruebas realizadas para verificar el correcto funcionamiento y la usabilidad de la aplicación, así como su rendimiento. Posteriormente, en el punto 8 se presenta el resultado final del proyecto.

Las conclusiones del proyecto se encuentran en el apartado 9, mientras que en el punto 10 se plantean posibles líneas futuras de desarrollo. Por último, en el punto 11 se encuentran las referencias con sus enlaces.

2 Metodología

A continuación, se detalla la metodología seguida para el desarrollo de esta aplicación, describiendo las fases principales:

- **Análisis y definición de objetivos**. En esta fase se definieron las metas principales del proyecto y se realizó un análisis inicial de cómo debía estructurarse la base de datos para gestionar eficientemente la información sobre ahogamientos.
- **Limpieza de datos**. Para asegurar la calidad de los datos, se realizó una limpieza y depuración utilizando OpenRefine. Esta fase incluyó la eliminación de duplicados, corrección de errores comunes, unificación de formatos y normalización de valores.
- **Migración de datos**. Para migrar los datos desde los archivos Excel a la base de datos SQLite, se desarrolló un proceso ETL (*Extract, Transform, Load*) utilizando Python. Este procedimiento tiene como objetivo extraer la información de los archivos, transformarla para adecuarla a la estructura de la base de datos y cargarla de forma automatizada en formato .db, facilitando así la integración y posterior análisis de los datos en la aplicación.
- **Desarrollo de la aplicación**. Esta fase de la aplicación fue la más extensa y compleja, ya que incluyó la selección de librerías adecuadas para la visualización de datos, como Bokeh, y la incorporación de librerías de *machine learning* para análisis avanzados. Además, fue necesario adquirir conocimientos y manejar el *framework* Django para construir el *backend* y gestionar la base de datos, así como desarrollar el código web para la interfaz de usuario utilizando tecnologías *frontend*. Este proceso integró la programación en Python y la implementación de funcionalidades que permitieran una interacción dinámica, visualizaciones atractivas y un análisis predictivo eficiente. Para organizar el desarrollo se optó por una metodología incremental, dividiendo el proyecto en módulos o incrementos independientes que se planificaron, desarrollaron y entregaron por etapas.
- **Pruebas y validación**. Se verificó el correcto funcionamiento de todas las funcionalidades implementadas en la aplicación. Además, se llevaron a cabo pruebas de usabilidad para asegurar que la interfaz fuera intuitiva y accesible para los usuarios, como se indica en la Tabla 11.
- **Documentación y entrega**. Finalmente, se procedió a la elaboración de la memoria del Trabajo de Fin de Grado, en la que se recogen el proceso seguido hasta concluir con el proyecto.

En la Figura 2, se muestra un cronograma sobre la duración de cada etapa a lo largo del desarrollo.

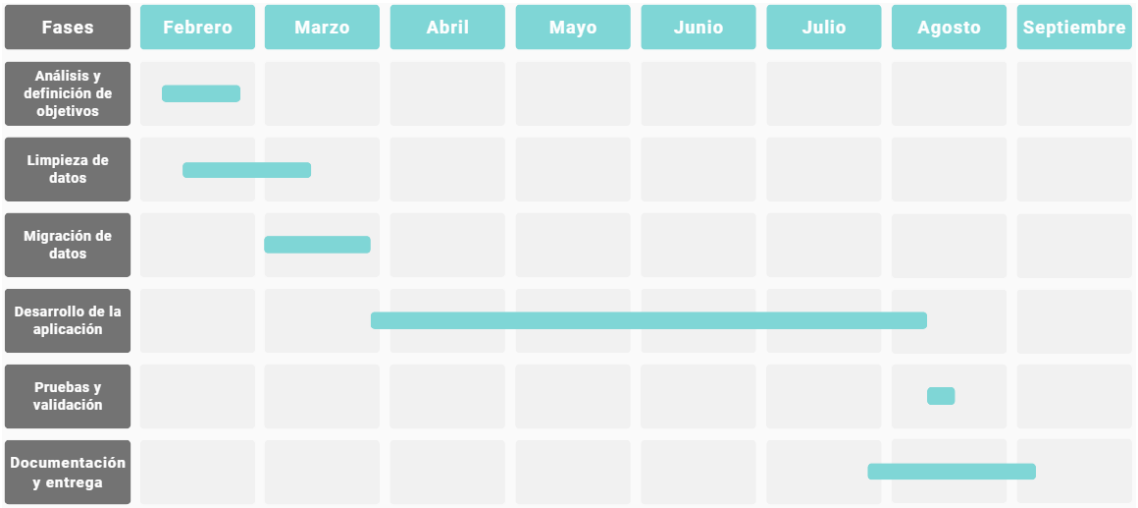


Figura 2 – Cronograma del proyecto

3 Tecnologías y herramientas

En esta sección, se recogerán todas las tecnologías que han sido usadas para este proyecto.

3.1 Visual Studio Code [2]

Se trata de un editor de código fuente multiplataforma desarrollado por Microsoft. Se ha empleado para la programación y desarrollo de la aplicación, facilitando la escritura, organización y depuración del código en Python, así como el manejo de archivos de configuración y estilos (HTML, CSS y JavaScript). Ofrece una amplia variedad de extensiones que han sido útiles durante el desarrollo, como soporte para Django, gestión de bases de datos SQLite y herramientas para el control de versiones mediante Git. Gracias a su interfaz intuitiva y funcionalidades avanzadas, ha permitido agilizar las tareas de edición, pruebas y mantenimiento del proyecto.

3.2 Django [3]

Es un *framework* de desarrollo aplicación en Python utilizado para la creación de la estructura y lógica de la aplicación. Con Django se han gestionado las rutas, modelos, vistas y toda la lógica de servidor, facilitando la integración con la base de datos SQLite y la escalabilidad del proyecto. Para asegurar un control eficiente de las dependencias y versiones de los paquetes utilizados, se ha creado y gestionado un entorno virtual de Python específico para el proyecto. Esto ha permitido trabajar de forma aislada y reproducible, evitando conflictos entre librerías y manteniendo un entorno de desarrollo limpio y organizado.

3.3 Python [4]

Es el lenguaje de programación principal del proyecto, como se muestra en la Tabla 13. Es empleado tanto para el desarrollo de la lógica de la aplicación como para la manipulación y análisis de datos. Python ha permitido integrar fácilmente el *framework* Django, la gestión de bases de datos con SQLite, la creación de visualizaciones y el desarrollo de modelos de *machine learning* mediante su gran selección de librerías. Su sintaxis clara y la amplia disponibilidad de librerías han facilitado el desarrollo modular, escalable y eficiente de todas las funcionalidades de la aplicación.

3.4 Git [5]

Es un sistema de control de versiones distribuido utilizado para gestionar y registrar los cambios realizados en el código fuente del proyecto. Git permite mantener un historial completo de modificaciones y recuperar archivos a estados anteriores. Aunque el desarrollo haya sido realizado por una única persona, la utilización de esta tecnología ha ayudado para compartir fácilmente las actualizaciones del código con mi tutor.

3.5 SQLite [6]

Se trata de un sistema de gestión de bases de datos relacional, muy ligero en cuanto a espacio, utilizado para almacenar, consultar y manipular los datos del proyecto de forma sencilla, ligera y eficiente. La gestión y visualización de la base de datos se ha realizado mediante SQLiteStudio, una herramienta gráfica multiplataforma que permite crear, editar y gestionar bases de datos SQLite de forma intuitiva, sin requerir conocimientos avanzados de SQL. Esta combinación ha facilitado la administración, el seguimiento de los datos y su integración con el resto de la aplicación.

3.6 Scikit-Learn [7]

Es una librería de código abierto para Python, orientada al aprendizaje automático y ampliamente utilizada en ciencia de datos. Scikit-Learn proporciona una gran variedad de algoritmos y herramientas para tareas de clasificación, regresión, agrupamiento y reducción de dimensionalidad, todo ello a través de una interfaz simple y coherente, que permite utilizar abstraer al programador de los cálculos que hay detrás de cada función.

3.7 Bokeh [8]

Es una biblioteca interactiva de visualización de datos para Python, utilizada en el proyecto para la creación de gráficos y visualizaciones avanzadas dentro de la aplicación. Gracias a Bokeh, ha sido posible implementar distintos tipos de gráficas y representaciones analíticas. La integración de Bokeh permite ofrecer al usuario visualizaciones interactivas, personalizables y de gran calidad, permitiendo también su descarga.

3.8 OpenRefine [9]

Esta aplicación ofrece una gran facilidad para realizar la limpieza de los datos, detectando posibles fallos ortográficos que dividían selecciones de datos que deberían ir juntas (ejemplo, “Cáceres”, “Caceres” y “caceres”). La aplicación se encarga de unificar estos valores. Gracias a su interfaz intuitiva y funcionalidades avanzadas, la herramienta fue clave para preparar los conjuntos de datos antes de su migración y análisis posterior, asegurando así una base de datos limpia, coherente y de calidad para el resto del proceso.

3.9 MagicDraw [10]

Es una herramienta de modelado UML que facilita la creación de diagramas visuales para el diseño y análisis de sistemas de software. Fue usada en una etapa muy inicial del proyecto para conseguir un visionado general de la base de datos.

3.10 Otras herramientas

También se han usado, en menor o mayor medida, las siguientes herramientas:

- Otras librerías de Python (pandas, numpy, spicy, geopandas, itertools...)
- Archivo de tipo *shapefile* con el mapa de España dividido por provincias, descargado de la web del Instituto Geográfico Nacional [11].
- En el caso del mapa de zonas calientes, se ha usado la web OpenStreetMap [12].

4 Análisis de requisitos

A continuación, se analizan en detalle los requisitos necesarios para el correcto desarrollo y funcionamiento de la aplicación.

4.1 Requisitos funcionales

En este apartado se definirán todos los requisitos funcionales del proyecto, indicando que servicios se ofrecerán y que respuesta obtiene el usuario.

Identificador	RF001
Requisito	Selecciona el tipo de gráfica o mapa
Descripción	El usuario puede seleccionar el tipo de gráfica o mapa que desea visualizar. Se ofrecen gráficas de diversos tipos y que cubren campos diferentes de la base de datos.
Flujo principal	1- El usuario selecciona el tipo de gráfica o mapa deseado. 2- El sistema cambia los filtros que estaba mostrando en la parte central y pasa mostrar los de la gráfica seleccionada

Tabla 1 – Requisito funcional 001

Identificador	RF002
Requisito	Selecciona los filtros para la gráfica o mapa
Descripción	El usuario quiere poder seleccionar y configurar los diferentes selectores, casillas y botones, para personalizar la información que se mostrará en la gráfica o mapa.
Flujo principal	1- El usuario accede a los filtros disponibles. 2- El usuario selecciona y configura uno o varios filtros según sus necesidades. 3- El sistema almacena las selecciones de filtros para usarlas en la generación de la gráfica o mapa.
Flujo alternativo	- Si algún filtro está incompleto o inválido, el sistema devuelve un texto en el que notifica el fallo al usuario. - Si no se selecciona ningún filtro, el sistema puede mostrar un mensaje para recordar que al menos un filtro es necesario para generar resultados relevantes.

Tabla 2 – Requisito funcional 002

Identificador	RF003
Requisito	Eliminar los filtros seleccionados para la gráfica o mapa
Descripción	El usuario quiere poder eliminar fácilmente cualquiera de los filtros que haya seleccionado previamente, para modificar la configuración de la gráfica o mapa de forma rápida y sin complicaciones. Podrá realizar esta acción eliminando cada selección, una a una, o todas de golpe, pulsando el botón “Limpiar filtros”.
Flujo principal	1- El usuario visualiza las opciones de filtro actualmente activas. 2.1- El usuario selecciona uno o varios filtros activos para eliminarlos. 2.2- El usuario elimina todos los filtros pulsando “Limpiar filtros”. 3- El sistema actualiza los filtros seleccionados y refleja los cambios para la generación de la gráfica o mapa.
Flujo alternativo	- Si el usuario no confirma la eliminación, los filtros permanecen activos.

Tabla 3 – Requisito funcional 003

Identificador	RF004
Requisito	Ofrecer la gráfica o mapa al usuario con los filtros elegidos
Descripción	El sistema debe mostrar al usuario la gráfica o mapa generado en base a los filtros que ha seleccionado, asegurando que la visualización refleje correctamente la información solicitada.
Flujo principal	1- El usuario pulsa el botón “Generar grafica/mapa”. 2- El sistema procesa los filtros y genera la gráfica o mapa correspondiente. 3- La gráfica o mapa se muestra dinámicamente al usuario, permitiendo la interacción con la visualización.
Flujo alternativo	- Si los filtros están incompletos o contienen errores, el sistema muestra un mensaje notificando al usuario. - Si no hay datos disponibles para los filtros seleccionados, el sistema informa al usuario de esta situación.

