



MÁSTER EN BIG DATA Y BUSINESS ANALYTICS

Integración y Optimización de Datos Clínicos para el Desarrollo de un Modelo Predictivo en Fallos Cardíacos: Un Enfoque de Data Science en el Contexto Hospitalario

TFM elaborado por: Antonio Alvarez Delgado

Tutor de TFM: Juan Manuel Moreno Lamparero

- Madrid 18 de diciembre de 2023 -

Contenido

Resumen.....	6
Introducción	7
Antecedentes	7
Contexto	7
Objetivos	9
Material y métodos	10
Resultados	12
Adquisición de fuentes de datos	12
Datos hospitalarios.....	12
Datos de pacientes	17
Datos clínicos: Fallo del corazón.....	21
Preprocesado de datos y selección de variables.....	22
Datos hospitales	22
Datos pacientes	25
Datos clínicos: Fallo del corazón.....	27
Estrategia big data	28
Aplicación de modelos de aprendizaje automático	29
Preprocesamiento de datos	30
Regresión logística	33
Máquinas de vectores de soporte.....	36
Arboles de decisión	38
Conclusión	40
Aplicación web	41
Dashboard	44
Conclusiones	47
Referencias	48

Índice de figuras

Figura 1.- Centros, servicios y establecimientos sanitarios, Ministerio de Sanidad.....	12
Figura 2.- Web scraping: Librerías y variables.	13
Figura 3.- Web scraping: Bucle encargado de recopilar la información.	13
Figura 4.- Web scraping: Almacenar resultados en archivo CSV.	14
Figura 5.- Finalidad CNH, Centros de salud.	15
Figura 6.- Dependencia Funcional CNH, Centros de salud.	16
Figura 7.- Random User Generator.....	17
Figura 8.- Procedimiento API: Importar librerías y definir variables.	17
Figura 9.- Estructura HTML que contiene la información.	18
Figura 10.- Procedimiento API: Generar archivo CSV y dar nombre a las variables.....	19
Figura 11.- Procedimiento API: Almacenar información en el archivo CSV.....	19
Figura 12.- Procedimiento API: Primera confirmación de formato.	20
Figura 13.- Procedimiento API: Segunda confirmación de formato.....	20
Figura 14.- Archivo CSV con datos de fallos del corazón.....	21
Figura 15.- Limpieza de caracteres: Eliminación de tildes.....	23
Figura 16.- Proceso ETL hospital.....	23
Figura 17.- Proceso ETL hospital: Filtrado de formato.	24
Figura 18.- Proceso ETL hospital: Corrección del formato de los códigos postales mediante script de java.	24
Figura 19.- Proceso ETL hospital: Unificar string, minúsculas.	25
Figura 20.- Proceso ETL hospital: Añadir IDHospital como clave primaria.....	25
Figura 21.- Vinculación paciente-hospital.	26
Figura 22.- Proceso ETL paciente.....	26
Figura 23.- Proceso ETL paciente: Eliminación de variables que causan redundancia.	27
Figura 24.- Proceso ETL paciente: Eliminación de caracteres en string.	27
Figura 25.- Vinculación paciente-fallo del corazón.....	28
Figura 26.- Proceso ETL fallos del corazón.	28
Figura 27.- Estrategia Big data.....	29
Figura 28.- Modelo predictivo: Selección de librerías y fuente de datos.....	30
Figura 29.- Modelo predictivo: Análisis de las variables presentes.	30
Figura 30.- Modelo predictivo: Comprobación de valores nulos.	31
Figura 31.- Modelo predictivo: Análisis de variables categóricas.	31
Figura 32.- Modelo predictivo: Tratado de variables categóricas con LabelEncoder().	32
Figura 33.- Modelo predictivo: Tratado de variables categóricas con get_dummies().	32
Figura 34.- Modelo predictivo: Regresión logística, modelo básico.....	33
Figura 35.- Modelo predictivo: Resultado de regresión logística, modelo básico.	34
Figura 36.- Modelo predictivo: Regresión logística, modelo con análisis de hiperparámetros. .	34
Figura 37.- Modelo predictivo: Resultado de regresión logística, modelo con análisis de hiperparámetros.....	35
Figura 38.- Modelo predictivo: Incompatibilidades de regresión logística, modelo con análisis de hiperparámetros.....	35
Figura 39.- Modelo predictivo: Máquinas de vectores de soporte, modelo básico.....	36
Figura 40.- Modelo predictivo: Resultado de máquinas de vectores de soporte, modelo básico.	36
Figura 41.- Modelo predictivo: Máquinas de vectores de soporte, modelo con análisis de hiperparámetros.....	37

Figura 42.- Modelo predictivo: Resultado de máquinas de vectores de soporte, modelo con análisis de hiperparámetros.	37
Figura 43.- Modelo predictivo: Árboles de decisión, modelo básico.	38
Figura 44.- Modelo predictivo: Resultado de árboles de decisión, modelo básico.	38
Figura 45.- Modelo predictivo: Árboles de decisión, modelo con análisis de hiperparámetros.	39
Figura 46.- Modelo predictivo: Resultado de árboles de decisión, modelo con análisis de hiperparámetros.....	40
Figura 47.- Conclusiones de los modelos predictivos desarrollados.....	40
Figura 48.- Aplicación web: Librerías y modelo predictivo.	41
Figura 49.- Aplicación web: Categorización y normalización de los valores introducidos por el usuario.....	42
Figura 50.- Diseño de la ventana emergente de la aplicación (I).	42
Figura 51.- Diseño de la ventana emergente de la aplicación (II).	43
Figura 52.- Interfaz de la aplicación web.....	43
Figura 53.- Dashboard: Primera hoja.	44
Figura 54.- Dashboard: Segunda hoja.	45

Resumen

En la actualidad, existe una abundancia de información clínica registrada que queda desaprovechada. Este proyecto se presenta como una solución potencial, proponiendo la creación de un banco de datos compartido que permitirá a los profesionales acceder a dicha información para obtener estadísticas y resultados que contribuirán al diagnóstico clínico.

La propuesta se centra en una estrategia de diseño de almacén de datos que asegura un tratamiento adecuado de la información. Como parte integral de este enfoque, se definen dos procesos ETL que facilitan la manipulación de los datos. Una vez que la información está disponible, se implementa un *dashboard* que permite visualizar toda la información recopilada, y se desarrolla un modelo predictivo para analizar los datos de fallos cardíacos en los pacientes.

En un esfuerzo adicional orientado al uso clínico, se ha creado una aplicación web que proporciona de manera instantánea las predicciones generadas. Este enfoque integral no solo busca optimizar el manejo de la información clínica existente, sino también mejorar la eficiencia y la precisión en el diagnóstico a través de herramientas avanzadas de análisis y predicción.

Introducción

Antecedentes

Históricamente, la gestión de datos clínicos ha enfrentado obstáculos significativos en términos de fragmentación y falta de estandarización. La diversidad de fuentes, la variabilidad en la estructura de datos y la falta de integración han obstaculizado los esfuerzos para aprovechar al máximo la información clínica disponible. En este sentido, diversos estudios han demostrado que la implementación de soluciones de *Data Science* puede ofrecer respuestas efectivas a estos desafíos, permitiendo la creación de modelos predictivos que mejoran la eficiencia diagnóstica y la toma de decisiones en el ámbito médico.

La necesidad de optimizar la gestión de datos clínicos relacionados con fallos cardíacos radica en la importancia de proporcionar a los profesionales herramientas avanzadas que faciliten la identificación temprana de patologías cardíacas, permitiendo intervenciones más rápidas y personalizadas. Este TFM se posiciona como un primer enlace para la implementación de estrategias que optimicen el manejo de datos clínicos en entornos hospitalarios, impulsando la creación de un banco de información centralizado y accesible.

En las secciones subsiguientes, se explorarán los objetivos específicos, metodologías propuestas y tecnologías seleccionadas para lograr la integración y optimización de datos clínicos, así como el desarrollo de un modelo predictivo aplicado a fallos cardíacos.

Contexto

Las enfermedades cardiovasculares (ECV) representan la principal causa de mortalidad a nivel global, cobrando alrededor de 17.9 millones de vidas anualmente, lo que constituye el 31% de todas las defunciones en el mundo. Cuatro de cada 5 muertes atribuibles a ECV están relacionadas con ataques cardíacos y accidentes cerebrovasculares, siendo notable que aproximadamente un tercio de estos fallecimientos se producen de manera prematura en individuos menores de 70 años.

La insuficiencia cardíaca emerge como un evento común asociado a las ECV, y el conjunto de datos específico abordado en este contexto consta de 11 características que pueden ser utilizadas para prever posibles enfermedades cardíacas. La detección temprana y el manejo eficaz de personas con enfermedades cardiovasculares o que presentan un alto riesgo cardiovascular (debido a la presencia de uno o más factores de riesgo como hipertensión,

diabetes, hiperlipidemia o enfermedad ya establecida) son imperativos. En este contexto, un modelo de aprendizaje automático puede desempeñar un papel fundamental al proporcionar herramientas predictivas que faciliten la identificación precoz de estas condiciones y permitan una gestión proactiva.

La aplicación de técnicas de aprendizaje automático en el análisis de datos clínicos relacionados con enfermedades cardíacas se convierte en una estrategia clave. A lo largo del proyecto, se aprovechará un conjunto de datos compuesto por 11 variables, con el cual, se busca no solo comprender mejor la incidencia de enfermedades cardíacas, sino también desarrollar un modelo predictivo eficiente que contribuya a la detección temprana y, por ende, a la gestión más efectiva de las enfermedades cardiovasculares. Este enfoque innovador destaca la importancia de la integración de la ciencia de datos en el ámbito médico para mejorar la atención y reducir la carga global de enfermedades cardiovasculares [\[1,2\]](#).

Objetivos

El objetivo principal del proyecto es dar utilidad a la cantidad de datos almacenados proporcionándoles a los profesionales sanitarios una fuente de datos confiable y unas herramientas útiles y eficientes: *dashboard* y modelo predictivo. Este objetivo se puede ver desglosado en las siguientes partes:

- Definir una estrategia de diseño de almacén de datos para datos hospitalarios y clínicos, especializados en datos de fallo del corazón.
- Obtención de datos de distintas fuentes y el procesado para almacenarlo en un *Datawarehouse*
- Creación de *dashboard* como fuente de visualización de información.
- Desarrollo de modelo predictivo a partir de una de las fuentes recogidas
- Crear aplicación interactiva para los profesionales que permita su aplicación directa con el modelo desarrollado.

Material y métodos

El material utilizado ha sido seleccionado en base a las herramientas dadas a lo largo del curso. Aunque han sido proporcionadas una gran variedad, se han seleccionado aquellas que cumplen dos requisitos fundamentales: *open source* y gran conectividad. Las herramientas utilizadas son las siguientes:

- Jupyter, Python
- R, RStudio
- Power BI
- Spoon
- SQLServer (teórica)

Cabe destacar que, para mejorar la escalabilidad y eficiencia del proyecto a futuro, se podría considerar algunas modificaciones en los materiales y tecnologías utilizados. A continuación, se proponen ajustes que podrían contribuir a una mayor escalabilidad:

- Plataforma de Nube: Introducir servicios en la nube, como AWS (Amazon Web Services) o Azure, podría potenciar la escalabilidad del sistema.
- Tecnología de Contenedores: Utilizar tecnologías de contenedores, como Docker, podría simplificar el despliegue y la gestión de aplicaciones, garantizando consistencia y portabilidad en diversos entornos.
- Frameworks de Big Data: En lugar de limitarse a SQL Server, considerar la inclusión de herramientas de Big Data, como Apache Spark, para el procesamiento y análisis de datos a gran escala.
- Automatización: Implementar herramientas de automatización y orquestación, como Apache Airflow, para gestionar y programar automáticamente tareas relacionadas con la obtención, procesamiento y análisis de datos.

En cuanto a métodos y procedimientos llevados a cabo, se nombran los siguientes:

- Obtención de información localizada en *APIs*.
- *Web scrapping* en base a url.
- Desarrollo de ETL devolviendo archivo XML en base a un CSV.
- Procesamiento de datos en dos fases distintas y, consecuentemente, con dos herramientas distintas: Python y Spoon.

- Limpieza de datos para una mejor representación en Power BI.
- Definición de modelos predictivos distintos con comparativa.
- Desarrollo de aplicación web interactiva en base al modelo más optimo.