Tabla 4 – Requisito funcional 004

Identificador	RF005
Requisito	Descargar la gráfica
Descripción	El usuario debe poder descargar la gráfica o mapa mostrado en la aplicación en formato .png, permitiéndole guardar la visualización para su posterior uso, análisis o inclusión en presentaciones e informes.
Flujo principal	1- El usuario pulsa el botón de descargar. 2- El sistema indica al usuario que elija un nombre para el documento. 3- El sistema genera un archivo en formato .png de la visualización actual y lo descarga automáticamente en el dispositivo del usuario.

Tabla 5 – Requisito funcional 005

Identificador	RF006
Requisito	Añadir nuevas víctimas y/o incidentes a la base de datos
Descripción	El administrador debe poder añadir manualmente nuevas víctimas y/o incidentes a la base de datos, proporcionando la información necesaria conforme a los campos definidos.
Flujo principal	1- El administrador accede a la interfaz de gestión. 2- Elige la opción para añadir una nueva víctima o incidente. 3- Completa los campos obligatorios del formulario, incluyendo los datos específicos de la víctima/incidente. 4- Revisa la información introducida. 5- Confirma y guarda el nuevo registro. 6- El sistema almacena el nuevo caso en la base de datos y muestra un mensaje de éxito.
Flujo alternativo	- Si falta algún campo obligatorio, el sistema señala el error y no permite guardar hasta completar la información. - Si ocurre un error durante el guardado, el sistema muestra un mensaje indicando el problema y mantiene el formulario para su revisión o corrección.

Tabla 6 – Requisito funcional 006

Identificador	RF007
Requisito	Editar víctimas y/o incidentes de la base de datos
Descripción	El administrador debe poder modificar los datos de víctimas y/o incidentes ya registrados en la base de datos, para actualizar información o corregir errores manteniendo la integridad del sistema. Para ello, deberá introducir el identificador correspondiente (codVictima o codIncidente) que permita localizar el registro a editar.
Flujo principal	1- El administrador accede a la interfaz de gestión. 2- Selecciona la opción de editar víctima o incidente. 3- Introduce el identificador único (codVictima o codIncidente) del registro a modificar. 4- El sistema localiza y muestra los datos actuales del registro. 5- El administrador edita los campos deseados. 6- Guarda los cambios. 7- El sistema actualiza la base de datos y confirma la edición.
Flujo alternativo	- Si el identificador introducido no corresponde a ningún registro, el sistema notifica al administrador y no permite la edición. - Si ocurre un error al guardar los cambios, el sistema informa del problema y mantiene la información editable para su corrección. - Si alguno de los campos que han sido editados queda vacío, y su contenido selección es obligatoria, se notifica al usuario y no se permite la edición.

Tabla 7 – Requisito funcional 007

Identificador	RF008
Requisito	Eliminar víctimas y/o incidentes de la base de datos
Descripción	El administrador debe poder eliminar registros de víctimas y/o incidentes existentes en la base de datos, asegurando el borrado permanente de la información seleccionada cuando sea necesario por motivos de corrección, depuración o actualización. Para identificar el registro a eliminar, deberá introducir el identificador correspondiente (codVictima o codIncidente).
Flujo principal	1- El administrador accede a la interfaz de gestión. 2- Selecciona la opción de eliminar víctima o incidente. 3- Introduce el identificador único (codVictima o codIncidente) del registro a eliminar. 4- El sistema localiza y muestra los datos del registro. 5- El administrador confirma la eliminación. 6- El sistema elimina el registro de la base de datos y muestra un mensaje de éxito.
Flujo alternativo	- Si el identificador introducido no corresponde a ningún registro, el sistema lo notifica. - Si ocurre un error durante el proceso, el sistema informa al administrador y no elimina la información hasta su correcta validación.

Tabla 8 – Requisito funcional 008

4.2 Requisitos no funcionales

A continuación, se muestran los requisitos no funcionales que establecen las cualidades, restricciones y condiciones bajo las cuales el sistema debe operar. Estos requisitos son fundamentales para asegurar que la plataforma funcione de manera adecuada y satisfaga las expectativas de los usuarios más allá de la simple funcionalidad, proporcionando una experiencia fluida, confiable y segura.

Identificador	RNF001
Requisito	Rendimiento
Descripción	El sistema debe generar las gráficas y mapas en un tiempo razonable para asegurar una experiencia de usuario fluida.

Tabla 9 – Requisito no funcional 001

Identificador	RNF002
Requisito	Seguridad
Descripción	El sistema debe proteger los datos almacenados mediante mecanismos de autenticación, autorización y encriptación para garantizar la confidencialidad e integridad de la información.

Tabla 10 – Requisito no funcional 002

Identificador	RNF003
Requisito	Usabilidad
Descripción	La interfaz debe ser intuitiva y accesible, permitiendo una navegación sencilla para usuarios.

Tabla 11 – Requisito no funcional 003

Identificador	RNF004
Requisito	Escalabilidad
Descripción	La arquitectura debe soportar aumentos en el volumen de datos sin degradar el rendimiento.

Tabla 12 – Requisito no funcional 004

Identificador	RNF005
Requisito	Lenguaje de programación
Descripción	Python fue seleccionado como lenguaje principal debido a sus avanzadas librerías para procesamiento numérico.

Tabla 13 – Requisito no funcional 005

5 Arquitectura y diseño

Aquí se detalla la distribución lógica y física de los módulos de software, las tecnologías seleccionadas, los patrones de diseño empleados, así como las decisiones que garantizan escalabilidad, seguridad y facilidad de mantenimiento.

Se ha seguido una arquitectura de tres capas para el diseño y desarrollo de la aplicación, como se observa en la Figura 3. Esta estructura divide el sistema en tres niveles lógicos y funcionales claramente diferenciados: la capa de presentación, que es la interfaz con la que interactúan los usuarios; la capa de lógica de negocio, encargada de procesar las peticiones, aplicar reglas y gestionar la lógica del sistema; y la capa de datos, que administra el almacenamiento y recuperación de la información.

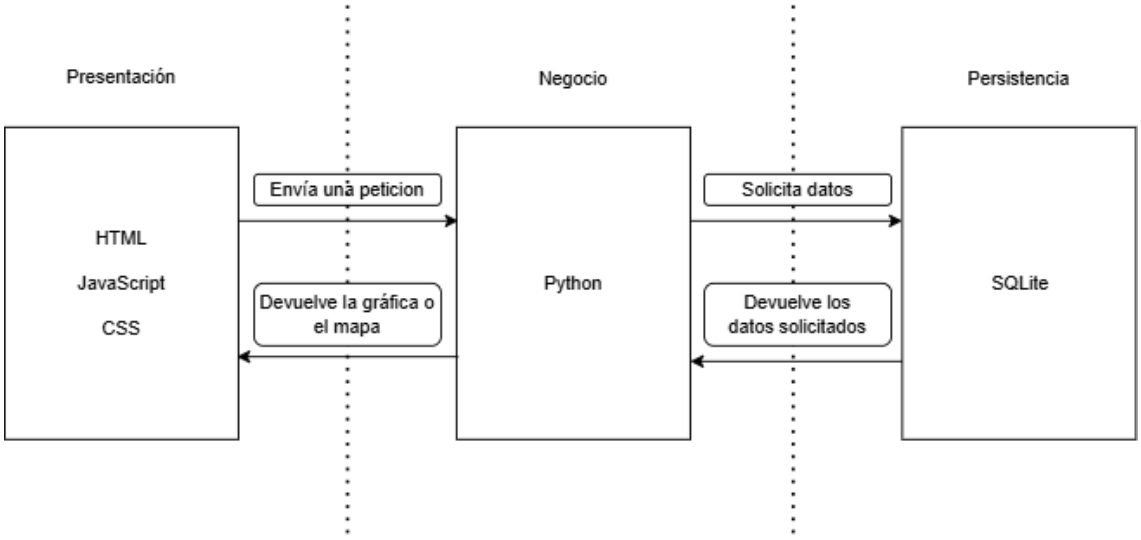


Figura 3 – Diseño arquitectónico

5.1 Capa de presentación

La capa de presentación es la parte del sistema con la que el usuario interactúa directamente. Su objetivo principal es mostrar la información de manera clara y accesible, así como capturar las acciones del usuario (como clics, selecciones y entradas de datos) para enviarlas al sistema para su procesamiento.

Esta capa incluye todos los elementos visuales como páginas web, botones, formularios y gráficos, y se encarga de ofrecer una experiencia amigable e intuitiva, asegurando que el usuario pueda navegar y usar la aplicación de forma eficiente. En el caso de esta aplicación, se han utilizado tecnologías fundamentales para el desarrollo del *frontend* como HTTP, JavaScript y CSS, como se aprecia en la Figura 4.

Esta capa realiza peticiones HTTP a la capa de negocio, utilizando el método “fetch” de JavaScript para intercambiar datos entre el *frontend* y *backend*. Estas peticiones envían y reciben datos en formato JSON, que es el estándar en aplicaciones modernas para la comunicación entre el navegador y el servidor.

Cuando un usuario selecciona filtros, configura opciones o pulsa un botón para generar una gráfica o mapa, el JavaScript recopila esta información y la envía al *backend* mediante una petición HTTP. El servidor procesa la solicitud, aplica la lógica necesaria y responde también en formato JSON con los datos o instrucciones que la capa de presentación necesita para renderizar la visualización solicitada.

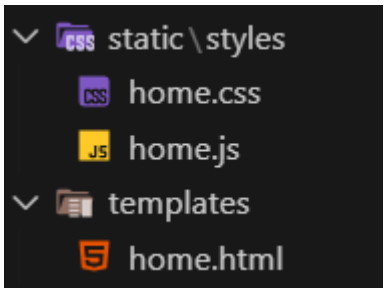


Figura 4 – Archivos que componen la capa de presentación

El archivo home.html es la plantilla HTML principal que estructura la página de inicio de la aplicación. En él se definen los elementos visuales como botones, menús, contenedores para la visualización de gráficos y filtros interactivos. En la Figura 5, se puede observar la parte del código en la que se definen los botones de selección de gráfica.

```
<!-- Panel lateral para selección del tipo de gráfica -->
<div class="selector-graficas">
  <h3>Selecciona tipo de gráfica</h3>
  <ul>
    <!-- Botones para cambiar entre distintos tipos de gráficas -->
    <li><button id="btn-grafica-apilada" type="button" class="active">Gráfica apilada</button></li>
    <li><button id="btn-grafica-circular" type="button">Gráfica circular</button></li>
    <li><button id="btn-grafica-lineas" type="button">Gráfica de líneas</button></li>
    <li><button id="btn-diagrama-dispersion" type="button">Diagrama de dispersión</button></li>
    <li><button id="btn-mapa" type="button">Mapa de España</button></li>
    <li><button id="btn-histograma" type="button">Histograma</button></li>
    <li><button id="btn-radar" type="button">Gráfica de radar</button></li>
  </ul>
</div>
```

Figura 5 – Extracto de código de home.html

En cuanto al archivo home.js, este script JavaScript gestiona la interactividad de la página de inicio. Controla eventos de los botones que seleccionan diferentes tipos de gráficos, alterna la visibilidad y el estado activo de los filtros asociados, y maneja selecciones multiselección para comunidades y provincias. Además, se

encarga de realizar peticiones asíncronas al servidor para obtener datos filtrados y actualizar las visualizaciones en tiempo real. En la Figura 6, se puede observar una parte del código en la que se activa y desactiva el desplegable de selección de lugar.

```
// Multi-select de lugares
const lugHeader = document.getElementById('lugar-header');
const lugOptions = document.getElementById('lugar-options');
const lugPills = document.getElementById('lugar-pills');
const lugCheckboxes = lugOptions.querySelectorAll('input[type="checkbox"]');
const lugMultiSelect = lugOptions.parentElement;

// Abrir/cerrar menú lugares
lugHeader.addEventListener('click', function(e) {
    if (lugHeader.classList.contains('disabled')) return;
        lugMultiSelect.classList.toggle('open');
});

// Cerrado al hacer click fuera
document.addEventListener('click', function(e) {
    if (!lugHeader.contains(e.target) && !lugOptions.contains(e.target)) {
        lugMultiSelect.classList.remove('open');
    }
});
```

Figura 6 – Extracto de código de home.js

Finalmente, el archivo home.css contiene las definiciones de estilos para la página web, asegurando una presentación visual coherente y atractiva. Controla aspectos como colores, tipografía, disposición en pantalla y efectos visuales para los diferentes componentes de la interfaz, como se observa en la Figura 7.

```
.contenedor-grafica {
  flex: 1 1 0;
  min-width: 900px;
  min-height: 600px;
  background: #fff;
  border-radius: 16px;
  box-shadow: 0 2px 12px rgba(21,154,156,0.07);
  padding: 32px 32px;
  margin-left: 0;
  display: flex;
  align-items: center;
  justify-content: center;
}
```

Figura 7 – Extracto de código de home.css

5.2 Capa de negocio

La capa de negocio es el componente central de la arquitectura de la aplicación, que se encarga de procesar las reglas del sistema, la gestión de datos, la validación de operaciones y la coordinación entre la capa de presentación (*frontend*) y la capa de persistencia (datos).