Resultados

Adquisición de fuentes de datos

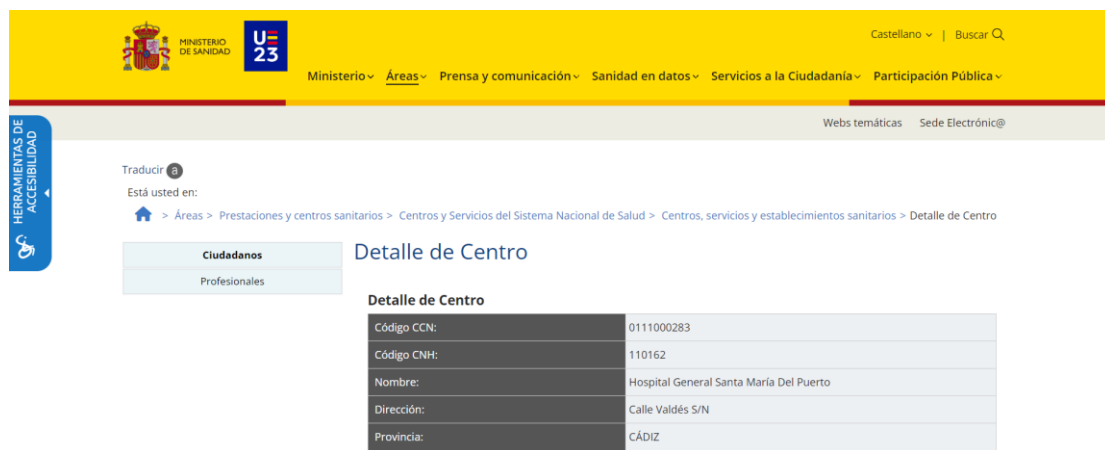
En la adquisición de datos ha prevalecido el uso de distintas técnicas para tener fuentes heterogéneas, permitiendo así poder trabajar más en el procesado de los mismos. En concreto se ha optado por tres fuentes comunes y utilizadas en empresas:

- Archivo CSV
- APIs
- Web scrapping

De igual forma, se han obtenido una fuente de datos distintas para cada una de las técnicas mencionadas.

Datos hospitalarios

Recoge toda la información de hospitales registrada en el Ministerio de Sanidad. Aunque la información aparece en otras extensiones, se ha optado por realizar *web scrapping*. Primeramente, se ha accedido a la página correspondiente a centros y servicios del Sistema Nacional de Salud [3].



The screenshot shows the 'Detalle de Centro' page on the Spanish Ministry of Health website. The page header includes the Ministry of Health logo and navigation links. The main content area displays the details of a specific hospital, including its CCN and CNH codes, name, address, and province.

Detalle de Centro	
Código CCN:	0111000283
Código CNH:	110162
Nombre:	Hospital General Santa María Del Puerto
Dirección:	Calle Valdés S/N
Provincia:	CÁDIZ

Figura 1.- Centros, servicios y establecimientos sanitarios, Ministerio de Sanidad.

Analizando la URL, se observa que los últimos dígitos corresponden al Código CNH del hospital. Por ende, se ha recopilado el código de los hospitales con más camas disponibles para realizar el estudio y se ha desarrollado un código en R encargado de realizar la función de recopilación.

Primeramente, se han importado las librerías que se utilizarán en el desarrollo y se han almacenado en una lista cada uno de los identificadores de los hospitales con los que deseamos trabajar.

```
# Cargar paquetes
library(rvest)
library(stringr)
library(dplyr)

# Identificadores de los hospitales seleccionados
identificadores <- c("110051", "110327", "140230", "140044", "180016", "180150", "230011", "230011", "290017", "290022", "290069", "410021",

# Crear una lista para almacenar los datos de cada hospital
datos_hospitales <- list()
```

Figura 2.- Web scraping: Librerías y variables.

A continuación, se define el bucle el cual va a recopilar toda la información de cada hospital solicitado. Se accede a la web, se localiza la tabla en la que se ubica la información y, si existe, extrae la información de toda la tabla. Para terminar el bucle, se almacena cada dato del hospital en la lista creada anteriormente.

```
# Recorrer la lista de identificadores y obtener los datos de cada hospital
for (identificador in identificadores) {
  # Definir la URL para el hospital actual
  url <- paste0("https://www.sanidad.gob.es/ciudadanos/centros.do?metodo=realizarDetalle&tipo=hospital&numero=", identificador)

  # Realizar la solicitud HTTP y parsear la página
  page <- read_html(url)

  # Encontrar la tabla
  tabla_datos_generales <- page %>% html_node(xpath = '//*[@summary="datos generales del hospital seleccionado"]')

  if (!is.null(tabla_datos_generales)) {
    # Extraer los datos de la tabla
    datos <- tabla_datos_generales %>%
      html_table() %>%
      as.data.frame()

    # Limpiar los datos en la variable "datos" eliminando \r, \n y \t
    datos <- datos %>%
      mutate_all(~str_replace_all(., "[\r\n\t]", ""))

    # Transponer la tabla para invertir filas y columnas
    datos_invertidos <- as.data.frame(t(datos))

    # Establecer la primera fila como nombres de columna
    colnames(datos_invertidos) <- datos_invertidos[1, ]
    datos_invertidos <- datos_invertidos[-1, ] # Eliminar la primera fila

    # Almacenar los datos del hospital en la lista
    datos_hospitales[[identificador]] <- datos_invertidos
  } else {
    cat("No se encontró la tabla para el identificador ", identificador, "\n")
  }
}
```

Figura 3.- Web scraping: Bucle encargado de recopilar la información.

Por último, se crea un *dataframe* con todos los datos y se almacenan en un archivo CSV.

```
# Crear un DataFrame con los datos de todos los hospitales
datos_completos <- do.call(rbind, datos_hospitales)

# Ruta al archivo CSV donde se guardará la información de todos los hospitales
archivo_csv_completo <- "datos_hospitales_completo.csv"

# Guardar los datos de todos los hospitales en un archivo CSV
write.csv(datos_completos, file = archivo_csv_completo, row.names = FALSE)

# Imprimir un mensaje de confirmación
cat("Los datos de todos los hospitales se han guardado en", archivo_csv_completo, "\n")
```

Figura 4.- Web scraping: Almacenar resultados en archivo CSV.

Todo este desarrollo se encuentra en el archivo llamado *WebScrapingFinal.R*.

A continuación, se va a mencionar toda la información recopilada [3].

CÓDIGO DE CENTRO NORMALIZADO REGCESS (CCN): Código interno al sistema del Registro general de centros, servicios y establecimientos sanitarios (REGCESS), asignado por éste a cada centro. Dicho código es unívoco y mantenido en el tiempo para cada centro.

NOTA: Este es el código destinado a servir de referencia para la identificación de cada centro autorizado en los distintos sistemas de información.

Para cada hospital se ofrece tanto el código CNH, con el que se ha venido identificando en el CNH, como el código CCN.

CÓDIGO DE CENTRO ASIGNADO HISTORICAMENTE EN EL CNH (CODCNH): Se refiere al código de 6 dígitos que se ha venido asignando al hospital cuando se incorpora por primera vez al CNH. Este código ha sido generado en el Ministerio de Sanidad y se mantiene a lo largo de la vida del hospital.

NOMBRE DEL CENTRO: Denominación del centro que figura en el registro de la Comunidad Autónoma, Ciudades con Estatuto de Autonomía o Ministerio de Defensa, que consta en su inscripción y autorización y que ha sido trasladado a REGCESS.

CAMAS: Se consideran las camas instaladas, a la fecha de recogida de datos, aquellas que constituyen la dotación fija del hospital y que están en disposición de ser usadas, aunque algunas de ellas puedan, por diversas razones, no estar en servicio en esa fecha.

CLASE DE CENTRO: Referida a alguna de las clases definidas en la Clasificación de centros, servicios y establecimientos sanitarios al que pertenece, según el Anexo I del Real Decreto 1277/2003 por el que se establecen las bases generales sobre autorización de centros, servicios y establecimientos sanitarios.

En él se definen los Hospitales (centros con internamiento) como centros sanitarios destinados a la asistencia especializada y continuada de pacientes en régimen de internamiento (como mínimo una noche), cuya finalidad principal es el diagnóstico o tratamiento de los enfermos ingresados en éstos, sin perjuicio de que también presten atención de forma ambulatoria.

La variable Clase de Centro es equivalente a la variable Finalidad Asistencial que se ha venido utilizando en ediciones anteriores del CNH.

Con el fin de facilitar la interpretación de los datos de esta variable, en la siguiente tabla se ofrece la correspondencia de los valores de finalidad asistencial que se han venido utilizando hasta ahora (2 columnas de la izquierda) con los actuales:

Finalidad CNH (Hasta CNH_2019)		Finalidad (Clase de centro) REGCESS	
Código	Literal	Código	Literal
1	General	C11	Hospitales Generales
2	Quirúrgico	C12	Hospitales especializados
3	Maternal	C12	Hospitales especializados
4	Infantil	C12	Hospitales especializados
5	Materno-Infantil	C12	Hospitales especializados
6	Psiquiátrico	C14	Hospitales de salud mental y tratamiento de toxicomanías
7	Enfermedades Del Tórax	C12	Hospitales especializados
8	Oncológico	C12	Hospitales especializados
9	Oftálmico u ORL	C12	Hospitales especializados
10	Traumatología y/o Rehabilitación	C12	Hospitales especializados
11	Rehabilitación Psicofísica	C12	Hospitales especializados
12	Médico-Quirúrgico	C12	Hospitales especializados
13	Geriatría y/o Larga Estancia	C13	Hospitales de media y larga estancia
14	Otros Monográficos	C190	Otros centros con internamiento
15	Leprológico o Dermatológico	C12	Hospitales especializados
16	Otra Finalidad	C190	Otros centros con internamiento

Figura 5.- Finalidad CNH, Centros de salud.

DEPENDENCIA FUNCIONAL: Se entiende por dependencia funcional de un centro el organismo o entidad jurídica del cual depende, es decir, la persona física o jurídica que ejerce dominio o jurisdicción, jerárquica o funcional, más inmediata sobre el establecimiento sanitario, independientemente de su forma de gestión.

Tanto la definición como los valores actuales se corresponden con los definidos en la Orden SCO/3866/2007, de 18 de diciembre por la que se establece el contenido y la estructura del Registro General de centros, servicios y establecimientos sanitarios del Ministerio de Sanidad y Consumo.

En la tabla siguiente se recogen las correspondencias entre los valores que se venían utilizando anteriormente (dos columnas de la izquierda) con los actuales:

Dependencia Funcional CNH (Hasta CNH_2019)		Dependencia Funcional REGCESS	
Código	Líteral	Código	Líteral
1	Instituto de Gestión Sanitaria-Ingresa	1	Instituto de Gestión Sanitaria/INGESA
2	Servicio Andaluz de Salud	2	Servicios e Institutos de Salud de las comunidades autónomas
3	Instituto Catalán de La Salud	2	Servicios e Institutos de Salud de las comunidades autónomas
4	Servicio Vasco de Salud-Osakidetza	2	Servicios e Institutos de Salud de las comunidades autónomas
5	Conselleria de Sanidad. G. Valenciana	2	Servicios e Institutos de Salud de las comunidades autónomas
6	Servicio Navarro de Salud-Osasunbidea	2	Servicios e Institutos de Salud de las comunidades autónomas
7	Servicio Gallego de Salud-Sergas	2	Servicios e Institutos de Salud de las comunidades autónomas
8	Servicio Canario de Salud	2	Servicios e Institutos de Salud de las comunidades autónomas
12	Instituto De Salud Carlos III	4	Otros centros o establecimientos públicos de dependencia autonómica
13	Otros Hospitales Públicos de Dependencia Estatal	3	Otros centros o establecimientos públicos de dependencia estatal
14	Administración Penitenciaria	3	Otros centros o establecimientos públicos de dependencia estatal
15	Comunidad Autónoma	4	Otros centros o establecimientos públicos de dependencia autonómica
16	Diputación o Cabildo	5	Diputación o Cabildo
17	Municipio	6	Municipio
18	Otros Públicos	8	Otra entidades u organismos públicos
19	Matep	21	Mutuas colaboradoras con la Seguridad Social
20	Privado-Benéfico Cruz Roja	22	Organizaciones no gubernamentales
21	Privado-Benéfico Iglesia	22	Organizaciones no gubernamentales
22	Otro Privado-Benéfico	22	Organizaciones no gubernamentales
23	Privado no Benéfico	23	Otros privados
24	Otra	20	Privados
25	Ministerio de Defensa	7	Ministerio de Defensa
26	Servicio de Salud del Principado de Asturias-SESPA	2	Servicios e Institutos de Salud de las comunidades autónomas
27	Servicio Cántabro de Salud-SCS	2	Servicios e Institutos de Salud de las comunidades autónomas
28	Servicio Riojano de Salud	2	Servicios e Institutos de Salud de las comunidades autónomas
29	Servicio Murciano de Salud	2	Servicios e Institutos de Salud de las comunidades autónomas
30	Servicio Aragonés de Salud-Salud	2	Servicios e Institutos de Salud de las comunidades autónomas
31	Servicio de Salud de Castilla-La Mancha-SESCAM	2	Servicios e Institutos de Salud de las comunidades autónomas
32	Servicio Extremeño de Salud-SES	2	Servicios e Institutos de Salud de las comunidades autónomas
33	Servei De Salut de les Illes Balears- Ib-Salut	2	Servicios e Institutos de Salud de las comunidades autónomas
34	Servicio Madrileño de Salud	2	Servicios e Institutos de Salud de las comunidades autónomas
35	Sanidad Castilla y León-Sacyl	2	Servicios e Institutos de Salud de las comunidades autónomas

Figura 6.- Dependencia Funcional CNH, Centros de salud.