Cumple la función de recibir la solicitud desde la capa de presentación, y aplica la lógica requerida, interactúa con la capa de persistencia para obtener los datos requeridos y retorna una respuesta.

En el contexto específico de esta aplicación, estas solicitudes son recibidas por funciones específicas dentro de Django, que actúan como puntos de entrada para el procesamiento. La capa de negocio valida y transforma los datos recibidos, traduciéndolos en consultas sobre el modelo de datos que utiliza el *framework*, y ejecuta la lógica particular necesaria para este proyecto, incluyendo agregaciones, cálculos estadísticos y reglas de bloqueo entre filtros.

Una vez que los datos han sido procesados y listos para la visualización o gestión, la capa de negocio prepara una respuesta adecuada y la envía en formato JSON.

En la Figura 8, se puede observar los archivos que componen esta capa.

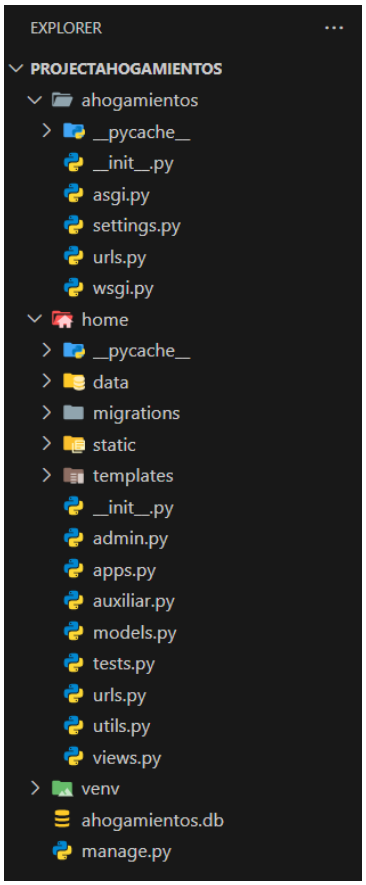


Figura 8 – Archivos que componen la capa de negocio

En la carpeta principal del proyecto, se encuentran los módulos centrales de configuración y despliegue. El archivo __init__.py indica que el directorio corresponde a un paquete de Python. Settings.py contiene la configuración global del proyecto, incluyendo parámetros de seguridad, rutas y la gestión de la base de datos. Urls.py establece las rutas principales.

En la aplicación interna home, nuevamente __init__.py define el paquete. El archivo admin.py personaliza el panel de administración de Django, permitiendo gestionar los modelos de datos desde la interfaz web. Apps.py identifica y configura la app dentro del proyecto. El módulo models.py define la estructura y los modelos de la base de datos, representando las entidades principales gestionadas en la aplicación, como se observa en la Figura 9.

```python
class Riesgo(models.Model):
    codriesgo = models.AutoField(db_column='codRiesgo', primary_key=True)  # Field name made lowercase.
    nombreriesgo = models.CharField(db_column='nombreRiesgo')  # Field name made lowercase.

    class Meta:
        managed = False
        db_table = 'Riesgo'

    def __str__(self):
        return self.nombreriesgo


class Tipoahogamiento(models.Model):
    codtipoahogamiento = models.AutoField(db_column='codTipoAhogamiento', primary_key=True)  # Field name made lowercase.
    nombretipoahogamiento = models.CharField(db_column='nombreTipoAhogamiento')  # Field name made lowercase.

    class Meta:
        managed = False
        db_table = 'TipoAhogamiento'

    def __str__(self):
        return self.nombretipoahogamiento
```

Figura 9 – Extracto de código de models.py

Views.py gestiona la lógica asociada a las vistas, procesando las peticiones recibidas y generando las respuestas y páginas que verá el usuario. Por último, urls.py organiza y define las rutas propias de la app home, como se ve en la figura 10, mientras que utils.py contiene funciones generales de utilidad que pueden ser reutilizadas por varios módulos.

```python
from django.urls import path
from . import views

urlpatterns = [
    path('generar_grafica_apilada/', views.generar_grafica_apilada, name='generar_grafica_apilada'),
    path('generar_grafica_circular/', views.generar_grafica_circular, name='generar_grafica_circular'),
    path('generar_grafica_lineas/', views.generar_grafica_lineas, name='generar_grafica_lineas'),
    path('generar_diagrama_dispersion/', views.generar_diagrama_dispersion, name='generar_diagrama_dispersion'),
    path('generar_mapa/', views.generar_mapa, name='generar_mapa'),
    path('generar_histograma/', views.generar_histograma, name='generar_histograma'),
    path('generar_radar/', views.generar_radar, name='generar_radar'),
    path('comparativa_sklearn/', views.comparativa_sklearn, name='comparativa_sklearn'),
    path('generar_mapa_hotspots/', views.generar_mapa_hotspots, name='generar_mapa_hotspots'),
    path('generar_kmeans/', views.generar_kmeans, name='generar_kmeans'),
    path('generar_tree/', views.generar_tree, name='generar_tree'),
    path('', views.home),
]
```

Figura 10 – Extracto de código de urls.py

El archivo manage.py, en la raíz del proyecto, sirve como entrada principal para la gestión administrativa de la aplicación en Django, permitiendo ejecutar comandos como migraciones, inicio del servidor y pruebas automatizadas.

5.3 Capa de persistencia

La capa de persistencia es la responsable de almacenar, consultar, actualizar y asegurar la integridad de los datos utilizados por el sistema. Su objetivo es abstraer a la capa de negocio de cómo se obtienen, guardan o modifican los datos. El tipo de base de datos es SQLite 3 y se pueden observar todas sus tablas en la Figura 11.

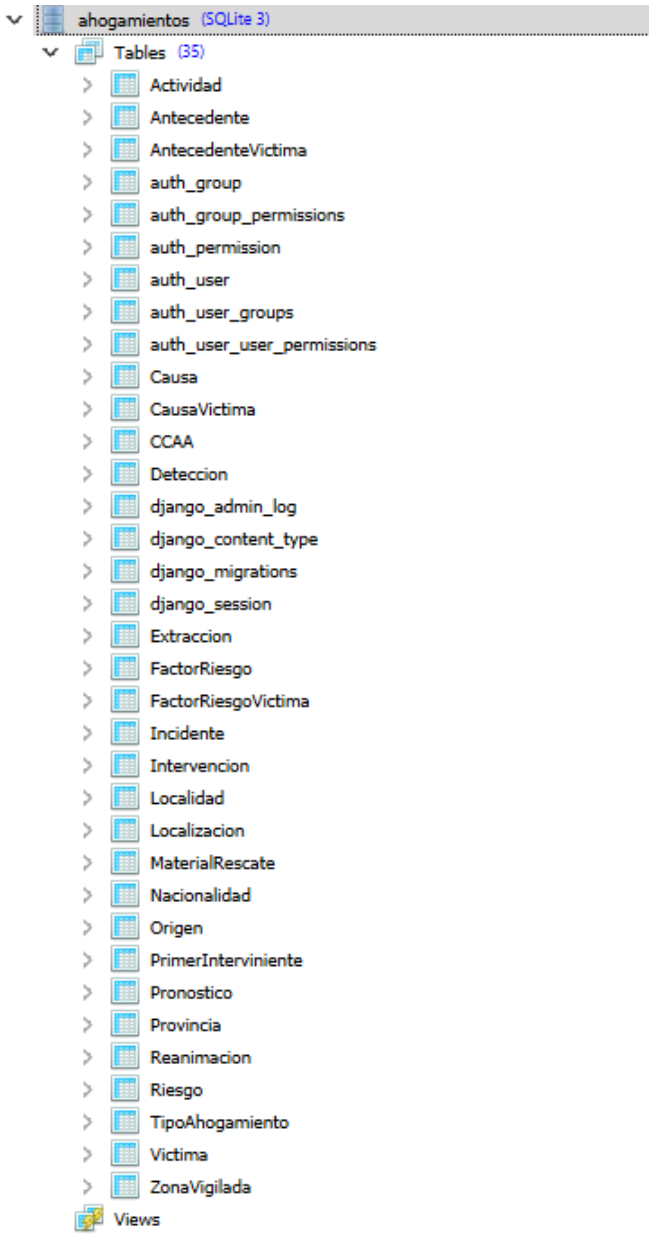


Figura 11 – Esctructura de la base de datos SQLite

5.3.1 Modelo conceptual UML

Para complementar la capa de persistencia y facilitar la comprensión estructural del sistema, se ha desarrollado un modelo conceptual utilizando diagramas UML, observable en la Figura 12. Este modelo permite representar visualmente los principales componentes, sus relaciones y la arquitectura básica de la aplicación. Esta ilustración sirvió para orientar como iba a ser la transformación de dos documentos tipo Excel denominados “incidentesCurado.xlsx” y “victimasCurado.xslx”, a un documento .db.

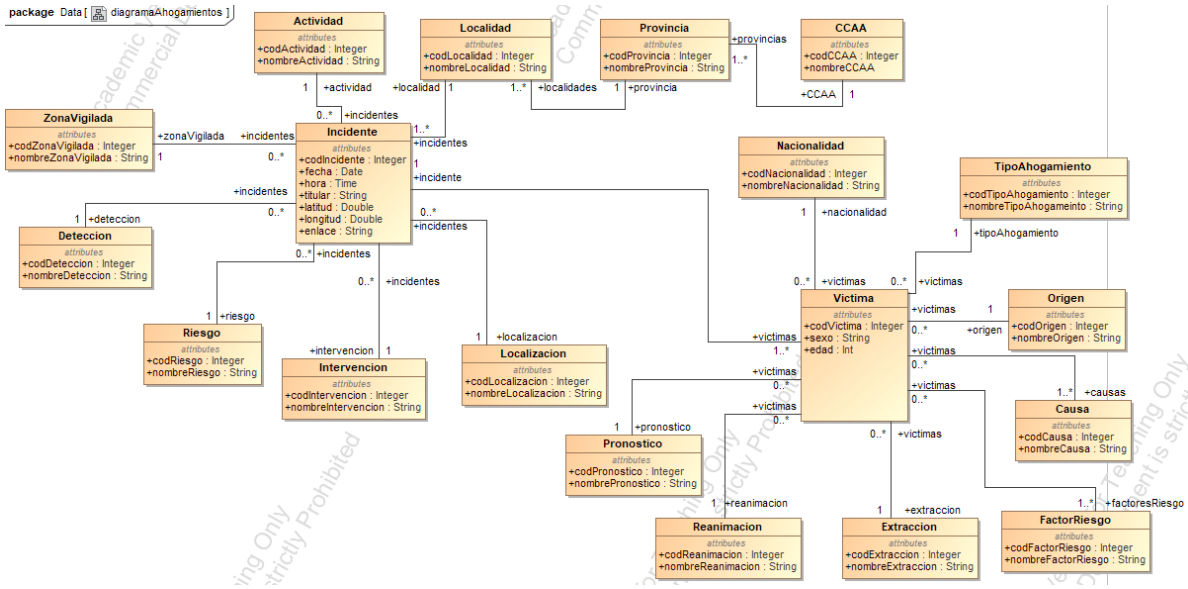


Figura 12 – Representación visual mediante UML

El diagrama UML presentado refleja el modelo conceptual de la base de datos del proyecto, centrado principalmente en las clases Incidente y Víctima, que funcionan como los núcleos fundamentales de la estructura de información. Ambas clases están altamente relacionadas con el resto de las entidades, que actúan como tablas auxiliares o subclases para completar y detallar la información relevante de cada registro.