ACREDITACIÓN DOCENTE: Informa que el centro cuenta con la acreditación docente que le capacita para impartir formación sanitaria especializada de postgrado y que en el momento de la recogida de datos disponía de profesionales en formación.

CONCIERTO SNS: Informa si un centro de dependencia privada presta servicios a la Administración Pública Sanitaria, independientemente de que el concierto sea parcial o sustitutorio.

ALTA TECNOLOGÍA. En cada uno de los apartados, se recoge el número de equipos en funcionamiento en el centro, sean o no propiedad del mismo, y con independencia de que estén gestionados por empresas o particulares ajenos a dicho centro.

Datos de pacientes

La siguiente información adquirida es la información personal del paciente. Para ello, se ha accedido a una web de generación de datos de usuarios aleatorios que permite recoger la información vía API [4].

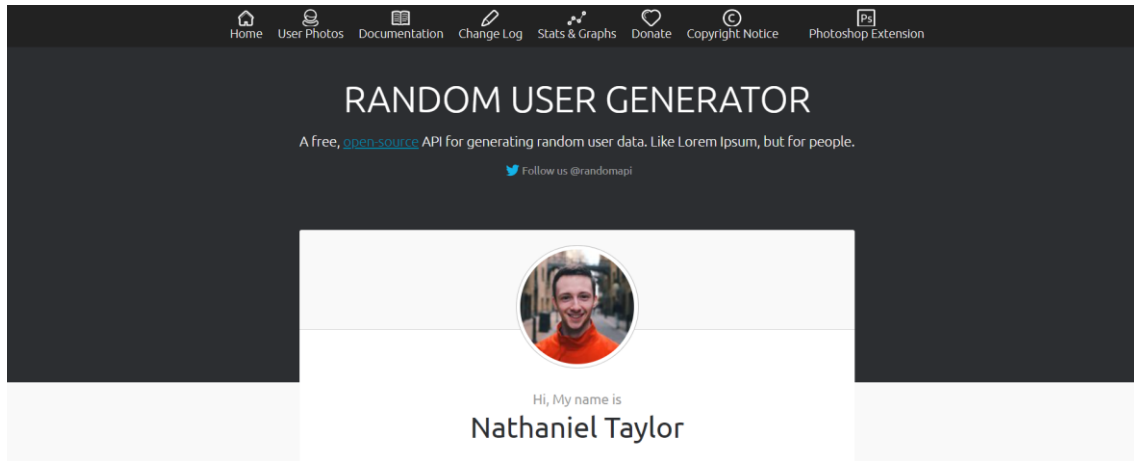


Figura 7.- Random User Generator.

Para realizar dicha funcionalidad, se ha desarrollado un código en Python. Este comienza importando las librerías necesarias, introduciendo la url requerida y realizando una solicitud de tipo GET a la API. Una solicitud de tipo GET es un tipo de solicitud HTTP (Hypertext Transfer Protocol) que se utiliza para recuperar datos de un recurso específico en un servidor. En términos simples, es una solicitud para obtener información de un recurso en un servidor web.

```
import requests
import csv

# URL de la API
api_url = "https://randomuser.me/api/?results=918&nat=ES&noseed&nat=ES&for"

# Realizar una solicitud GET a la API
response = requests.get(api_url)
```

Figura 8.- Procedimiento API: Importar librerías y definir variables.

Este desarrollo se encuentra desarrollado en el archivo *ExtraccionDeDatosPacientes.ipynb*.

La url introducida es:

`https://randomuser.me/api/?results=918&nat=ES&noseed&nat=ES&format=gender,registered,location,phone&gender=both&age=20-70`

A la hora de acceder a la API se aplicaron ciertos filtros para obtener datos acotados:

- results=918: Solicita un total de 918 resultados (usuarios aleatorios).
- nat=ES: Filtra por nacionalidad española (ES).
- noseed: Solicita resultados aleatorios sin utilizar una semilla específica.
- format=gender,registered,location,phone: Define el formato de la respuesta que incluye información sobre género, fecha de registro, ubicación y número de teléfono.
- gender=both: Incluye usuarios de ambos géneros (masculino y femenino).
- age=20-70: Filtra por usuarios con edades comprendidas entre 20 y 70 años.

Una vez se accede, se tiene la siguiente estructura:

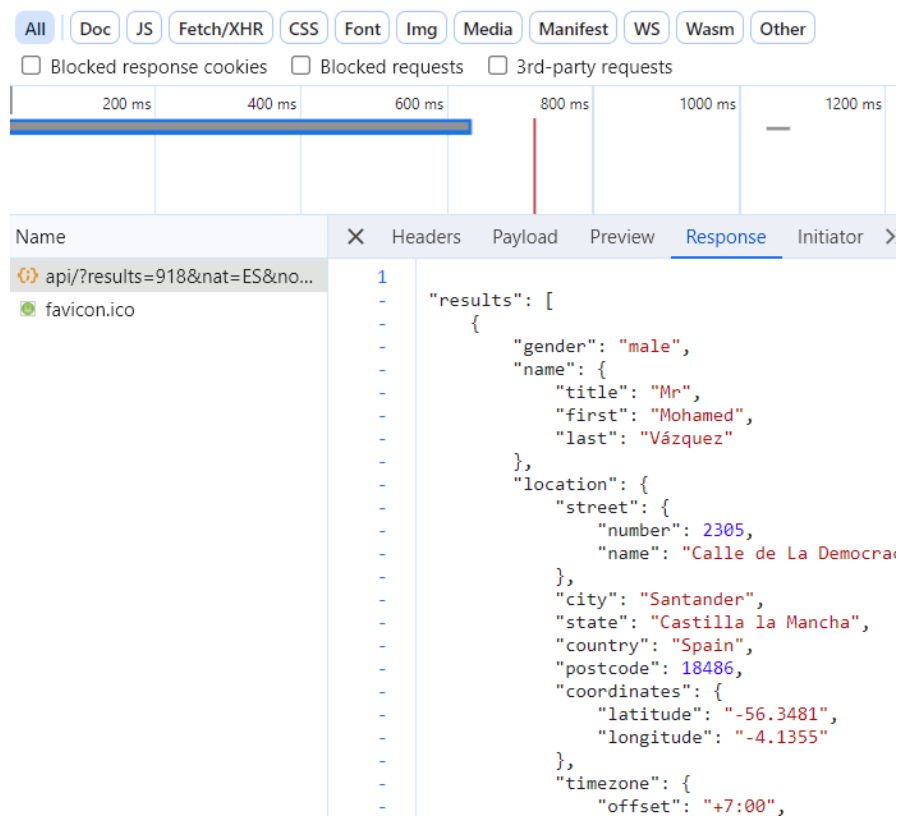


Figura 9.- Estructura HTML que contiene la información.

El siguiente paso es verificar si se puede acceder a ella, status = 200, y si se da el caso, se genera un archivo CSV y se define el nombre de cada una de las variables.

```
# Verificar si la solicitud fue exitosa
if response.status_code == 200:
    data = response.json()["results"]

    # Abrir un archivo CSV para escribir los datos
    with open("datosPacientes.csv", mode="w", newline="", encoding="utf-8") as csv_file:
        fieldnames = [
            "id",
            "nameFirst",
            "nameLast",
            "gender",
            "date",
            "age",
            "country",
            "state",
            "city",
            "streetName",
            "streetNumber",
            "postcode",
            "email",
            "phone"
        ]

        writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
        writer.writeheader()
```

Figura 10.- Procedimiento API: Generar archivo CSV y dar nombre a las variables.

A continuación, se comienza a almacenar la información necesaria en base a la estructura anteriormente planteada. En caso de error, se imprime por pantalla un mensaje.

```
for entry in data:
    row = {
        "id": entry["id"]["value"],
        "nameFirst": entry["name"]["first"],
        "nameLast": entry["name"]["last"],
        "gender": entry["gender"],
        "date": entry["registered"]["date"],
        "age": entry["registered"]["age"],
        "country": entry["location"]["country"],
        "state": entry["location"]["state"],
        "city": entry["location"]["city"],
        "streetName": entry["location"]["street"]["name"],
        "streetNumber": entry["location"]["street"]["number"],
        "postcode": entry["location"]["postcode"],
        "email": entry["email"],
        "phone": entry["phone"]
    }
    writer.writerow(row)

print("Los datos se han guardado en 'datosPacientes.csv'.")

else:
    print("Error al realizar la solicitud a la API:", response.status_code)
```

Figura 11.- Procedimiento API: Almacenar información en el archivo CSV.

De forma adicional, y para evitar problemas de caracteres raros, se ha confirmado el tipo de codificación almacenado.

```
# Realizar una solicitud GET a la API
response = requests.get(api_url)

# Verificar el tipo de codificación en la respuesta
if 'utf-8' in response.headers.get('content-type', '').lower():
    print("Los datos se generan en formato UTF-8.")
else:
    print("Los datos no se generan en formato UTF-8.")

# Imprimir la respuesta para examinar los datos
print(response.text)
```

Los datos se generan en formato UTF-8.

```
{
  "results": [
    {
      "gender": "female",
      "name": {
        "title": "Mrs",
        "first": "Milagros",
        "last": "Ruiz"
      },
      "location": {
        "street": {
          "number": 6756,
          "name": "Calle Mota"
        },
        "city": "Torrejón de Ardoz",
        "state": "Ceuta",
        "country": "Spain",
        "postcode": 92111,
        "coordinates": {
          "latitude": "64.6681",
          "longitude": "84.7161"
        },
        "timezone": {
          "offset": "+10:00",
          "description": "Eastern Australia, Guam, Vladivostok"
        }
      },
      "email": "milagros.ruiz@example.com",
      "login": {
        "uuid": "ad7faed6-ecd5-4650-aab1-ab8e1ccba3",
        "username": "bigpeacock980",
        "password": "randolph",
        "salt": "UYuzzN",
        "md5": "87f89d6432caace927975a0f085ea649",
        "sha1": "c93f29158e6e7a2ef93ab7051cddc034e8ecd9",
        "sha256": "be839bdc05f1d462eea21f1f935a5f038feef6a56c9dbaae9452eab178eae6"
      },
      "dob": {
        "date": "1982-02-27T14:47:36.236Z",
        "age": 41
      }
    }
  ]
}
```

Figura 12.- Procedimiento API: Primera confirmación de formato.

```
import csv

# Abrir un archivo CSV para lectura con la codificación UTF-8
with open("datos.csv", mode="r", encoding="utf-8") as csv_file:
    reader = csv.DictReader(csv_file)
    for row in reader:
        print(row)
```

```
{
  "id-value": "25108947-I",
  "name-first": "Gregorio",
  "name-last": "Ram",
  "gender": "male",
  "registered-date": "2006-06-10T03:33:31.347Z",
  "registered-age": "17",
  "location-country": "Spain",
  "location-state": "Asturias",
  "location-city": "Lugo",
  "location-street-name": "Calle de Téllez",
  "location-street-number": "7523",
  "location-postcode": "10032",
  "email": "gregorio.ra",
  "phone": "937-375-703"
}
{
  "id-value": "01371896-E",
  "name-first": "Miguel",
  "name-last": "Moral",
  "gender": "male",
  "registered-date": "2012-08-07T01:06:32.736Z",
  "registered-age": "11",
  "location-country": "Spain",
  "location-state": "Navarra",
  "location-city": "Pamplona",
  "location-street-name": "Calle de la Virgen",
  "location-street-number": "100",
  "location-postcode": "31001",
  "email": "miguel.moral@example.com",
  "phone": "941-123-456"
}
```

Figura 13.- Procedimiento API: Segunda confirmación de formato.

Datos clínicos: Fallo del corazón

Por último, se ha accedido a un *dataset* de fallos del corazón. Este se ha descargado directamente con formato CSV [2].

	A	B	C	D	E	F	G	H	I
1	Age,Sex,ChestPainType,RestingBP,Cholesterol,FastingBS,RestingECG,MaxHR,ExerciseAngina,Oldpeak,ST_Slope,HeartDisease								
2	40,M,ATA,140,289,0,Normal,172,N,0,Up,0								
3	49,F,NAP,160,180,0,Normal,156,N,1,Flat,1								
4	37,M,ATA,130,283,0,ST,98,N,0,Up,0								
5	48,F,ASY,138,214,0,Normal,108,Y,1.5,Flat,1								
6	54,M,NAP,150,195,0,Normal,122,N,0,Up,0								
7	39,M,NAP,120,339,0,Normal,170,N,0,Up,0								
8	45,F,ATA,130,237,0,Normal,170,N,0,Up,0								
9	54,M,ATA,110,208,0,Normal,142,N,0,Up,0								
10	37,M,ASY,140,207,0,Normal,130,Y,1.5,Flat,1								
11	48,F,ATA,120,284,0,Normal,120,N,0,Up,0								
12	37,F,NAP,130,211,0,Normal,142,N,0,Up,0								
13	58,M,ATA,136,164,0,ST,99,Y,2,Flat,1								
14	39,M,ATA,120,204,0,Normal,145,N,0,Up,0								
15	49,M,ASY,140,234,0,Normal,140,Y,1,Flat,1								
16	42,F,NAP,115,211,0,ST,137,N,0,Up,0								
17	54,F,ATA,120,273,0,Normal,150,N,1.5,Flat,0								
18	38,M,ASY,110,196,0,Normal,166,N,0,Flat,1								

Figura 14.- Archivo CSV con datos de fallos del corazón.

Es interesante mencionar que fue creado combinando diferentes conjuntos de datos ya disponibles de manera independiente, pero que no habían sido combinados previamente. En este conjunto de datos, se han fusionado cinco conjuntos de datos relacionados con enfermedades cardíacas a lo largo de 11 características comunes [2]. Los cinco conjuntos de datos utilizados para su creación son:

- Cleveland: 303 observaciones.
- Hungarian: 294 observaciones.
- Switzerland: 123 observaciones.
- Long Beach VA: 200 observaciones.
- Stalog (Heart) Data Set: 270 observaciones.

La información recopilada hace referencia a los siguientes atributos [2]:

- Edad: Edad del paciente [años]
- Sexo: Sexo del paciente [M: Masculino, F: Femenino]
- Tipo de Dolor en el Pecho: Tipo de dolor en el pecho [TA: Angina Típica, ATA: Angina Atípica, NAP: Dolor no Anginoso, ASY: Asintomático]
- Presión Arterial en Reposo: Presión arterial en reposo [mm Hg]

- Colesterol: Colesterol sérico [mm/dl]
- Glucemia en Ayunas: Nivel de glucosa en sangre en ayunas [1: si Glucemia en Ayunas > 120 mg/dl, 0: de lo contrario]
- Electrocardiograma en Reposo: Resultados del electrocardiograma en reposo [Normal: Normal, ST: con anomalía en la onda ST-T (inversiones de la onda T y/o elevación o depresión de > 0.05 mV), LVH: mostrando hipertrofia ventricular izquierda probable o definitiva según los criterios de Estes]
- Frecuencia Cardíaca Máxima: Frecuencia cardíaca máxima alcanzada [Valor numérico entre 60 y 202]
- Angina Inducida por Ejercicio: Angina inducida por ejercicio [Y: Sí, N: No]
- Depresión Antigua (Oldpeak): Depresión antigua = ST [Valor numérico medido en depresión]
- Pendiente de la Onda ST en el Ejercicio Máximo: La pendiente del segmento ST en el pico del ejercicio [Arriba: ascendente, Plano: plano, Abajo: descendente]
- Enfermedad Cardíaca: Clase de salida [1: enfermedad cardíaca, 0: Normal]

Preprocesado de datos y selección de variables

Datos hospitales

De forma manual, se ha añadido la longitud y latitud de la localización para una visualización más precisa a futuro y se han eliminado las tildes para evitar nuevamente errores de formato. Esto se puede observar en el archivo *LimpiezaDeCaracteres.ipynb*.

Para la eliminación de tildes, se ha recurrido a la librería *unicodedata*.

```
import csv
import unicodedata

def quitar_tildes(input_str):
    nfkd_form = unicodedata.normalize('NFKD', input_str)
    return ''.join([c for c in nfkd_form if not unicodedata.combining(c)])

def quitar_tildes_en_csv(input_file, output_file):
    with open(input_file, 'r', encoding='utf-8') as csvfile:
        reader = csv.reader(csvfile)
        header = next(reader) # Lee la primera fila (encabezado)

        # Aplica la función de quitar_tildes a cada celda del CSV
        data = [[quitar_tildes(cell) for cell in row] for row in reader]

    with open(output_file, 'w', newline='', encoding='utf-8') as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(header)
        writer.writerows(data)

# Aplica función
quitar_tildes_en_csv('datos_hospitales_completo.csv', 'datos_hospitales_final.csv')
```

Figura 15.- Limpieza de caracteres: Eliminación de tildes.

Seguidamente, se ha aplicado un proceso de ETL, denominado *ETLHospitales.ktr*, que cuenta con la siguiente estructura:

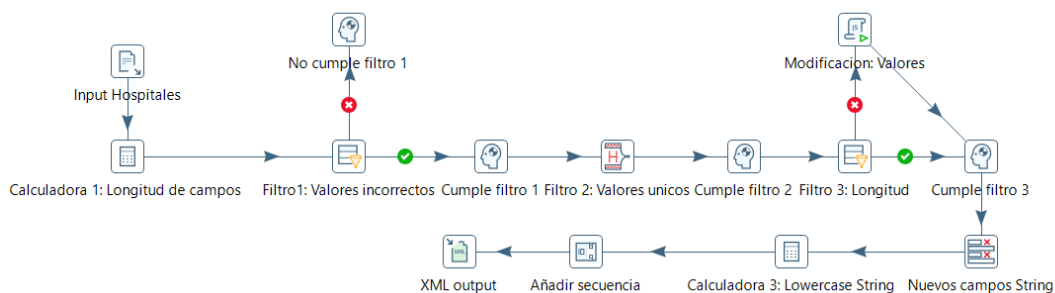


Figura 16.- Proceso ETL hospital.

Este contiene:

- Filtrado de formato de: Código CCN, Código CNH, código postal y número de camas.

Nombre de paso: Filtro1: Valores incorrectos

Enviar 'verdadero' a paso: Cumple filtro 1

Enviar 'falso' a paso: No cumple filtro 1

La condición:

☐

+

CodigoCCN IS NOT NULL

AND

CodigoCNH IS NOT NULL

AND

CodigoCCN_Length = [10]

AND

CodigoCNH_Length = [6]

AND

NumeroDeCamasInstaladas >= [0]

Figura 17.- Proceso ETL hospital: Filtrado de formato.

- Código CCN y Código CNH son valores únicos.
- Corrección de formato en códigos postales, pudiéndose encontrar omitido los ceros a la izquierda.

Nombre de paso: Modificacion: Valores

Java script functions :

- > Transform Scrip
- > Transform Cons
- > Transform Funct
- ▼ Input fields
 - ▶ CodigoCCN
 - ▶ CodigoCNH
 - ▶ Nombre
 - ▶ Direccion
 - ▶ Provincia
 - ▶ ComunidadA
 - ▶ Municipio
 - ▶ CodigoPosta
 - ▶ Telefono
 - ▶ NumeroDeC
 - ▶ Dependenci

Java script :

Script 1

```
// Definimos la variable actual
var currentValue = Codigo Postal;
// Añadimos un 0 a la izquierda
var newValue = '0'+currentvalue;
// Actualizamos el valor
CodigoPostal = newValue;
```

Posición: 6,

Compatibility mod ☐ Optimization level 9

Campos

#	Nombre de campo	Renombar a	Tipo	Longitud	Precisión	Replace value 'Fieldname' or 'Rena
1	CodigoPostal		String			S

Figura 18.- Proceso ETL hospital: Corrección del formato de los códigos postales mediante script de java.

- Campos de tipología *String* en minúsculas para unificar textos.

Nombre paso
Calculadora 3: Lowercase String

☒ Throw an error on non existing files

Campos:

#	Nuevo campo	Cálculo	Campo A	Campo B	Campo C	Tipo de valor	Longitud	Precisión	Eliminar	Conversion mask	Decimal symbol	Gr
1	Nombre	LowerCase of a string A	Nombre_OLD			None			N			
2	Direccion	LowerCase of a string A	Direccion_OLD			None			N			
3	Provincia	LowerCase of a string A	Provincia_OLD			None			N			
4	ComunidadAutonoma	LowerCase of a string A	ComunidadAutonoma_OLD			None			N			
5	Municipio	LowerCase of a string A	Municipio_OLD			None			N			
6	DependenciaFuncional	LowerCase of a string A	DependenciaFuncional_OLD			None			N			
7	ClaseDeCentro	LowerCase of a string A	ClaseDeCentro_OLD			None			N			

Figura 19.- Proceso ETL hospital: Unificar string, minúsculas.

- IDHospital añadido como clave primaria.

Nombre de paso: Añadir secuencia

Nombre de valor: IDHospital

Utilizar una base de datos para generar la secuencia

¿Utilizar base de datos para obtener secuencia? ☐

Conexión: Editar... Nuevo... Wizard...

Nombre de esquema: Schemas...

Nombre de secuencia: SEQ_ Sequences...

Utilizar un contador de la transformación para generar la secuencia

¿Utilizar contador para calcular secuencia? ☒

Nombre contador (opcional):

Valor inicial: 1

Incremento: 1

Valor máximo: 999999999

Figura 20.- Proceso ETL hospital: Añadir IDHospital como clave primaria.

Finalmente, el archivo generado se denomina *DatosHospitales.xml*.

Datos pacientes

En este caso, se han eliminado las tildes para evitar errores de formato UTF8 como en los datos de hospitales, y se ha realizado la vinculación Paciente-Hospital de forma aleatoria. Para realizar el segundo paso, se ha tenido en cuenta la enumeración propuesta en los datos de hospitales.

```

import pandas as pd
import numpy as np
import random

# Cargar datos de hospitales y pacientes desde archivos CSV
datos_hospitales = pd.read_csv('datos_hospitales_completo.csv')
datos_pacientes = pd.read_csv('datosPacientes.csv')

# Valores posibles para la columna "IDHospital"
codigos_IDHospitales_posibles = ["00001", "00002", "00003", "00004", "00005", "00006",
                                  "00010", "00011", "00012", "00013", "00014", "00015", "00016",
                                  "00019", "00020", "00021", "00022", "00023", "00024", "00025",
                                  "00029", "00030"]

# Añadir la nueva columna "CódigoCNH" a los datos de pacientes
datos_pacientes['IDHospital'] = ""

# Recorrer cada fila y asignar un valor aleatorio de la lista de códigos
for index, row in datos_pacientes.iterrows():
    codigo_IDHospitales_aleatorio = random.choice(codigos_IDHospitales_posibles)
    datos_pacientes.at[index, 'IDHospital'] = codigo_IDHospitales_aleatorio

# Guardar el DataFrame actualizado de pacientes en un nuevo archivo CSV
datos_pacientes.to_csv('datos_pacientes_vinculado.csv', index=False)

```

Figura 21.- Vinculación paciente-hospital.

Esta vinculación se muestra en el archivo *VinculacionPacienteHospital.ipynb* mostrado en el anexo.