Cabe destacar que los valores que pueden tomar las distintas subtablas fueron definidos previamente en la base de datos original del proyecto. Es decir, estas entidades no contienen datos arbitrarios (a excepción de la tabla “Localidad” y “Nacionalidad”), sino listas de valores ya establecidas y definidas en la estructura de datos original, garantizando la coherencia y continuidad al migrar y normalizar la información en el nuevo modelo relacional. Además, se tomó la decisión deliberada de no modificar ninguno de los valores existentes ni añadir nuevos durante la restructuración, precisamente para no dificultar el proceso de migración y asegurar que los datos históricos pudieran integrarse de forma directa y sin pérdida de información.

La clase Incidente representa cada suceso de ahogamiento y centraliza las relaciones con otras entidades que aportan contexto, como Actividad (actividad realizada en el momento del incidente), ZonaVigilada (si la zona contaba con vigilancia), Localidad (ubicación geográfica del incidente), Provincia y CCAA, Riesgo (condiciones en las que se encontraba el agua), Localización (el tipo de lugar en que se produce el incidente), Detección (quién detectó el suceso) e Intervención (asistencia o rescate efectuado).

También la fecha, la hora y las coordenadas del suceso (latitud y longitud) forman parte directamente de la tabla Incidente, ya que son valores específicos y únicos para cada registro, a diferencia de los atributos normalizados en subtablas que suelen ser categorías o clasificaciones reutilizables entre distintos casos. Además, la tabla Incidente incluye los campos "titular" y "enlace", que almacenan el titular de la noticia y el enlace a la fuente informativa correspondiente, proporcionando así contexto adicional y permitiendo consultar la información original relacionada con cada suceso registrado.

Por su parte, la clase Víctima facilita el almacenamiento y gestión de los datos personales y características de la persona implicada, asociándola a otras tablas como Nacionalidad, Antecedente (posibles enfermedades previas), Pronóstico (estado anticipado o evolución esperada de la víctima), Reanimación (intervenciones de reanimación aplicadas), MaterialRescate (equipos y materiales utilizados en el rescate), Extracción (método empleado para extraer a la víctima del lugar), PrimerInterviniente (persona o entidad que realizó la primera intervención), Origen (relación de la víctima con el lugar del incidente), Causa (motivo o razón que provocó en mayor medida el incidente), FactorRiesgo (condición que aumentó la probabilidad del incidente) y TipoAhogamiento (categoría del tipo de ahogamiento).

En cuanto a las relaciones, el diagrama muestra distintas cardinalidades (1:N, N:N), lo que permite vincular múltiples incidentes a una provincia, varias víctimas a un mismo incidente o asociar factores de riesgo y causas específicas de manera organizada.

6 Implementación

A continuación, se describen los pasos seguidos desde el inicio del proyecto hasta la obtención del resultado final.

6.1 Limpieza y normalización con OpenRefine

Para garantizar la consistencia y fiabilidad de la información, antes de iniciar el proceso ETL, se llevó a cabo una fase de limpieza y normalización de los datos utilizando la herramienta OpenRefine.

Esta aplicación proporciona potentes ayudas automáticas mediante sus algoritmos de agrupación por similitud, como el *clustering* de valores, visible en la Figura 13. Estas funciones permiten identificar automáticamente variantes ligeramente distintas de un mismo dato, ya sea por errores tipográficos, diferencias en el uso de mayúsculas/minúsculas o por pequeñas alteraciones en la escritura, sugiriendo su unificación bajo un mismo valor. Así, es posible consolidar de forma semiautomática los datos redundantes simplemente aceptando las agrupaciones propuestas por la herramienta.

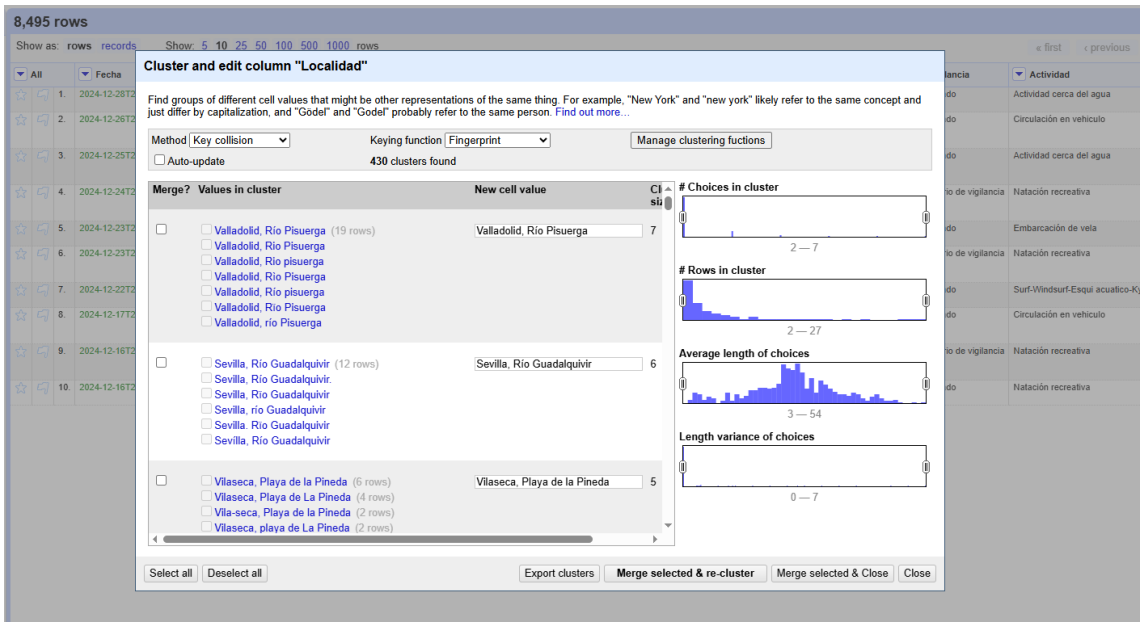


Figura 13 – Ejemplo de la función de "clustering"

6.2 Desarrollo y función del ETL

Se decidió desarrollar un proceso ETL (Extracción, Transformación y Carga) para transferir los datos almacenados en varios archivos Excel, los cuales fueron generados originalmente a partir de una base de datos Microsoft Access. El objetivo principal de este proceso es migrar y organizar dicha información a una base de datos SQLite en formato .db, siguiendo el esquema relacional establecido en el modelo representado en la Figura 12.

Este enfoque permite convertir los datos crudos y dispersos en los archivos Excel en un sistema estructurado y optimizado para consultas, análisis y desarrollo de nuevas aplicaciones.

En la fase de extracción, se leen los distintos archivos Excel que contienen información relacionada con incidentes de ahogamiento, víctimas, y entidades auxiliares (como actividades, causas, localidades, entre otros).

Durante la transformación, se procesan estos datos para identificar valores únicos y asignar las relaciones correctas basadas en claves primarias y foráneas, asegurándose de cumplir con la integridad referencial del modelo.

Por último, en la fase de carga, se insertan los datos correctamente organizados en las tablas de la base SQLite, respetando las dependencias y relaciones del esquema. En la Figura 14, una de las funciones ETL desarrolladas.

```python
def etl_victima_tablas(excel, db_path):
    conexion = sqlite3.connect(db_path)
    cursor = conexion.cursor()

    cursor.execute("PRAGMA foreign_keys = ON")

    df = pd.read_excel(excel)

    tablas = ['Origen', 'TipoAhogamiento', 'Pronostico', 'Reanimacion',
              'Extraccion', 'Causa', 'FactorRiesgo', 'Antecedente', 'MaterialRescate', 'PrimerInterviniente']


    for nombre_tabla in tablas:
        unique_values = df[nombre_tabla].dropna().unique()
        cod_value = 1

        for value in unique_values:
            cursor.execute(f"SELECT COUNT(*) FROM {nombre_tabla} WHERE nombre{nombre_tabla} = ?", (value,))

            if cursor.fetchone()[0] == 0:
                cursor.execute(f"INSERT INTO {nombre_tabla} (cod{nombre_tabla}, nombre{nombre_tabla}) VALUES (?, ?)", (cod_value, value))
                cod_value += 1

    conexion.commit()
    conexion.close()

    print("ETL tablas auxiliares victima completado.")
```

Figura 14 – Función del ETL para generar las columnas auxiliares de Victima

6.3 Selección y prueba de la librería de visualización de datos

En un inicio, se evaluaron diversas opciones ampliamente reconocidas en el ecosistema Python: Matplotlib, Plotly y Bokeh. El objetivo principal era encontrar una herramienta que ofreciera no solo la representación clara de los datos, sino también un alto grado de interactividad para facilitar la exploración y comprensión de la información por parte de los usuarios.

Finalmente, tras varias pruebas piloto realizadas en una página en blanco —ya que la interfaz definitiva de la aplicación aún no estaba implementada— se optó por Bokeh. Esta librería destacó por su facilidad para integrar componentes interactivos (como zoom, selección de rangos, herramientas de Hover, etc.) que se encuentran a la derecha de la gráfica como se observa en la Figura 15.

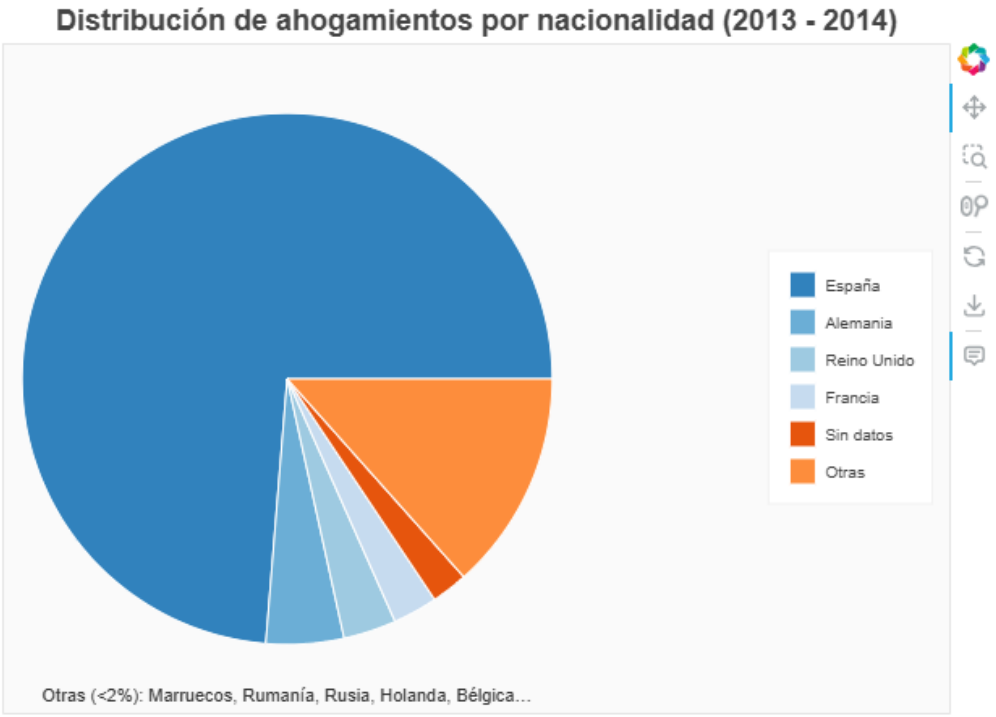


Figura 15 – Gráfica circular generada con Bokeh, con una barra de herramientas Hover situada a la derecha

6.4 Creación del proyecto en Django

Tras la preparación y estructuración de los datos, el siguiente paso fue la creación del entorno y la arquitectura base de la aplicación utilizando el *framework* Django. Se optó por Django por su robustez, escalabilidad y por integrar de forma nativa un completo sistema de administración, lo que facilitó enormemente tanto el desarrollo inicial como la futura gestión de los datos.

En primer lugar, se generó el nuevo proyecto Django denominado ahogamientos, lo que proporcionó la estructura básica de la aplicación, junto con los archivos de configuración inicial necesarios para su correcto funcionamiento.

Dado que la aplicación diseñada cuenta con una única página visual centralizada, se creó solamente una aplicación interna o app llamada home. Esta organización modular sencilla permite mantener el código ordenado y fácilmente gestionable, concentrando toda la lógica y vistas dentro de una sola app. Aunque el proyecto tiene un alcance limitado a una interfaz principal, mantener la separación en una app facilita futuras ampliaciones o modificaciones.

A continuación, se configuró la conexión con la base de datos SQLite previamente construida durante el proceso ETL, indicándolo en el archivo de configuración settings.py. Además, se definieron los modelos de datos, en models.py, usando el comando automático inspectdb.

Gracias a la definición explícita de los modelos, Django pudo generar automáticamente un sistema de migraciones que permitió sincronizar la estructura de la base de datos y facilitar su modificación futura sin pérdida de información. Por último, se configuró el panel de administración (admin.py), registrando los modelos principales y añadiendo personalizaciones básicas para permitir una gestión eficiente y visual de todos los registros desde la aplicación.