Posteriormente, se ha aplicado el proceso ETL denominado *ETLPacientes.ktr* que se muestra a continuación.

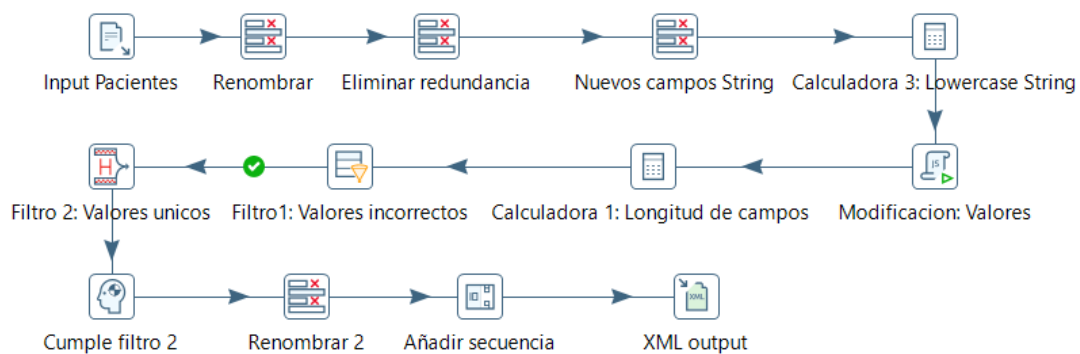


Figura 22.- Proceso ETL paciente.

Este contiene:

- Renombre de columnas y eliminar redundancia de campos. La eliminación de campos se debe a su aparición en otra fuente de datos o a redundancia de campos relacionados.

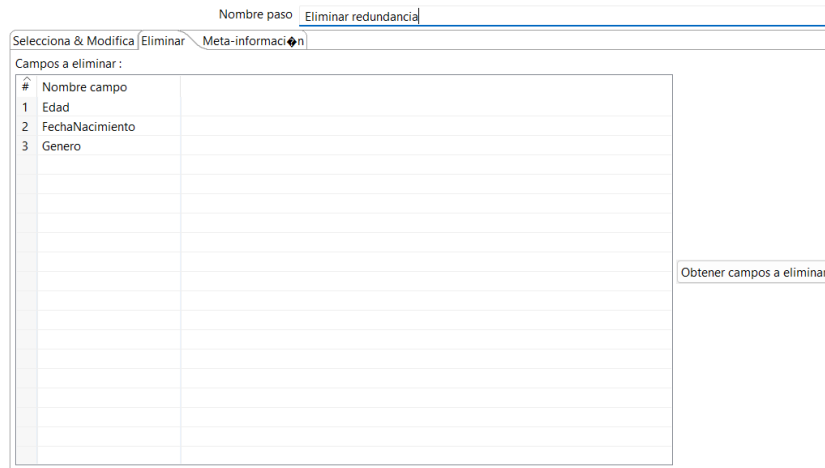


Figura 23.- Proceso ETL paciente: Eliminación de variables que causan redundancia.

- Campos de tipología *String* en minúsculas para unificar textos.
- Eliminar caracteres dentro de *String*.

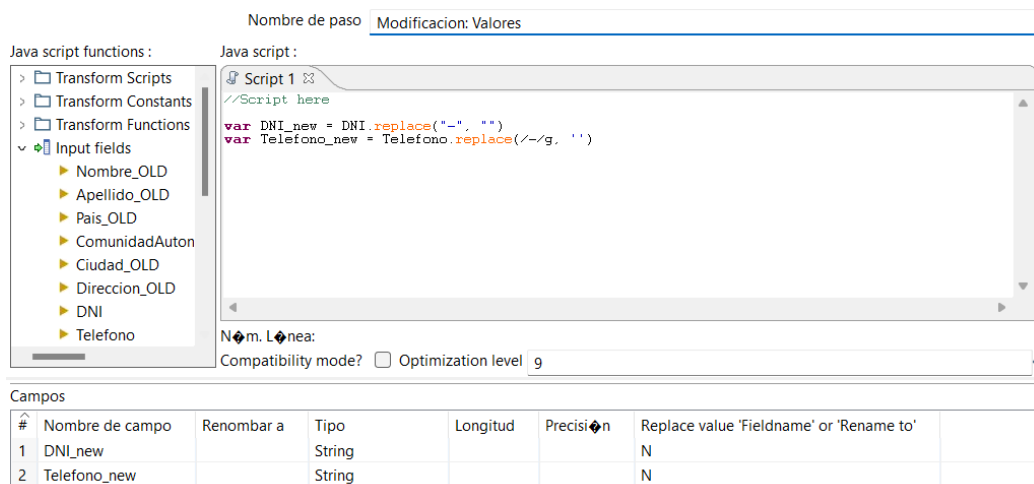


Figura 24.- Proceso ETL paciente: Eliminación de caracteres en string.

- Filtro de valores únicos y formato.
- Añadir IDPaciente como clave primaria.

De igual forma, se genera un archivo final denominado *DatosPacientes.xml*.

Datos clínicos: Fallo del corazón

Primeramente, se ha realizado la vinculación Paciente-FalloDelCorazon entre el archivo CSV Fallo del corazón y el XML generado anteriormente mediante el siguiente código.

```
import pandas as pd
import xml.etree.ElementTree as ET

# Cargar datos del corazón
datos_corazon = pd.read_csv('heart.csv')

# Parsear el archivo XML de pacientes
tree = ET.parse('DatosPacientes.xml')
root = tree.getroot()

# Crear un DataFrame de pandas con los datos de pacientes
columnas_pacientes = ['CodigoPostal', 'Email', 'IDHospital', 'Numero', 'Nombre', 'Apellido', 'Pais',
datos_pacientes = pd.DataFrame(columns=columnas_pacientes)

for paciente in root.findall('Row'):
    datos_paciente = {}
    for columna in columnas_pacientes:
        datos_paciente[columna] = paciente.find(columna).text
    datos_pacientes = datos_pacientes.append(datos_paciente, ignore_index=True)

# Asegurarse de que ambas tablas tienen la misma cantidad de filas
if len(datos_pacientes) == len(datos_corazon):
    # Agregar la columna de ID de pacientes a la tabla de datos del corazón
    datos_corazon['IDPaciente'] = datos_pacientes['IDPaciente']

    # Guardar la tabla de datos del corazón actualizada
    datos_corazon.to_csv('heart_final.csv', index=False)
    print("Columna de ID de pacientes agregada correctamente.")
else:
    print("Las tablas tienen diferentes números de filas y no se puede realizar la copia.")
```

Figura 25.- Vinculación paciente-fallo del corazón.

Este desarrollo se encuentra implementado en el archivo *VinculacionCancerPaciente.ipynb*.

Estos datos vienen con un formato correcto y únicamente se ha renombrado el nombre de las columnas para mantener la uniformidad de lenguaje español. Pudiéndose encontrar en el archivo es *ETLCancer.ktr* del anexo.

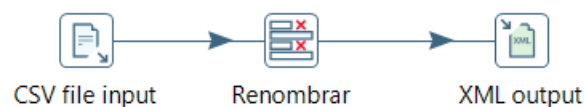


Figura 26.- Proceso ETL fallos del corazón.

Tal y como los casos anteriores, se ha obtenido un archivo llamado *DatosCancer.xml*.

Estrategia big data

La estructura a desarrollar se basa en el *data warehouse de Inmon*, es decir, se plantea un diseño *top-down*. Se busca una arquitectura centralizada que pueda servir como fuente única y del que puedan partir otros proyectos de desarrollo para distintos problemas clínicos. El objetivo es

consolidar los datos de las diversas fuentes mencionadas, pudiendo añadirse más si fuera necesario. Es por ello que se han eliminado posibles redundancias, mejorando la consistencia de la información almacenada. De esta forma, se obtiene una gran capacidad de escalabilidad para manejar volúmenes de datos crecientes y necesidades que puedan verse cambiadas.

En síntesis, se crea una estructura escalable que puede adaptarse a diversos enfoques según sea necesario. Esta se enfoca en datos clínicos relacionados con insuficiencia cardíaca y presentan dos desarrollos: *dashboard* y modelo predictivo con aplicación web.



Figura 27.- Estrategia Big data.

En el proyecto se omitirá la parte de añadir al *datawarehouse* así como del desarrollo de los *data marts*, ya que no se encuentra entre los objetivos del proyecto. Consecuentemente y como se ha mostrado previamente, los datos llegarán al formato XML totalmente óptimos para poder ser almacenados, pero en vez de ello, se emplearán directamente.

Aplicación de modelos de aprendizaje automático

En base a los datos recogidos y las soluciones posibles, positivos (1) y negativos (0), se han tenido en cuenta varios modelos adecuados para clasificación binaria: Regresión logística, Árboles de decisión y Maquinas de vectores de soporte. El objetivo de tener varias opciones es ver cual aporta resultados con un mejor rendimiento. Todo este desarrollo queda recogido en su totalidad en el archivo *Modelling.ipynb*.

Se ha comenzado definiendo las librerías que se utilizaran a primera instancia, estableciendo el directorio actual y seleccionando el archivo con el que se va a trabajar.

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib
```

```
import os

directorio_actual = os.getcwd()
print("Directorio actual:", directorio_actual)
```

Directorio actual: C:\Users\Anton\Desktop\TFM

```
df=pd.read_csv("DatosCancer/heart.csv")
df.head()
```

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0

Figura 28.- Modelo predictivo: Selección de librerías y fuente de datos.

A continuación, se muestra la tipología y descripción de todos los datos existentes.

```
string_col = df.select_dtypes(include="object").columns
df[string_col]=df[string_col].astype("string")
df.dtypes
```

```
Age          int64
Sex          string
ChestPainType string
RestingBP    int64
Cholesterol  int64
FastingBS    int64
RestingECG   string
MaxHR        int64
ExerciseAngina string
Oldpeak      float64
ST_Slope     string
HeartDisease int64
dtype: object
```

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Age	918.0	53.510893	9.432617	28.0	47.00	54.0	60.0	77.0
RestingBP	918.0	132.396514	18.514154	0.0	120.00	130.0	140.0	200.0
Cholesterol	918.0	198.799564	109.384145	0.0	173.25	223.0	267.0	603.0
FastingBS	918.0	0.233115	0.423046	0.0	0.00	0.0	0.0	1.0
MaxHR	918.0	136.809368	25.460334	60.0	120.00	138.0	156.0	202.0
Oldpeak	918.0	0.887364	1.066570	-2.6	0.00	0.6	1.5	6.2
HeartDisease	918.0	0.553377	0.497414	0.0	0.00	1.0	1.0	1.0

Figura 29.- Modelo predictivo: Análisis de las variables presentes.

Preprocesamiento de datos

Se han llevado a cabo los siguientes procesos:

Comprobar que no hay valores 0 que supongan eliminación completa de la fila por tratarse de datos incoherentes.

```
# Buscar si hay valores NULL
df.isnull().sum()
```

```
Age          0
Sex          0
ChestPainType 0
RestingBP    0
Cholesterol  0
FastingBS    0
RestingECG   0
MaxHR        0
ExerciseAngina 0
Oldpeak      0
ST_Slope     0
HeartDisease 0
dtype: int64
```

Figura 30.- Modelo predictivo: Comprobación de valores nulos.

Estudio de las variables categóricas para su posterior tratado.

```
# Tratamiento de variables categoricas
```

```
df[string_col].head()
for col in string_col:
    print(f"La distribución de valores categóricos en la columna {col} es : ")
    print(df[col].value_counts())
```

```
La distribución de valores categóricos en la columna Sex es :
M    725
F    193
Name: Sex, dtype: Int64
La distribución de valores categóricos en la columna ChestPainType es :
ASY    496
NAP    203
ATA    173
TA      46
Name: ChestPainType, dtype: Int64
La distribución de valores categóricos en la columna RestingECG es :
Normal    552
LVH       188
ST         178
Name: RestingECG, dtype: Int64
La distribución de valores categóricos en la columna ExerciseAngina es :
N    547
Y    371
Name: ExerciseAngina, dtype: Int64
La distribución de valores categóricos en la columna ST_Slope es :
Flat    460
Up      395
Down     63
Name: ST_Slope, dtype: Int64
```

Figura 31.- Modelo predictivo: Análisis de variables categóricas.

Seguidamente se procede al tratado de las variables categóricas. Para ello, se seguirán dos posibles tratamientos.

1. El primero es utilizando *Label Encoding*, el cual es útil cuando se tiene variables categóricas ordinales, es decir, categorías con un orden inherente. Este método es eficiente y útil cuando el orden de las categorías importa para el modelo, especialmente en algoritmos basados en árboles de decisión.