6.5 Aspecto visual de la página

Dado que la aplicación cuenta con una única página visual centralizada, el enfoque se centró en crear una interfaz clara, intuitiva y funcional que permitiera a los usuarios interactuar con los datos de manera eficiente.

La construcción del *frontend* se realizó utilizando tecnologías web estándar, principalmente HTML, CSS y JavaScript, integrados dentro del sistema de plantillas propio de Django (HTML en la carpeta templates y CSS y JavaScript en static/styles). El aspecto de la aplicación se muestra en la Figura 24; ejemplos de código en las Figuras 5, 6 y 7.

6.6 Desarrollo del *Backend*

Tras definir el aspecto visual de la página, se procedió a construir el *backend* de la aplicación en Django, encargándose de gestionar la lógica de negocio, el acceso a los datos y la comunicación con el *frontend*.

El *backend* de este proyecto está enteramente gestionado en el módulo views.py, que se encarga principalmente de proveer diferentes *endpoints* que reciben peticiones HTTP y devuelven respuestas JSON con datos o componentes HTML.

Estas son las diferentes funcionalidades que tiene esta capa:

- Renderiza la plantilla principal home.html que contiene la base visual de la aplicación a través de la función home. No realiza operaciones complejas y sirve como punto de partida.

- Genera las gráficas usando Bokeh (generar_grafica_apilada, generar_grafica_circular, generar_grafica_lineas, generar_diagrama_dispersion, generar_mapa, generar_histograma, generar_radar) siguen esta pauta:

    - Reciben parámetros (filtros de año, sexo, mortalidad, ubicaciones, etc.) mediante request.POST, como en la Figura 16.
    - Filtran y agrupan los datos usando ORM de Django (Victima.objects.filter(...)) para extraer un QuerySet.
    - Transforman este QuerySet a un DataFrame de pandas para una manipulación más cómoda y flexible.
    - Crean los gráficos con Bokeh, generando las figuras, rellenos, leyendas, herramientas interactivas y configuraciones de visualización que mejoran la presentación.
    - Devuelven al *frontend* el script HTML, los recursos CSS y JS necesarios empaquetados en un JSON con llave 'grafica_html'.

```python
@csrf_exempt
def generar_diagrama_dispersion(request):
    # Obtiene parámetros del POST
    color_sexo = request.POST.get('color_sexo', 'false') == 'true'
    solo_mortales = request.POST.get('solo_mortales', 'false') == 'true'
    mostrar_linea = request.POST.get('mostrar_linea', 'false') == 'true'

    # Construye filtro base
    filtro = {}

    # Sólo mostrar ahogamientos mortales en caso de que se active el filtro
    if solo_mortales:
        filtro['pronostico__nombrepronostico'] = "Ahogamiento mortal"

    # Consulta con los filtros elegidos
    qs = (
        Victima.objects
        .filter(**filtro)
        .exclude(edad=None)
        .values('edad', 'sexo')
        .annotate(Ahogamientos=Count('codvictima'))
        .order_by('edad')
    )
```

Figura 16 – Extracto de código de la función generar_diagrama_dispersion de views.py

- Toda la consulta tiene como base el modelo Victima y sus relaciones (a través de claves foráneas) permiten acceder al resto de datos.

- Visualización de mapas e interactividad geoespacial con la función generar_mapa. Se usa la librería GeoPandas para leer los *shapefiles* de provincias, se unen con los datos estadísticos y se crea la visualización con paletas de color para mostrar cantidades de casos en cada provincia, usando objetos de Bokeh como GeoJSONDataSource, ColorBar y HoverTool.

- En la función comparativa_sklearn, la librería Scikit-Learn se emplea para modelar y analizar la probabilidad de mortalidad en un incidente, dadas ciertas características de entrada.

- Manejo de puntos calientes mediante *clustering* espacial con la función específica de *hotspots*. Se utiliza la librería Scikit-Learn permitiendo identificar de manera automática aquellas zonas geográficas donde se concentran mayores cantidades de casos.

Estos dos últimos apartados serán abordados en mayor medida en a continuación.

6.6.1 Integración de algoritmos de *machine learning* en el *backend*

La librería Scikit-Learn ha sido fundamental en la capa de *backend* para dotar a la aplicación de capacidades avanzadas de análisis de datos, aprendizaje automático y visualización explicativa. Sus principales usos en el proyecto se describen a continuación:

- **Detección de hotspots mediante *clustering* espacial**. Uno de los usos más destacados de Scikit-Learn es la identificación automática de zonas calientes o *hotspots* de riesgo mediante el algoritmo de *clustering* DBSCAN. Aprovechando los datos de localización (latitud y longitud) de los incidentes, se normalizan primero las coordenadas para evitar sesgos por escala utilizando StandardScaler y, a continuación, se aplica DBSCAN para agrupar aquellos casos que se encuentran espacialmente próximos.

  Este análisis permite descubrir áreas con alta concentración de incidentes sin depender de nombres de lugares o errores humanos en la introducción de datos.

- **Modelado predictivo de la mortalidad con Random Forest**. Mediante la función específica de comparación basada en Scikit-Learn (comparativa_sklearn), se emplea el algoritmo RandomForestClassifier para construir un modelo capaz de predecir la probabilidad de mortalidad en un incidente a partir de características seleccionadas (actividad, localización, zona vigilada, factor de riesgo, intervención y edad).

- **Reducción dimensional y visualización con PCA**. Para facilitar la interpretación visual y la comparación entre incidentes, se aplica la técnica de Análisis de Componentes Principales (PCA) de Scikit-Learn. Esta transformación proyecta tanto los registros históricos como los nuevos casos introducidos a un plano bidimensional. Así, se puede mostrar gráficamente la relación del caso consultado respecto al histórico, permitiendo identificar similitudes y patrones.

- **Codificación y transformación de datos**. La limpieza y codificación de variables categóricas y numéricas mediante herramientas como OneHotEncoder y StandardScaler posibilitan tanto el entrenamiento de los modelos predictivos como los análisis espaciales y la preparación de datos para visualización.

- **Análisis de estacionalidad mediante K-Means**. Utilizando Scikit-Learn, se ha implementado un análisis de agrupamiento mensual de los ahogamientos por medio del algoritmo KMeans. De cara a identificar patrones estacionales, se calculan las medianas del número de ahogamientos por mes en el rango de años seleccionado, y estas se agrupan en *clusters* que comparten similar incidencia.

  El proceso comienza normalizando los datos mensuales y ejecutando KMeans para segmentar los meses según la concentración relativa de incidentes. El resultado visual permite diferenciar claramente grupos de meses con conductas homogéneas, facilitando la detección de periodos críticos.

  En el análisis, se ha seleccionado un total de 3 clusters empleando la técnica del codo, como se muestra en la Figura 17. Esta técnica consiste en representar la inercia frente al número de clusters y elegir el valor de k en el que la disminución de la inercia empieza a ser menos significativa, identificando el punto de inflexión o “codo” de la curva. Así, se consigue un equilibrio entre segmentación y simplicidad en la agrupación de los datos.

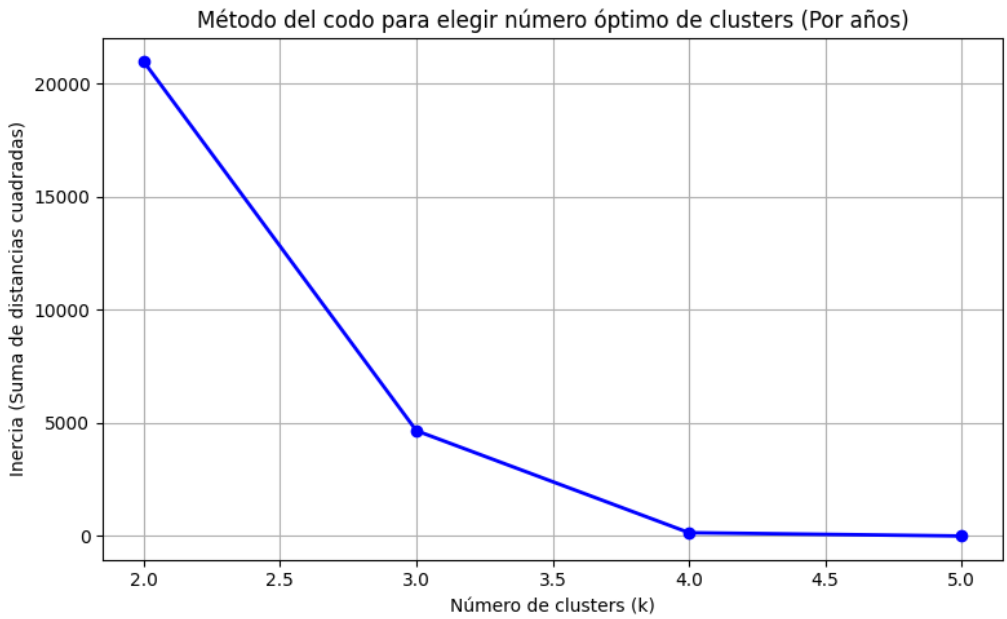


Figura 17 – Gráfica del método del codo

- **Predicción de mortalidad con árboles de decisión**. En el proyecto se ha implementado un modelo de árbol de decisión utilizando la clase DecisionTreeClassifier de Scikit-Learn para analizar el impacto combinado de entre dos y tres variables seleccionadas por el usuario sobre el pronóstico de mortalidad en incidentes de ahogamiento. El proceso comienza con la selección de variables pertinentes (por ejemplo, edad, tipo de localización, intervención, condiciones ambientales), que son binarizadas según condiciones clínicas o contextuales específicas. Los datos se preparan para ofrecer al árbol información relevante y convertible en reglas interpretables.

  El modelo se entrena ajustando su profundidad al número exacto de variables seleccionadas, lo que asegura interpretabilidad y evita el sobreajuste. Tras el entrenamiento, se genera y visualiza el árbol de decisión resultante como una imagen interactiva embebida con ayuda de Plot_Tree. Cada nodo y rama del árbol representa una combinación de las variables escogidas y conduce, a través de reglas lógicas claras, hasta una predicción final: mortalidad o no mortalidad.

En conjunto, los diferentes usos de la librería Scikit-Learn han permitido dotar al *backend* de funcionalidades de inteligencia artificial, análisis avanzado y soporte explicativo, incrementando notablemente el valor analítico y visual de la aplicación, y facilitando tanto la exploración interactiva como la prevención fundamentada de los riesgos asociados a los incidentes de ahogamiento.

7 Pruebas

Las pruebas son una fase fundamental en el desarrollo de cualquier aplicación, ya que permiten verificar que todas las funcionalidades implementadas cumplen con los requisitos establecidos y que el sistema se comporta de manera correcta y fiable ante diferentes escenarios de uso.

En el caso de esta aplicación, cuyo núcleo es la visualización dinámica de datos a través de gráficas interactivas generadas a partir de filtros y análisis estadísticos, las pruebas adquieren una especial relevancia para garantizar que cada gráfica refleja fielmente la información subyacente y que las interacciones del usuario producen resultados esperados y consistentes.

7.1 Pruebas unitarias

En el proyecto se realizaron pruebas para verificar el correcto funcionamiento de cada función por separado. Estas pruebas consistieron en realizar peticiones con diversos parámetros y comprobar que las respuestas JSON mostraban mensajes de error adecuados cuando los filtros eran incorrectos o insuficientes, como se aprecia en la Figura 18.

Estas pruebas permitieron evaluar individualmente las funciones principales, comprobando tanto respuestas exitosas con datos válidos como la gestión adecuada de errores ante entradas incorrectas o incompletas. Aunque no se emplearon herramientas automatizadas para las pruebas, este método fue suficiente para asegurar la calidad y fiabilidad del código durante el desarrollo.

Figura 18 – Error tras tratar de generar una gráfica ningún valor seleccionado

7.2 Pruebas funcionales de la generación de gráficas

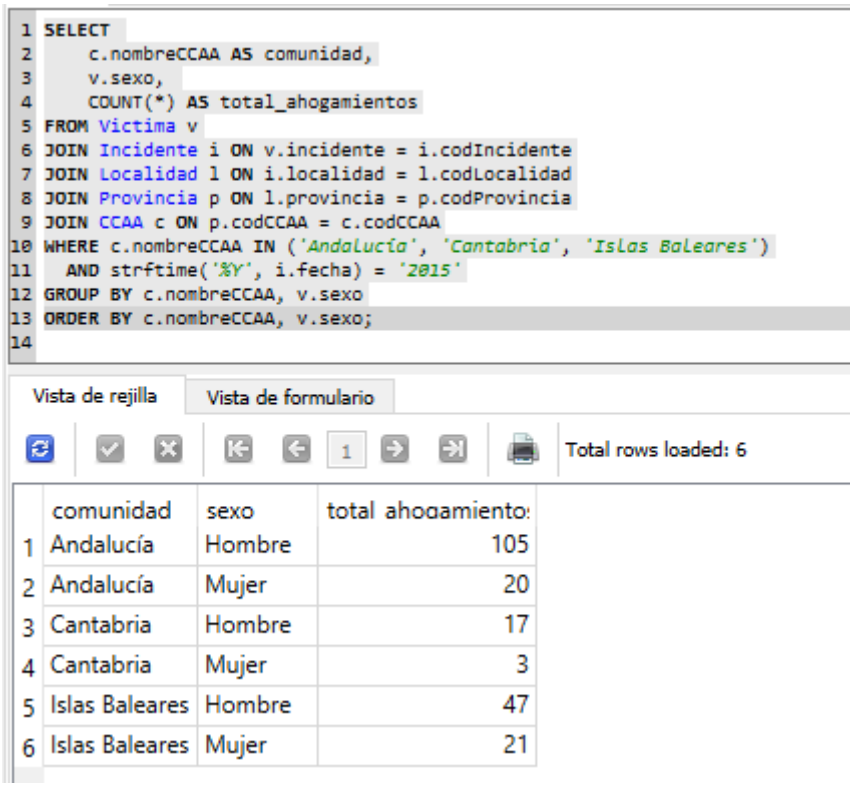
Esta fase consiste en evaluar que, al aplicar distintos filtros y parámetros, las visualizaciones resultantes reflejen correctamente los datos subyacentes, permitiendo que el usuario interprete adecuadamente los resultados y tome decisiones informadas.

Para ello, se han definido casos de uso que van desde escenarios básicos, donde se aplican filtros simples y controlados con datos conocidos, hasta situaciones más complejas que combinan múltiples filtros y variables, simulando condiciones reales y variadas de uso.

Estas pruebas se han realizado comparando los resultados que ofrecía la aplicación con consultas SQL realizadas en la base de datos.

En el caso que se expone a continuación, se seleccionaron como comunidades autónomas Andalucía, Cantabria e Islas Baleares. Se selecciono el año 2015 y el filtro que divide los resultados según el sexo.

Los resultados obtenidos muestran que en Andalucía se registraron 105 ahogamientos de hombres y 20 de mujeres; en Cantabria, 17 de hombres y 3 de mujeres; mientras que, en Islas Baleares, el total fue de 47 hombres y 21 mujeres. Se pueden observar los resultados en las Figuras 19 y 20.



```
SELECT
    c.nombreCCAA AS comunidad,
    v.sexo,
    COUNT(*) AS total_ahogamientos
FROM Victima v
JOIN Incidente i ON v.incidente = i.codIncidente
JOIN Localidad l ON i.localidad = l.codLocalidad
JOIN Provincia p ON l.provincia = p.codProvincia
JOIN CCAA c ON p.codCCAA = c.codCCAA
WHERE c.nombreCCAA IN ('Andalucía', 'Cantabria', 'Islas Baleares')
  AND strftime('%Y', i.fecha) = '2015'
GROUP BY c.nombreCCAA, v.sexo
ORDER BY c.nombreCCAA, v.sexo;
```

Vista de rejilla | Vista de formulario

Total rows loaded: 6

	comunidad	sexo	total ahogamiento:
1	Andalucía	Hombre	105
2	Andalucía	Mujer	20
3	Cantabria	Hombre	17
4	Cantabria	Mujer	3
5	Islas Baleares	Hombre	47
6	Islas Baleares	Mujer	21

Figura 19 – Consulta en la base de datos usando SQLiteStudio

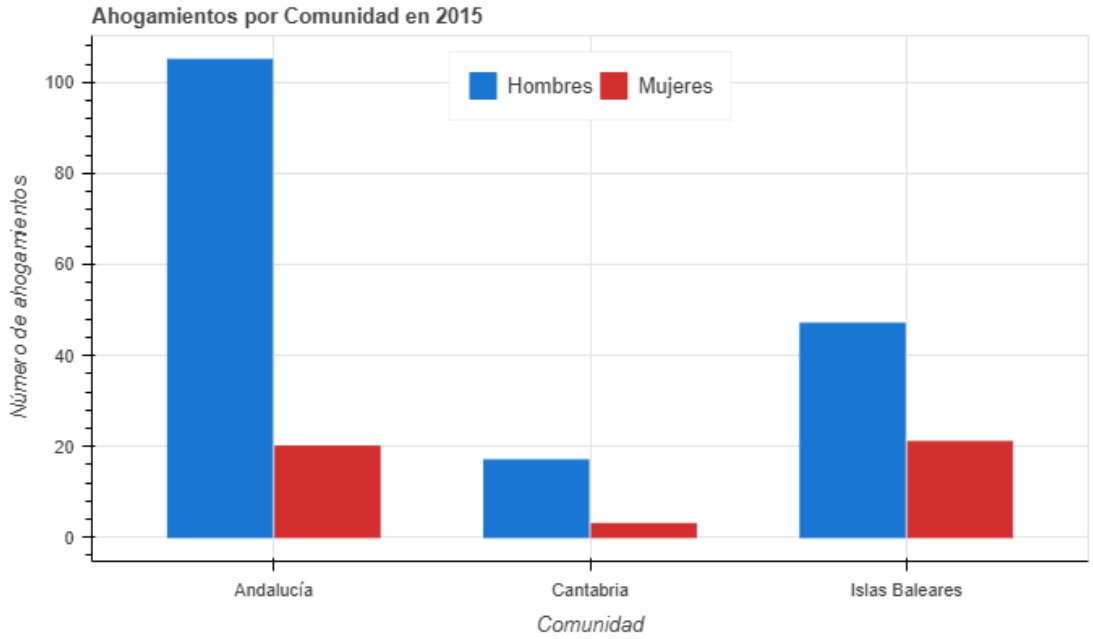


Figura 20 – Gráfica con los filtros elegidos

Al comparar estos datos con los valores representados en la gráfica, se observa que coinciden exactamente. No hay ninguna discrepancia numérica, lo que indica que la consulta SQL y el proceso de transformación hacia la visualización se han realizado de manera correcta.

7.3 Pruebas de interacción y usabilidad

Esta fase se centra en validar que los componentes de la interfaz, especialmente los controles de filtrado y las visualizaciones gráficas, respondan correctamente a las acciones del usuario y reflejen los cambios de manera inmediata y coherente.

Es importante verificar que, al modificar los filtros, los gráficos se actualicen dinámicamente sin necesidad de recargar la página, que los elementos visuales como los *pills* se añadan y eliminen de forma adecuada, y que los selectores se activen o desactiven conforme a la lógica de negocio diseñada para evitar selecciones incompatibles o erróneas, como la prohibición de seleccionar comunidades autónomas y provincias en la misma consulta.

También se ha verificado el correcto funcionamiento del botón “Limpiar filtros” en cada uno de los diferentes formularios.

7.4 Pruebas de rendimiento

En el contexto de tu aplicación, las pruebas de rendimiento orientadas a la velocidad de respuesta adquieren una importancia crítica, ya que la experiencia del usuario depende directamente del tiempo que transcurre desde que este ejecuta una consulta hasta que obtiene la visualización solicitada, como se indica en la Tabla 9. Si el sistema tarda demasiado en generar y mostrar una gráfica o un mapa, el usuario percibirá lentitud, perderá interés y, en algunos casos, abandonará la interacción antes de obtener el resultado.

Se han realizado pruebas con diferentes tipos de filtro en cada una de las consultas, obteniendo resultados satisfactorios, como se observa en la Figura 21.

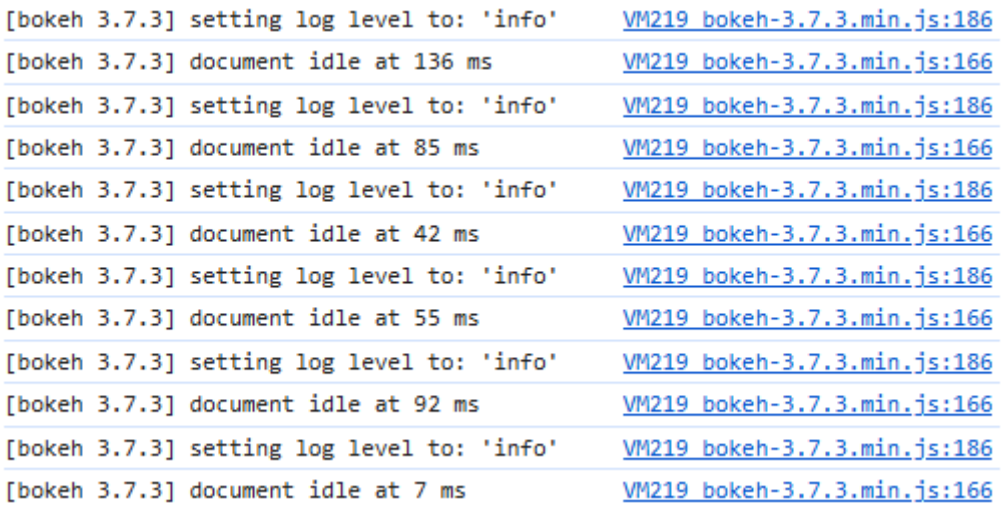


Figura 21 – Estadísticas de generación de gráficas conseguidas a partir de las herramientas de desarrollo del navegador

7.5 Pruebas de inserción de nuevos datos (administrador)

En este caso, se han realizado pruebas específicas de alta de registros por parte del administrador para verificar el correcto funcionamiento de la funcionalidad de inserción de nuevos datos en la base de datos. Para ello, se procedió a introducir registros de prueba tanto en la tabla de incidentes como en la de víctimas, simulando el flujo completo de creación de datos tal y como lo realizaría un usuario con privilegios administrativos.

Estas inserciones se realizaron con el objetivo de comprobar que el sistema aceptaba la información correctamente, que se aplicaban las validaciones necesarias y que los nuevos registros quedaban inmediatamente disponibles para su consulta y visualización en las distintas gráficas y filtros de la aplicación. Se puede observar un ejemplo en las Figuras 22 y 23.

Una vez completada la verificación y confirmada la correcta operativa, las entradas generadas para la prueba fueron eliminadas para mantener la integridad y limpieza de la base de datos, evitando que los datos ficticios interfirieran en los resultados reales del sistema.