```
# Transformar las columnas categóricas en valores numéricos
df_tree = df.apply(LabelEncoder().fit_transform)
df_tree.head()

scaler_tree = MinMaxScaler()
data_tree = pd.DataFrame(scaler.fit_transform(df_tree), columns=df_tree.columns)
```

Figura 32.- Modelo predictivo: Tratado de variables categóricas con `LabelEncoder()`.

2. El segundo es binarizar los datos utilizando *get_dummies*, formato que puede ser utilizado para modelos lineales y máquinas de vector soporte entre otros. Este método es apropiado cuando las categorías no tienen un orden intrínseco y son nominalmente distintas. Si bien es cierto que genera un gran número de columnas, se ha optado por ella para tener más variedad de resultados y consecuentemente, variedad de rendimientos.

```
df_numerical = pd.get_dummies(df, columns=string_col, drop_first=False)
df_numerical.head()
```

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	HeartDisease	Sex_F	Sex_M	ChestPainType_ASY	...	ChestPainType_NAP	ChestPainType_TA	R
0	40	140	289	0	172	0.0	0	0	1	0	...	0	0	
1	49	160	180	0	156	1.0	1	1	0	0	...	1	0	
2	37	130	283	0	98	0.0	0	0	1	0	...	0	0	
3	48	138	214	0	108	1.5	1	1	0	1	...	0	0	
4	54	150	195	0	122	0.0	0	0	1	0	...	1	0	

5 rows x 21 columns

```
# Normalizacion para eliminar dimension y evitar los sesgos con el uso de unidades.
# En este caso se utilizará MINMAXSCALER
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
data = pd.DataFrame(scaler.fit_transform(df_numerical), columns=df_numerical.columns)
print(data)
```

	Age	RestingBP	Cholesterol	FastingBS	MaxHR	Oldpeak	\
0	0.244898	0.70	0.479270	0.0	0.788732	0.295455	
1	0.428571	0.80	0.298507	0.0	0.676056	0.409091	
2	0.182673	0.65	0.460370	0.0	0.267606	0.205455	

Figura 33.- Modelo predictivo: Tratado de variables categóricas con `get_dummies()`.

Regresión logística

Tanto para este modelo como para los posteriores que se van a desarrollar, se va a realizar un primer modelado donde se contemplará tanto el caso con valores predeterminados ofrecidos por el modelo, como el caso con análisis de hiperparámetros buscando los valores más óptimos para el desempeño del modelo.

Primeramente, se expone el modelo con valores predeterminados o modelo básico. Los pasos seguidos con los siguientes:

1. Importar librerías
2. Separación de valores: Variables predictores y variables objetivo.
3. División del conjunto de datos: Entrenamiento y test.
4. Ajuste del modelo, con parámetros predeterminados, utilizando los datos de entrenamiento.
5. Visualización de los parámetros utilizados.
6. Obtención de resultados
7. Visualización de rendimiento de ambos conjuntos de datos.

```
#MODELO BASICO
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix

# Variables predictoras (X): todas las columnas excepto 'HeartDisease'
x_basic = data.drop('HeartDisease', axis=1)

# Variable objetivo (Y): 'HeartDisease'
y_basic = data['HeartDisease']

# Division
x_train_basic, x_test_basic, y_train_basic, y_test_basic = train_test_split(x_basic, y_basic, random_state = 1)

# Ajuste del modelo
classifier_basic = LogisticRegression().fit(x_train_basic, y_train_basic)
print(classifier_basic.get_params())
y_train_pred_basic = classifier_basic.predict(x_train_basic)
y_test_pred_basic = classifier_basic.predict(x_test_basic)

# Mostrar resultados de rendimiento
print('Resultados del conjunto de entrenamiento')
print('Precision: ', accuracy_score(y_train_basic, y_train_pred_basic))
print('Exactitud: ', precision_score(y_train_basic, y_train_pred_basic))
print('Exhaustividad: ', recall_score(y_train_basic, y_train_pred_basic))

print('Resultados del conjunto de test')
print('Precision: ', accuracy_score(y_test_basic, y_test_pred_basic))
print('Exactitud: ', precision_score(y_test_basic, y_test_pred_basic))
print('Exhaustividad: ', recall_score(y_test_basic, y_test_pred_basic))
```

Figura 34.- Modelo predictivo: Regresión logística, modelo básico.

Los resultados obtenidos son los siguientes:

```
{'C': 1.0, 'class_weight': None, 'dual': False, 'fit_intercept': True, 'intercept_scaling': 1, 'l1_ratio': None, 'max_iter': 10
0, 'multi_class': 'auto', 'n_jobs': None, 'penalty': 'l2', 'random_state': None, 'solver': 'lbfgs', 'tol': 0.0001, 'verbose':
0, 'warm_start': False}
Resultados del conjunto de entrenamiento
Precision: 0.8633720930232558
Exactitud: 0.8601583113456465
Exhaustividad: 0.888283378746594
Resultados del conjunto de test
Precision: 0.8826086956521739
Exactitud: 0.9191176470588235
Exhaustividad: 0.8865248226950354
```

Figura 35.- Modelo predictivo: Resultado de regresión logística, modelo básico.

A continuación, se aplica *gridsearchCV* para realizar ajuste de hiperparámetros. Debido a errores de no convergencia, se ha optado por aumentar el número máximo de iteraciones a 10000. Los valores de hiperparámetros analizados son:

- 'penalty': ['l1', 'l2', 'elasticnet']
- 'C': [0.001, 0.01, 0.1, 1, 10, 100]
- 'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']

```
# MODELO CON ANALISIS DE HIPERPARAMETROS
from sklearn.model_selection import GridSearchCV

# Definir el modelo de regresión logística
logistic_classifier = LogisticRegression(max_iter=10000)

# Valores de hiperparámetros a probar
param_grid_logistic = {
    'penalty': ['l1', 'l2', 'elasticnet'],
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
}

# Configurar la búsqueda con validación cruzada
grid_search_logistic = GridSearchCV(logistic_classifier, param_grid_logistic, cv=5, scoring='accuracy')

# Ajustar el modelo a los datos de entrenamiento
grid_search_logistic.fit(x_train_basic, y_train_basic)

# Obtener el mejor conjunto de hiperparámetros
best_params_logistic = grid_search_logistic.best_params_
print(f"Mejor conjunto de hiperparámetros para Regresión Logística: {best_params_logistic}")

# Utilizar el mejor modelo para predecir en los conjuntos de entrenamiento y prueba
y_train_pred_logistic = grid_search_logistic.predict(x_train_basic)
y_test_pred_logistic = grid_search_logistic.predict(x_test_basic)

# Mostrar resultados de rendimiento con el mejor modelo
print('Resultados del conjunto de entrenamiento con ajuste de hiperparámetros (Regresión Logística)')
print('Precisión: ', accuracy_score(y_train_basic, y_train_pred_logistic))
print('Exactitud: ', precision_score(y_train_basic, y_train_pred_logistic))
print('Exhaustividad: ', recall_score(y_train_basic, y_train_pred_logistic))

print('Resultados del conjunto de prueba con ajuste de hiperparámetros (Regresión Logística)')
print('Precisión: ', accuracy_score(y_test_basic, y_test_pred_logistic))
print('Exactitud: ', precision_score(y_test_basic, y_test_pred_logistic))
print('Exhaustividad: ', recall_score(y_test_basic, y_test_pred_logistic))
```

Figura 36.- Modelo predictivo: Regresión logística, modelo con análisis de hiperparámetros.

Para el análisis se han tenido en cuenta el mayor conjunto de opciones posibles. Esto supone que pueden llegar a existir combinaciones incompatibles que no pueden ser analizadas. Los resultados obtenidos son los siguientes.

```
Mejor conjunto de hiperparámetros para Regresión Logística: {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
Resultados del conjunto de entrenamiento con ajuste de hiperparámetros (Regresión Logística)
Precisión: 0.8633720930232558
Exactitud: 0.8545454545454545
Exhaustividad: 0.896457765667575
Resultados del conjunto de prueba con ajuste de hiperparámetros (Regresión Logística)
Precisión: 0.8913043478260869
Exactitud: 0.9202898550724637
Exhaustividad: 0.900709219858156
```

Figura 37.- Modelo predictivo: Resultado de regresión logística, modelo con análisis de hiperparámetros.

Adicionalmente, se puede comprobar las combinaciones incompatibles mediante el siguiente *dataframe*.

```
# Obtener resultados de la búsqueda de hiperparámetros
results = pd.DataFrame(grid_search_logistic.cv_results_)

# Seleccionar columnas relevantes
columns_of_interest = ['params', 'mean_test_score', 'std_test_score']
results_subset = results[columns_of_interest]

# Mostrar la tabla comparativa
print(results_subset)
```

	params	mean_test_score	\
0	{'C': 0.001, 'penalty': 'l1', 'solver': 'newto...		NaN
1	{'C': 0.001, 'penalty': 'l1', 'solver': 'lbfgs'}		NaN
2	{'C': 0.001, 'penalty': 'l1', 'solver': 'libli...	0.466571	
3	{'C': 0.001, 'penalty': 'l1', 'solver': 'sag'}		NaN
4	{'C': 0.001, 'penalty': 'l1', 'solver': 'saga'}	0.508696	
..	...		
85	{'C': 100, 'penalty': 'elasticnet', 'solver': ...}		NaN
86	{'C': 100, 'penalty': 'elasticnet', 'solver': ...}		NaN
87	{'C': 100, 'penalty': 'elasticnet', 'solver': ...}		NaN
88	{'C': 100, 'penalty': 'elasticnet', 'solver': ...}		NaN
89	{'C': 100, 'penalty': 'elasticnet', 'solver': ...}		NaN

	std_test_score
0	NaN
1	NaN
2	0.002688
3	NaN
4	0.032390
..	...
85	NaN
86	NaN
87	NaN
88	NaN
89	NaN

[90 rows x 3 columns]

Figura 38.- Modelo predictivo: Incompatibilidades de regresión logística, modelo con análisis de hiperparámetros.

Máquinas de vectores de soporte

Al igual que en el caso anterior, se comienza importando la librería a utilizar y se reutiliza los dos conjuntos de datos previamente definidos. Esto se realiza debido a que cuentan con el mismo tratamiento de las variables categóricas.

```
# MODELO BASICO
from sklearn.svm import SVC

svm_classifier_basic = SVC().fit(x_train_basic, y_train_basic)
print(svm_classifier_basic.get_params())

y_train_pred_SVM_basic = svm_classifier_basic.predict(x_train_basic)
y_test_pred_SVM_basic = svm_classifier_basic.predict(x_test_basic)

# Mostrar resultados de rendimiento
print('Resultados del conjunto de entrenamiento')
print('Precisión: ', accuracy_score(y_train_basic, y_train_pred_SVM_basic))
print('Exactitud: ', precision_score(y_train_basic, y_train_pred_SVM_basic))
print('Exhaustividad: ', recall_score(y_train_basic, y_train_pred_SVM_basic))

print('Resultados del conjunto de test')
print('Precisión: ', accuracy_score(y_test_basic, y_test_pred_SVM_basic))
print('Exactitud: ', precision_score(y_test_basic, y_test_pred_SVM_basic))
print('Exhaustividad: ', recall_score(y_test_basic, y_test_pred_SVM_basic))
```

Figura 39.- Modelo predictivo: Máquinas de vectores de soporte, modelo básico.

Obteniéndose nuevamente unos valores óptimos para la utilización del modelo predictivo.

```
{'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef
0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gamma': 'scale', 'kern
el': 'rbf', 'max_iter': -1, 'probability': False, 'random_state': None, 'shrink
ing': True, 'tol': 0.001, 'verbose': False}
Resultados del conjunto de entrenamiento
Precisión: 0.872093023255814
Exactitud: 0.8567774936061381
Exhaustividad: 0.9128065395095368
Resultados del conjunto de test
Precisión: 0.8913043478260869
Exactitud: 0.9084507042253521
Exhaustividad: 0.9148936170212766
```

Figura 40.- Modelo predictivo: Resultado de máquinas de vectores de soporte, modelo básico.

Para el caso con ajuste de hiperparámetros, se ha tenido en cuenta los siguientes valores:

- 'C': [0.001, 0.01, 0.1, 1, 10, 100]
- 'kernel': ['linear', 'poly', 'rbf', 'sigmoid']
- 'gamma': ['scale', 'auto', 0.001, 0.01, 0.1, 1, 10, 100]

Además, se ha optado por una validación cruzada dividiendo los datos en 5 conjuntos distintos 'cv=5' y una clasificación en base a la exactitud 'scoring='accuracy'".