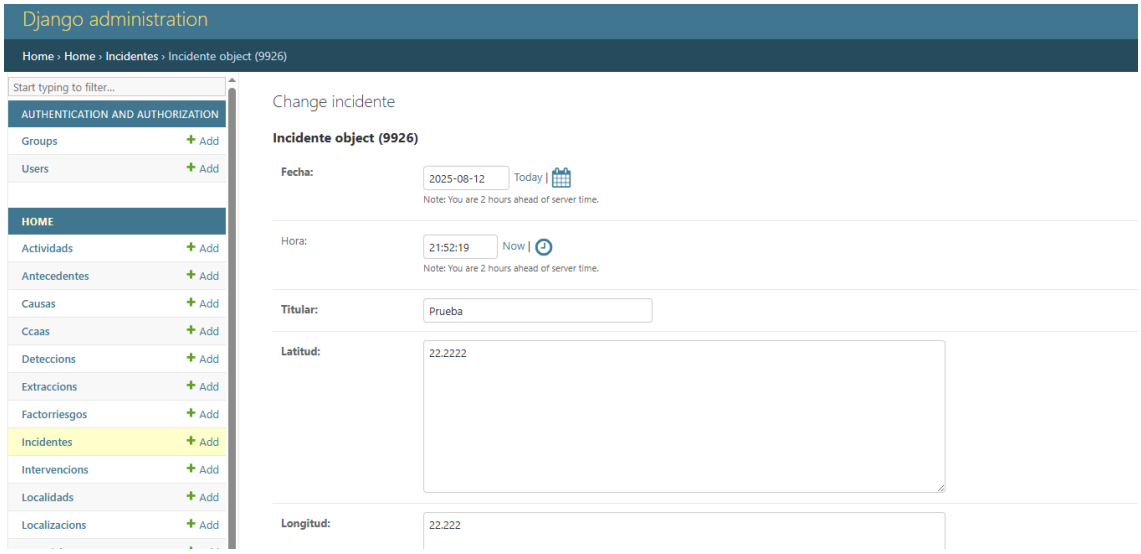


Figura 22 - Inserción de un incidente de prueba usando Django Administration

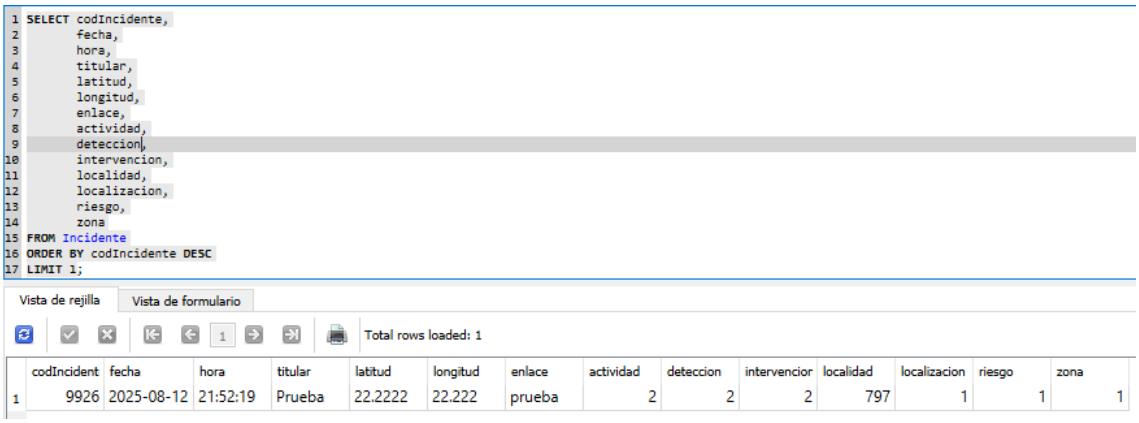


Figura 23 – El incidente de prueba aparece en la base de datos

7.6 Pruebas de aceptación

Las pruebas de aceptación de tu TFG se han realizado presentando la aplicación al "cliente", en este caso Luis Miguel, mostrando todas las funcionalidades implementadas como la generación de gráficas, diagramas, mapas y consultas avanzadas, y la nueva herramienta de administración de la base de datos. Durante dicha evaluación, Luis Miguel realizó observaciones sobre posibles mejoras y futuros desarrollos, las cuales quedaron registradas en el apartado 10 de "Trabajo futuro".

El cliente mostró especial interés en las consultas que emplean *machine learning* y, en particular, la funcionalidad de detección de puntos calientes, mostrando interés en su aplicación para un informe que desea realizare proximamente, añadiendo capturas sobre los *hotspots* de cada comunidad autónoma.

La valoración general por parte del Luis Miguel fue muy satisfactoria, dando por más que correcto el resultado de la aplicación.

8 Resultado final

El aspecto inicial diseñado para el proyecto presenta un enfoque visual claro y sencillo, con un esquema de colores dominante en tonos suaves de azul, como se observa en la Figura 24. El encabezado superior resalta el título “Prevención de Ahogamientos” en un color blanco sobre un fondo azul más oscuro, aportando un contraste que facilita la lectura y establece la identidad del proyecto.

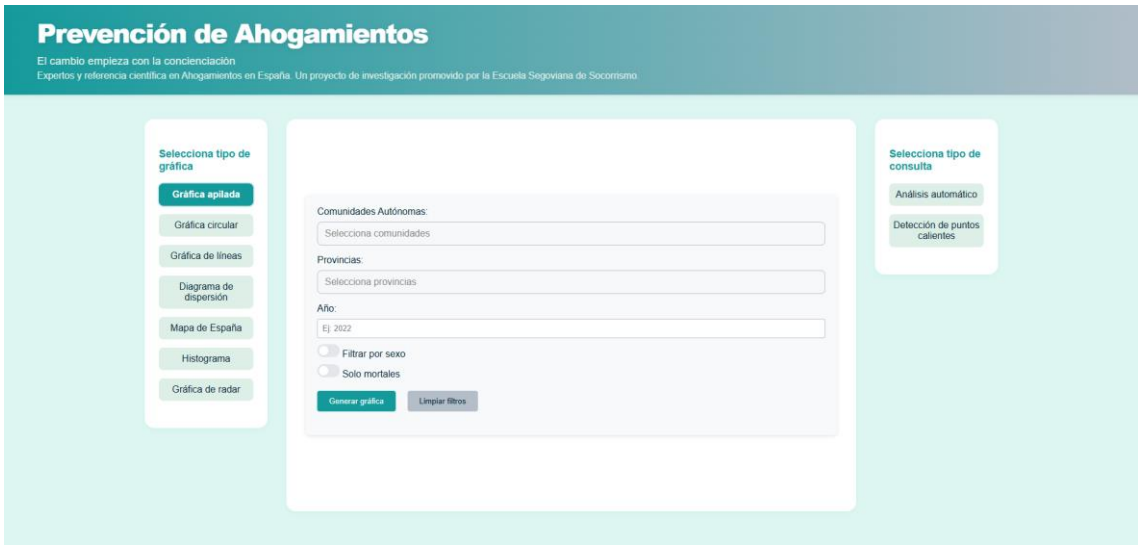


Figura 24 – Visión inicial de la aplicación

La aplicación dispone de una única página principal, organizada en tres columnas claramente diferenciadas. En la columna izquierda, se sitúa un menú vertical con botones para seleccionar el tipo de gráfica que el usuario desea visualizar, como se puede observar en la Tabla 1. Cada una de estas gráficas está siendo generada por la librería Bokeh. Por defecto, la gráfica que esta seleccionada es la apilada.

La columna central, de mayor tamaño, actúa como el área principal de interacción. Aquí el usuario puede aplicar distintos filtros para configurar la visualización de los datos, como la selección de comunidades autónomas y provincias mediante menús desplegables, un campo de texto para introducir el año, y opciones adicionales de filtro sobre sexo y mortalidad, mediante interruptores de activación. Estos filtros cambian en función del tipo de gráfica que se desea utilizar, pero en todos los casos aparecen vacíos o desactivados de manera predeterminada. Además, incluye botones para “Generar gráfica” y “Limpiar filtros”, garantizando un control intuitivo y eficiente de la consulta.

Las selecciones de categorías que requieren elegir entre un listado de opciones (nacionalidades, comunidades autónomas, provincias...) se realizan mediante componentes de tipo *multi-select*. Cada opción elegida se muestra de forma inmediata como una etiqueta visual que contiene el nombre de la selección y un

botón para eliminarla individualmente, como indica la Tabla 3. Así, el usuario puede ver claramente todas las opciones activas y modificar los filtros eliminando cualquier selección sin necesidad de desplegar el listado completo, como se aprecia en la Figura 25.

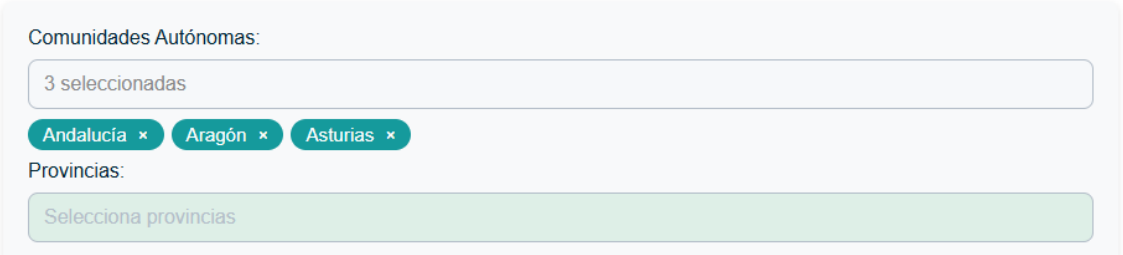


Figura 25 – Selección de varias comunidades autónomas

En el caso concreto de la gráfica apilada, se ha utilizado JavaScript para bloquear de forma bidireccional los selectores de comunidades autónomas y provincias. Esto significa que cuando el usuario selecciona alguna comunidad autónoma, el selector de provincias queda bloqueado y no permite hacer ninguna selección, y viceversa. Esta medida se implementa para evitar la mezcla de selecciones entre ambos filtros, asegurando que no se combinen datos de comunidades y provincias simultáneamente, lo que podría generar inconsistencias en los resultados y en la gráfica generada.

A la derecha, se encuentra una columna que alberga opciones para elegir entre diferentes tipos de consultas analíticas, las cuales utilizan la librería Scikit-Learn. Esta librería es una de las principales herramientas de aprendizaje automático en Python, reconocida por su facilidad de uso, amplia documentación y riqueza de algoritmos disponibles para tareas como clasificación, regresión o agrupamiento.

Gracias a Scikit-Learn, la aplicación puede procesar grandes volúmenes de datos para detectar patrones complejos, extraer relaciones ocultas y generar resultados avanzados directamente accesibles para el usuario desde la aplicación. Utilizando esta tecnología se ha diseñado un predictor de probabilidad de ahogamiento mortal según variables, un detector de puntos calientes sobre el mapa de España, como se aprecia en la Figura 26, un separador de grupos usando *clustering* y un generador de árboles de decisión que predice la mortalidad de una serie de variables entrelazadas.

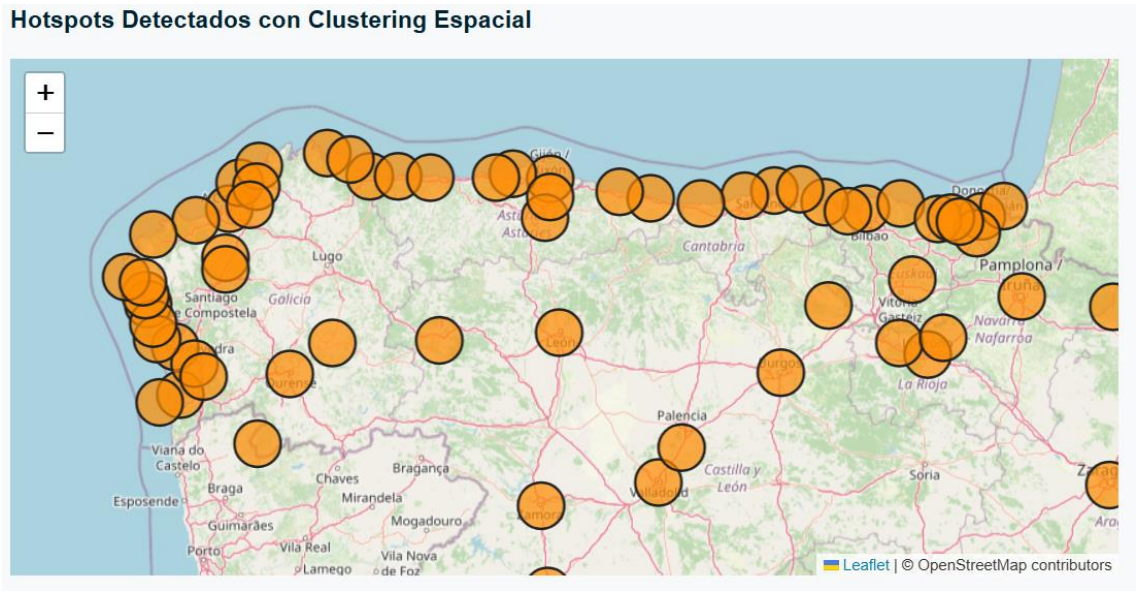


Figura 26 – Mapa de zonas calientes entre los años 2015 y 2017

Tanto la columna de la izquierda, como la columna de la derecha, están conectadas mediante JavaScript para gestionar la interacción del usuario de forma dinámica. Cuando el usuario selecciona una opción en cualquiera de estas dos columnas, esa opción queda visualmente destacada con un color azul más oscuro, indicando que es la selección activa. Al mismo tiempo, cualquier otra opción previamente seleccionada en la misma columna se desactiva y vuelve a su estado original, asegurando que solo una única opción esté marcada en cada menú en todo momento.

Una vez que el usuario ha seleccionado el tipo de gráfica que desea visualizar y ha configurado los filtros correspondientes según sus necesidades, como se muestra en la Tabla 2, debe pulsar el botón “Generar gráfica” para que el sistema ejecute la consulta solicitada. El sistema valida que los filtros han sido correctamente rellenados y procesados para garantizar la exactitud y relevancia de los datos mostrados. Si todo está en orden, la gráfica o el mapa solicitados aparecerán dinámicamente justo debajo del panel donde se encuentran los filtros, como se indica en la Tabla 4. Esta disposición de la aplicación tras generar la gráfica se muestra en la Figura 27.

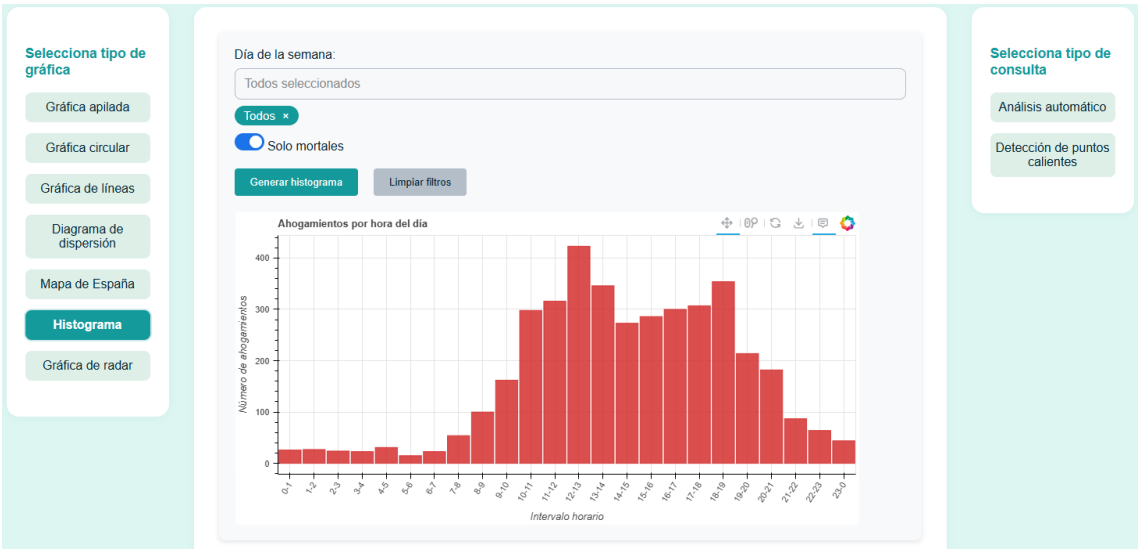


Figura 27 – Ejemplo sobre la generación de gráficas

Estas visualizaciones son interactivas, detallando información cuando colocas el cursor encima. También cuenta con la posibilidad de ampliar o reducir el zoom. Para mayor comodidad, la plataforma ofrece la opción de descargar la imagen generada en formato .png, como se muestra en la Tabla 5, ya sea para incluirla en informes, presentaciones o para un análisis más profundo fuera de la aplicación.

El otro objetivo del proyecto ha sido habilitar una opción para añadir, editar o eliminar registros en la base de datos, como se indica en las Tablas 6, 7 y 8, respectivamente. Para ello, se usó la función de administrador de Django, que proporciona una interfaz aplicación completa, robusta y fácil de utilizar para gestionar los datos de tu aplicación, como aparece en la Figura 28. Esta funcionalidad viene integrada en Django y permite realizar todas las operaciones básicas (crear, consultar, modificar y eliminar registros) sobre cualquiera de los modelos registrados con solo unos pocos pasos de configuración, sin necesidad de desarrollar una interfaz propia desde cero.

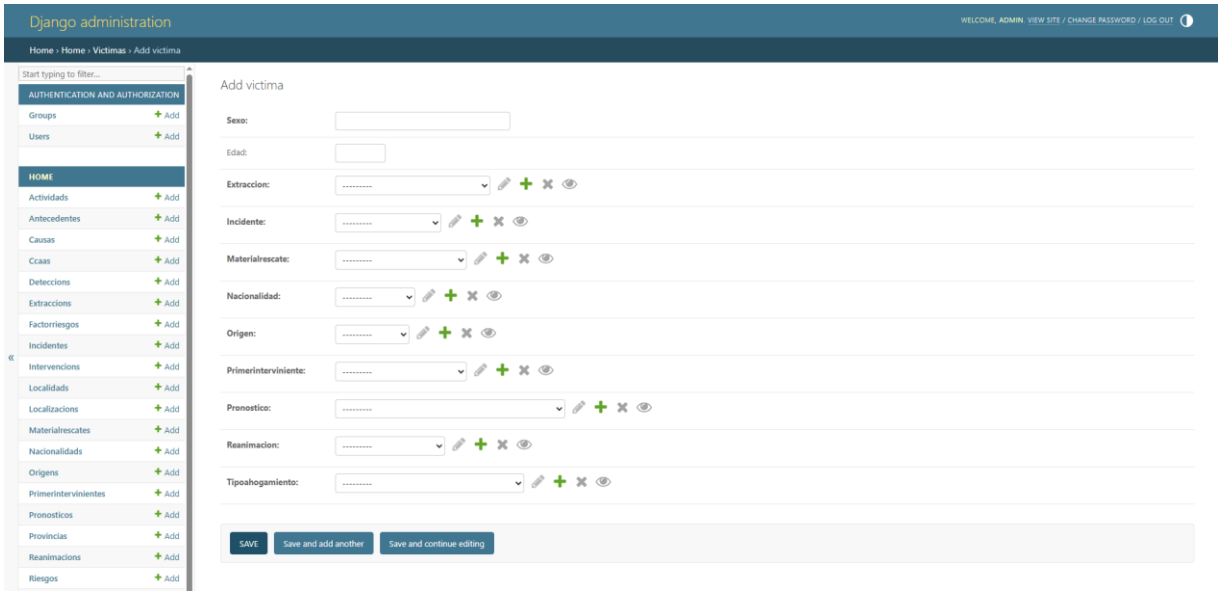


Figura 28 – Interfaz para añadir nuevas víctimas

Para poder acceder a esta funcionalidad de Django, debemos introducir "http://localhost:8000/admin" en nuestro buscador, donde se nos pedirá introducir el nombre de usuario y su contraseña. Este primer usuario ha debido ser creado previamente a través de la terminal utilizando "python manage.py createsuperuser" y seguir las instrucciones.

Una vez iniciada la sesión, como se indica en la Tabla 10, el administrador accede a un panel centralizado desde el cual puede gestionar de manera eficaz toda la información relevante de la aplicación. Esta interfaz, segura y solo accesible para usuarios autorizados, permite administrar los diferentes modelos de datos, como incidentes, víctimas, actividades, localidades o nacionalidades, entre otros. Principalmente, esta ortientada a la incorporación de nuevos datos a las tablas "Incidente" y "Victima", ya que el resto de tablas tienen valores definidos o que rara vez van a requerir un cambio.

Por otra parte, el panel de "Django administration" facilita la edición en línea de objetos relacionados (por ejemplo, añadir o modificar víctimas directamente desde el incidente correspondiente), así como la aplicación de acciones masivas sobre conjuntos de registros seleccionados.

En Django, además de la potente interfaz de administración de datos que ofrece el panel de administración, también puedes gestionar usuarios, grupos y permisos de forma sencilla y flexible. Esto te permite controlar con precisión quién puede acceder a cada parte de tu aplicación y qué acciones puede realizar.

9 Conclusiones

El desarrollo de esta aplicación representa un proyecto integral y bien logrado que ha cumplido con los objetivos planteados inicialmente, enfocados en crear una plataforma interactiva y accesible para el análisis y prevención de ahogamientos en España.

A lo largo del trabajo se ha logrado construir un sistema robusto que integra la visualización dinámica de datos, filtrado avanzado, y la incorporación de técnicas de *machine learning* para el análisis predictivo y la detección de patrones espaciales, lo que proporciona una herramienta valiosa tanto para usuarios generales como para expertos en prevención.

Durante el desarrollo, se han adquirido y aplicado conocimientos técnicos profundos en diversas áreas fundamentales. Se ha reforzado el dominio de Python y sus librerías científicas, como pandas para manejo y análisis de datos, y Scikit-Learn para la implementación de algoritmos de *machine learning* (KMeans, Random Forest, Decision Trees, PCA, DBSCAN).

El proyecto también implicó un aprendizaje significativo en el uso de OpenRefine para la limpieza y normalización efectiva de datos, lo cual es clave para garantizar la calidad de la información procesada.

Por otra parte, se ha adquirido experiencia relevante en el desarrollo de la aplicación mediante el *framework* Django, que permitió estructurar adecuadamente el *backend*, gestionar la base de datos con SQLite y facilitar la comunicación eficiente con el *frontend*. El apartado visual y de experiencia de usuario se enriqueció a través del uso combinado de HTML, CSS y JavaScript, incluyendo el manejo de la librería Bokeh, que aportó funcionalidad avanzada para la generación de gráficas y mapas interactivos.

Esta aplicación es especialmente importante porque aborda un problema de creciente relevancia social. La tendencia observada en el verano de 2025, con incrementos significativos en los incidentes, subraya la necesidad de herramientas tecnológicas que faciliten una mejor comprensión y análisis de estos datos, promoviendo la conciencia y acciones preventivas.

10 Trabajo futuro

En un Trabajo de Fin de Grado, evidentemente no es posible implementar todas las funciones nuevas o mejoras que uno desearía debido a limitaciones de tiempo, recursos y alcance del proyecto. A continuación, se presentan algunas de las posibles mejoras y nuevas funcionalidades que podrían incorporarse en futuras versiones de la aplicación para potenciar su utilidad y alcance.

- **Implementar un sistema de gestión de usuarios que permita almacenar consultas y gráficas personalizadas**. Esta funcionalidad consistiría en integrar un módulo de autenticación y perfiles que habilite a cada usuario registrado la posibilidad de guardar sus filtros, parámetros de búsqueda y visualizaciones generadas directamente en la aplicación. De este modo, se podría acceder posteriormente a dichas consultas o gráficas desde cualquier dispositivo y en cualquier momento, evitando la necesidad de exportarlas o descargarlas localmente.

- **Ampliación de visualizaciones, personalización y capacidades analíticas**. De cara a futuras versiones, la plataforma podría enriquecerse mediante la incorporación de nuevas visualizaciones más inmersivas y dinámicas, que vayan más allá de las clásicas representaciones de barras, líneas o sectores. Esto incluiría gráficos interactivos en 3D, mapas de calor geoespaciales avanzados o visualizaciones narrativas generadas por inteligencia artificial que expliquen automáticamente tendencias y anomalías.

- **Automatización del proceso de añadir nuevos datos a la base de datos**. Actualmente, el registro de nuevos casos de ahogamiento en la aplicación se realiza de forma manual. Este proceso consiste en recibir notificaciones a través de correos electrónicos que informan sobre incidentes recientes reportados en noticias u otros canales de comunicación. A partir de estas notificaciones, un administrador o responsable ingresa los datos correspondientes en el sistema, lo que puede implicar un esfuerzo considerable y un margen de tiempo antes de que la información esté disponible para su análisis y visualización.

  Automatizar esta tarea permitiría captar y almacenar datos de forma inmediata y directa, reduciendo la dependencia del procesamiento humano y minimizando la latencia entre la ocurrencia del incidente y su inclusión en la base de datos.

- **Nueva interfaz de mantenimiento de la base de datos**. A pesar de que en la aplicación se esta usando Django Administration para gestionar la base de datos, el desarrollo de una interfaz específica y más adaptada a las necesidades y estética de la plataforma proporcionaría una experiencia más intuitiva y directa para los administradores. Esta nueva interfaz podría integrarse dentro del propio panel principal de la aplicación y permitir una nueva función para volcar volúmenes grandes de datos.

11 Referencias

[1] AHOGAMIENTOS.COM, https://ahogamientos.blogspot.com/
[Ultimo acceso: 31/08/2025]

[2] Visual Studio Code, https://code.visualstudio.com/
[Ultimo acceso: 25/08/2025]

[3] Django, https://www.djangoproject.com/
[Ultimo acceso: 25/08/2025]

[4] Python, https://www.python.org/
[Ultimo acceso: 24/08/2025]

[5] Git, https://git-scm.com/
[Ultimo acceso: 30/08/2025]

[6] SQLite, https://www.sqlite.org/index.html
[Ultimo acceso: 25/08/2025]

[7] Scikit-Learn, https://scikit-learn.org/
[Ultimo acceso: 25/08/2025]

[8] Bokeh, https://docs.bokeh.org/en/latest/
[Ultimo acceso: 17/08/2025]

[9] OpenRefine, https://openrefine.org/
[Ultimo acceso: 29/03/2025]

[10] MagicDraw, https://www.nomagic.com/products/magicdraw
[Ultimo acceso: 31/08/2025]

[11] Instituto Geográfico Nacional, https://www.ign.es/
[Ultimo acceso: 29/07/2025]

[12] OpenStreetMap: https://www.openstreetmap.org/
[Ultimo acceso: 05/08/2025]