```
# MODELO CON AJUSTE DE HIPERPARÁMETROS
from sklearn.model_selection import GridSearchCV

# Definir el modelo de SVM
svm_classifier = SVC()

# Valores de hiperparámetros a probar
param_grid_svm = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'gamma': ['scale', 'auto', 0.001, 0.01, 0.1, 1, 10, 100]
}

# Configurar la búsqueda con validación cruzada
grid_search_svm = GridSearchCV(svm_classifier, param_grid_svm, cv=5, scoring='accuracy')

# Ajustar el modelo a los datos de entrenamiento
grid_search_svm.fit(x_train_basic, y_train_basic)

# Obtener el mejor conjunto de hiperparámetros
best_params_svm = grid_search_svm.best_params_
print(f"Mejor conjunto de hiperparámetros para SVM: {best_params_svm}")

# Utilizar el mejor modelo para predecir en los conjuntos de entrenamiento y prueba
y_train_pred_svm = grid_search_svm.predict(x_train_basic)
y_test_pred_svm = grid_search_svm.predict(x_test_basic)

# Mostrar resultados de rendimiento con el mejor modelo
print('Resultados del conjunto de entrenamiento con ajuste de hiperparámetros (SVM)')
print('Precisión: ', accuracy_score(y_train_basic, y_train_pred_svm))
print('Exactitud: ', precision_score(y_train_basic, y_train_pred_svm))
print('Exhaustividad: ', recall_score(y_train_basic, y_train_pred_svm))

print('Resultados del conjunto de prueba con ajuste de hiperparámetros (SVM)')
print('Precisión: ', accuracy_score(y_test_basic, y_test_pred_svm))
print('Exactitud: ', precision_score(y_test_basic, y_test_pred_svm))
print('Exhaustividad: ', recall_score(y_test_basic, y_test_pred_svm))
```

Figura 41.- Modelo predictivo: Máquinas de vectores de soporte, modelo con análisis de hiperparámetros.

Obteniéndose el siguiente resultado.

```
Mejor conjunto de hiperparámetros para SVM: {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
Resultados del conjunto de entrenamiento con ajuste de hiperparámetros (SVM)
Precisión: 0.8706395348837209
Exactitud: 0.8696808510638298
Exhaustividad: 0.8910081743869209
Resultados del conjunto de prueba con ajuste de hiperparámetros (SVM)
Precisión: 0.8913043478260869
Exactitud: 0.9142857142857143
Exhaustividad: 0.9078014184397163
```

Figura 42.- Modelo predictivo: Resultado de máquinas de vectores de soporte, modelo con análisis de hiperparámetros.

Arboles de decisión

Nuevamente, se comienza importando las librerías a utilizar. A diferencia, las variables categóricas se tratarán como el primer caso de los planteados, utilizando *Label Encoding*. Seguidamente, se divide en dos conjuntos de datos y se realiza el mismo procedimiento.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder

# Transformar las columnas categóricas en valores numéricos
df_tree = df.apply(LabelEncoder().fit_transform)
df_tree.head()

scaler_tree = MinMaxScaler()
data_tree = pd.DataFrame(scaler.fit_transform(df_tree), columns=df_tree.columns)

# Variables predictoras (X): todas las columnas excepto 'HeartDisease'
x_tree = data_tree.drop('HeartDisease', axis=1)

# Variable objetivo (Y): 'HeartDisease'
y_tree = data_tree['HeartDisease']

# División
x_tree_train, x_tree_test, y_tree_train, y_tree_test = train_test_split(x_tree, y_tree, random_state = 1)
```

Figura 43.- Modelo predictivo: Arboles de decisión, modelo básico.

En este caso, se tomará el criterio *entropy* y no se impondrá profundidad alguna ni ningún otro parámetro.

```
DecisionTree_train=DecisionTreeClassifier(criterion="entropy").fit(x_tree_train,y_tree_train)
DecisionTree_test=DecisionTreeClassifier(criterion="entropy").fit(x_tree_test,y_tree_test)
y_train_pred_DdecisionTree=DecisionTree_train.predict(x_tree_train)
y_test_pred_DdecisionTree = DecisionTree_test.predict(x_tree_test)

# Mostrar resultados de rendimiento
print('Resultados del conjunto de entrenamiento')
print('Precisión: ', accuracy_score(y_tree_train,y_train_pred_DdecisionTree))
print('Exactitud: ', precision_score(y_tree_train,y_train_pred_DdecisionTree))
print('Exhaustividad: ', recall_score(y_tree_train,y_train_pred_DdecisionTree))

print('Resultados del conjunto de test')
print('Precisión: ', accuracy_score(y_tree_test,y_test_pred_DdecisionTree))
print('Exactitud: ', precision_score(y_tree_test,y_test_pred_DdecisionTree))
print('Exhaustividad: ', recall_score(y_tree_test,y_test_pred_DdecisionTree))

Resultados del conjunto de entrenamiento
Precisión:  1.0
Exactitud:  1.0
Exhaustividad:  1.0
Resultados del conjunto de test
Precisión:  1.0
Exactitud:  1.0
Exhaustividad:  1.0
```

Figura 44.- Modelo predictivo: Resultado de árboles de decisión, modelo básico.

Obtener unos resultados tan buenos hace sospechar que el modelo se encuentra sobreajustado. Por ello, se va a realizar directamente el ajuste de hiperparámetros con *gridsearchCV*. Los valores que se analizan son los siguientes:

- 'criterion': ['gini', 'entropy']
- 'max_depth': [None, 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 29, 30]
- 'min_samples_split': [2, 5, 10]
- 'min_samples_leaf': [1, 2, 4]

```
from sklearn.model_selection import GridSearchCV

# Definir el modelo de árbol de decisiones
tree_classifier = DecisionTreeClassifier()

# Valores de hiperparámetros a probar
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 29, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Configurar la búsqueda con validación cruzada
grid_search = GridSearchCV(tree_classifier, param_grid, cv=5, scoring='accuracy')

# Ajustar el modelo a los datos de entrenamiento
grid_search.fit(x_tree_train, y_tree_train)

# Obtener el mejor conjunto de hiperparámetros
best_params = grid_search.best_params_
print(f"Mejor conjunto de hiperparámetros: {best_params}")

# Utilizar el mejor modelo para predecir en los conjuntos de entrenamiento y prueba
y_train_pred_best = grid_search.predict(x_tree_train)
y_test_pred_best = grid_search.predict(x_tree_test)

# Mostrar resultados de rendimiento con el mejor modelo
print('Resultados del conjunto de entrenamiento con ajuste de hiperparámetros')
print('Precisión: ', accuracy_score(y_tree_train, y_train_pred_best))
print('Exactitud: ', precision_score(y_tree_train, y_train_pred_best))
print('Exhaustividad: ', recall_score(y_tree_train, y_train_pred_best))

print('Resultados del conjunto de prueba con ajuste de hiperparámetros')
print('Precisión: ', accuracy_score(y_tree_test, y_test_pred_best))
print('Exactitud: ', precision_score(y_tree_test, y_test_pred_best))
print('Exhaustividad: ', recall_score(y_tree_test, y_test_pred_best))
```

Figura 45.- Modelo predictivo: Árboles de decisión, modelo con análisis de hiperparámetros.

Obteniéndose como resultado lo siguiente.

```
Mejor conjunto de hiperparámetros: {'criterion': 'entropy', 'max_depth': 5, 'min_samples_leaf': 4, 'min_samples_split': 2}
Resultados del conjunto de entrenamiento con ajuste de hiperparámetros
Precisión: 0.8880813953488372
Exactitud: 0.8897849462365591
Exhaustividad: 0.9019073569482289
Resultados del conjunto de prueba con ajuste de hiperparámetros
Precisión: 0.8521739130434782
Exactitud: 0.8794326241134752
Exhaustividad: 0.8794326241134752
```

Figura 46.- Modelo predictivo: Resultado de árboles de decisión, modelo con análisis de hiperparámetros.

Conclusión

Realizando un promedio de los resultados obtenidos, se obtiene lo siguiente:

```
Regresión logística:
Básico - Precisión: 0.8729903943377149 Analizado - Precisión: 0.8773382204246714
Básico - Exactitud: 0.889637979202235 Analizado - Exactitud: 0.8874176548089592
Básico - Exhaustividad: 0.8874041007208147 Analizado - Exhaustividad: 0.8985834927628655
Máquinas de vectores de soporte:
Básico - Precisión: 0.8816986855409504 Analizado - Precisión: 0.880971941354904
Básico - Exactitud: 0.8826140989157452 Analizado - Exactitud: 0.891983282674772
Básico - Exhaustividad: 0.9138500782654067 Analizado - Exhaustividad: 0.8994047964133186
Árboles de decisión:
Analizado - Precisión: 0.8701276541961578
Analizado - Exactitud: 0.8873385162837473
Analizado - Exhaustividad: 0.8871238912400718
```

Figura 47.- Conclusiones de los modelos predictivos desarrollados.

En cuanto a la regresión logística se encuentran valores muy semejantes, pero se observa una leve mejora en los resultados del modelo con análisis de hiperparámetros. Esto se puede deber a la diferencia de valores encontradas en los hiperparámetros 'C' y 'solver'.

Respecto a las máquinas de vectores de soporte, se encuentran diferencias en los valores de 'C' y 'gamma'. Al igual que en el caso anterior, se muestran resultados muy parecidos. Debido a la leve mejora, y a la comparativa de hiperparámetros realizada, se selecciona como mejor opción la analizada.

En el caso del árbol de decisión, no se ha tenido en cuenta los resultados con sobreajustes, seleccionando por óptimos los valores obtenidos tras el análisis de hiperparámetros.

Para concluir, aunque se han obtenido rendimientos excelentes y semejantes, se percibe una leve mejora en aquellos que se ha realizado el análisis de hiperparámetros, lo cual es coherente dado que se analizan numerosas combinaciones posibles y se complementa con la validación cruzada. Por lo tanto, se opta por trabajar con el modelo de Máquinas de Vectores de Soporte que ha sido sometido a un análisis de hiperparámetros para realizar predicciones.

Aplicación web

Adicionalmente, a partir del modelo seleccionado se ha desarrollado una aplicación web que permite al usuario introducir los datos de las variables y obtener la predicción (1 o 0). Para ello, se comienza importando todas las librerías a utilizar e incluyendo el desarrollo práctico llevado a cabo en el modelado.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
import tkinter as tk
from tkinter import ttk

df=pd.read_csv("DatosCancer/heart.csv")
string_col = df.select_dtypes(include="object").columns
df[string_col]=df[string_col].astype("string")
df[string_col].head()
df_numerical=pd.get_dummies(df,columns=string_col,drop_first=False)
df_numerical.head()

scaler = MinMaxScaler()
data = pd.DataFrame(scaler.fit_transform(df_numerical), columns=df_numerical.columns)

# Variables predictoras (X): todas las columnas excepto 'HeartDisease'
x = data.drop('HeartDisease', axis=1)

# Variable objetivo (Y): 'HeartDisease'
y = data['HeartDisease']

# Division
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state = 1)

# Ajuste del modelo
classifier = SVC(C=10,gamma=0.01,kernel='rbf').fit(x_train, y_train)
y_train_pred = classifier.predict(x_train)
y_test_pred = classifier.predict(x_test)
```

Figura 48.- Aplicación web: Librerías y modelo predictivo.

Seguidamente, se crea una función para recoger los valores que el usuario ingresará en la calculadora y se aplicarán al modelo ya obtenido. Los datos introducidos se incluirán en todos los datos ya registrados y, se realizará la categorización y normalización con ellos. Este paso permitirá transformar los datos introducidos por el usuario a datos normalizados y categorizados listos para ser utilizado por el modelo. Una vez aplicado, la información introducida por el usuario se aplica al modelo.

```
def prediccion():
    try:
        # Obtener los valores ingresados por el usuario
        datos = {
            'Age': int(entradas_variables[0].get()),
            'Sex': str(entradas_variables[1].get()),
            'ChestPainType': str(entradas_variables[2].get()),
            'RestingBP': int(entradas_variables[3].get()),
            'Cholesterol': int(entradas_variables[4].get()),
            'FastingBS': int(entradas_variables[5].get()),
            'RestingECG': str(entradas_variables[6].get()),
            'MaxHR': int(entradas_variables[7].get()),
            'ExerciseAngina': str(entradas_variables[8].get()),
            'Oldpeak': int(entradas_variables[9].get()),
            'ST_Slope': str(entradas_variables[10].get()),
        }
        data_input = pd.DataFrame(datos, index=[0])
        # Añadir la nueva fila al principio del DataFrame original
        df_input = pd.concat([data_input, df], ignore_index=True)
        df_input = df_input.drop(columns=['HeartDisease'])
        string_col = df_input.select_dtypes(include="object").columns
        df_input[string_col] = df_input[string_col].astype("string")
        df_numerical_input = pd.get_dummies(df_input, columns=string_col, drop_first=False)
        scaler_input = MinMaxScaler()
        data = pd.DataFrame(scaler_input.fit_transform(df_numerical_input), columns=df_numerical_input.columns)
        y_output = classifier.predict(data.head(1))
        resultado_var.set(f"Resultado clínico: {int(y_output)}")

    except Exception as e:
        resultado_var.set(f"Error: {str(e)}")
```

Figura 49.- Aplicación web: Categorización y normalización de los valores introducidos por el usuario.

Por último, se realiza el diseño de la ventana emergente con la que trabajará el usuario.

```
# Crear la ventana principal
ventana = tk.Tk()
ventana.title("Calculadora de fallo cardíaco")

# Crear etiquetas y entradas para las variables
etiquetas_variables = []
entradas_variables = []
nombre_variables = ["Edad", "Género", "Dolor en el pecho", "Presion arterial en reposo", "Colesterol", "Glucemia en ayunas", "Electrocardiograma en reposo", "Frecuencia cardiaca máxima", "Angina", "Oldpeak", "ST_Slope"]

for i in range(0, 11):
    etiqueta = ttk.Label(ventana, text=f"{nombre_variables[i]}:", anchor="w")
    etiqueta.grid(row=i, column=0, padx=10, pady=5)
    etiquetas_variables.append(etiqueta)

    entry_variable = ttk.Entry(ventana)
    entry_variable.grid(row=i, column=1, padx=10, pady=5)
    entradas_variables.append(entry_variable)
```

Figura 50.- Diseño de la ventana emergente de la aplicación (I).

```
# Etiqueta de informacion adicional
precision_label = ttk.Label(ventana, text="Mínimo rendimiento obtenido del modelo", anchor="w")
precision_label.grid(row=0, column=2, padx=10, pady=5)

exhaustividad_label = ttk.Label(ventana, text="Precisión: 0.881", anchor="w")
exhaustividad_label.grid(row=1, column=2, padx=10, pady=5)

generosidad_label = ttk.Label(ventana, text="Exactitud: 0.892", anchor="w")
generosidad_label.grid(row=2, column=2, padx=10, pady=5)

generosidad_label = ttk.Label(ventana, text="Exhaustividad: 0.899", anchor="w")
generosidad_label.grid(row=3, column=2, padx=10, pady=5)

# Botón para calcular la suma
boton_calcular = ttk.Button(ventana, text="Calculadora", command=prediccion)
boton_calcular.grid(row=13, column=0, columnspan=2, pady=10)

# Etiqueta para mostrar el resultado
resultado_var = tk.StringVar()
etiqueta_resultado = ttk.Label(ventana, textvariable=resultado_var)
etiqueta_resultado.grid(row=14, column=0, columnspan=2, pady=5)

# Iniciar el bucle principal
ventana.mainloop()
```

Figura 51.- Diseño de la ventana emergente de la aplicación (II).

Obteniéndose como resultado la siguiente representación.

Figura 52.- Interfaz de la aplicación web.

El desarrollo de dicha aplicación se encuentra disponible en el archivo *AplicacionLaptop.py* del anexo.

Dashboard

Dado que la parte de introducir los datos en las bases de datos fue omitida, los archivos XMLs generados se utilizan directamente para la elaboración del *dashboard*. Posteriormente, se han realizado unas modificaciones de formato y mejoras para una mejor visualización. En concreto:

- Cambio de tipología de variables.
- Reemplazo de variables.
- Añadir contador.
- Añadir columnas personalizadas.
- Cambio de nombre de columnas.

Los resultados se muestran a continuación, y aparecen en el archivo *Dashboard.pbix*.

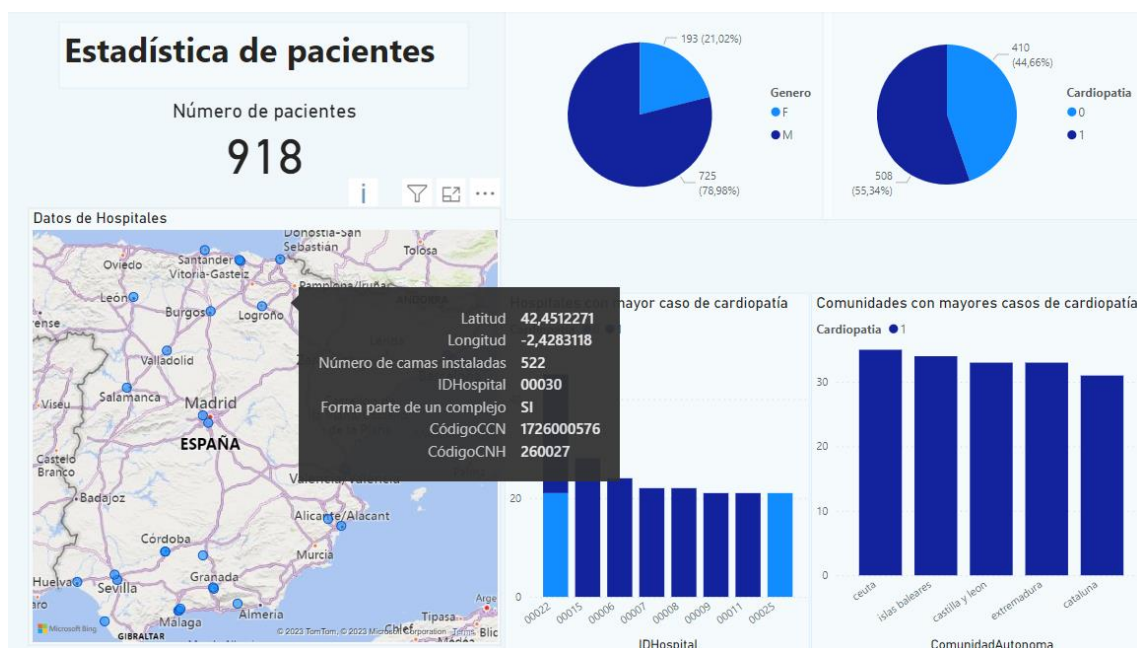


Figura 53.- Dashboard: Primera hoja.

Primera hoja, donde se muestran todos los hospitales utilizados en el estudio. Posicionándose sobre cada uno, aparece una ventana emergente con información detallada, brindando una visión completa de la red hospitalaria involucrada. Esta función interactiva permite acceder fácilmente a detalles clave de cada centro médico, como su ubicación, capacidad, y especialidades relevantes.

Además, de manera rápida e intuitiva, se pueden observar las siguientes estadísticas cruciales para la toma de decisiones estratégicas:

- El número total de pacientes que ha participado en el estudio, proporcionando una perspectiva global sobre la amplitud de la muestra.
- La distribución del género de los pacientes, facilitando una comprensión rápida de posibles disparidades y necesidades específicas.
- La proporción de pacientes con cardiopatía, ofreciendo una visión sobre la prevalencia de la enfermedad en la población estudiada.
- Los hospitales donde se han tratado el mayor número de casos positivos, destacando aquellos centros con una carga significativa de pacientes con enfermedades cardíacas.
- Las comunidades donde se encuentra el mayor número de casos, identificando áreas geográficas con una mayor incidencia de cardiopatías.

Estos indicadores clave proporcionan una base sólida para la asignación eficiente de fondos destinados al tratamiento de la cardiopatía. La información presentada en esta primera hoja se concibe como un recurso valioso para los cargos superiores, brindando una perspectiva integral y visualmente accesible que puede informar de manera efectiva las decisiones estratégicas y la planificación de recursos en el ámbito de la salud cardiovascular.



Figura 54.- Dashboard: Segunda hoja.

Respecto a la segunda hoja, diseñada específicamente para trabajadores que utilizan los datos clínicos a diario, se ha concebido como una herramienta integral que proporciona acceso

eficiente a la totalidad de los datos clínicos y personales de los pacientes. Esta hoja está diseñada para facilitar la labor diaria de los profesionales de la salud, permitiéndoles realizar búsquedas sencillas y obtener información de manera rápida y precisa.

La interfaz de la segunda hoja se ha estructurado de manera intuitiva, con un filtro destacado que permite buscar pacientes por su identificación (ID). Al ingresar el ID correspondiente, la hoja responde instantáneamente mostrando todos los datos clínicos y personales asociados al paciente en cuestión. Esta función agiliza significativamente el acceso a información específica, proporcionando a los trabajadores la capacidad de recuperar datos relevantes de forma inmediata.

Además, se ha incorporado un diseño limpio y organizado que facilita la visualización de datos detallados sin generar confusiones. Este enfoque centrado en la usabilidad está destinado a mejorar la eficiencia y productividad de aquellos profesionales que dependen de datos clínicos a diario para realizar diagnósticos, tratamientos y seguimientos de pacientes.

En conclusión, la segunda hoja del *dashboard* se presenta como una herramienta esencial para el personal médico, brindando un acceso rápido y simplificado a la información clínica y personal de los pacientes, con un énfasis particular en la capacidad de búsqueda por ID para garantizar una eficiencia operativa en el manejo cotidiano de datos clínicos.

Conclusiones

En base al proyecto realizado, se pueden derivar las siguientes conclusiones:

- El proyecto ha logrado el objetivo principal de proporcionar utilidad de datos de distintas fuentes. En concreto, se ha definido una estructura de diseño de almacén de datos, se ha desarrollado un *dashboard* y se ha obtenido un modelo predictivo óptimo con aplicación web.
- Tanto las herramientas utilizadas como los métodos han sido eficientes para el procesamiento y análisis de los datos.
- La inclusión de datos de distintas fuentes ha enriquecido la heterogeneidad del conjunto de datos.
- Se ha realizado un exhaustivo preprocesamiento de datos, incluyendo la limpieza, normalización y vinculación de datos provenientes de diferentes fuentes.
- Se han aplicado diversos modelos de aprendizaje automático con distintas configuraciones, siendo seleccionado como modelo final máquinas de vectores de soporte debido a los resultados comparativos y mejora de rendimiento.
- Se ha implementado un *dashboard* que ofrece visualizaciones interactivas para profesionales sanitarios, y una aplicación web que permite obtener predicciones basadas en el modelo desarrollado.
- Se han propuesto ajustes para mejores futuras en cuanto a escalabilidad, servicios en la nube, tecnologías de contenedores y diversos *frameworks*.

En resumen, el proyecto ha abordado de manera integral la gestión de datos en el ámbito sanitario, desde la adquisición y procesamiento hasta el modelado predictivo y la creación de herramientas interactivas para profesionales. Las conclusiones destacan el logro de los objetivos planteados, la selección acertada de herramientas y métodos, así como la atención a la calidad y diversidad de los datos.

Referencias

- 1- Organización Mundial de la Salud. Recogido el 12 de noviembre de 2023, de https://www.who.int/es/health-topics/cardiovascular-diseases#tab=tab_1
- 2- Soriano, Federico. Conjunto de datos para la predicción de fallo cardíaco. Recogido el 16 de noviembre de 2023, de <https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction/data>
- 3- Ministerio de Sanidad. Centros y Servicios del Sistema Nacional de Salud. Recogido el 18 de noviembre de 2023, de <https://www.sanidad.gob.es/ciudadanos/centros.do?metodo=realizarDetalle&tipo=hospital&numero=110162>
- 4- Hunt, Arron y Armstrong, Keith. Open source API para la generación de usuarios aleatorios, Random User Generator. Recogido el 18 de noviembre de 2023, de <https://randomuser.me/>