IMF Smart Education

MASTER IN BIG DATA AND BUSINESS ANALYTICS

Integration and Optimization of Clinical Data for the Development of a Predictive Model in Heart Failure: A Data Science Approach in the Hospital Context

Master's Thesis prepared by: Antonio Alvarez Delgado

TFM tutor: Juan Manuel Moreno Lamparero

- Madrid, December 18, 2023 -

Content

Figure Index

Summary

Currently, there is an abundance of recorded clinical information that is not being tapped. This project is presented as a potential solution, proposing the creation of a shared database that will allow professionals to access this information to obtain statistics and results that will contribute to clinical diagnosis.

The proposal focuses on a data warehouse design strategy that ensures adequate treatment of information. As an integral part of this approach, two ETL processes are defined that facilitate data manipulation. Once the information is available, a dashboard is implemented that allows all the information collected to be visualized, and a predictive model is developed to analyze the data on heart failure in patients.

In an additional effort aimed at clinical use, a web application has been created that instantly provides the generated predictions. This comprehensive approach not only seeks to optimize the management of existing clinical information, but also to improve efficiency and accuracy in diagnosis through advanced analysis and prediction tools.

Introduction

Background

Historically, clinical data management has faced significant obstacles in terms of fragmentation and lack of standardization. Diversity of sources, variability in data structure, and lack of integration have hampered efforts to make the most of available clinical information. In this sense, various studies have shown that the implementation of *Data Science* solutions can offer effective responses to these challenges, allowing the creation of predictive models that improve diagnostic efficiency and decision-making in the medical field.

The need to optimize the management of clinical data related to heart failure lies in the importance of providing professionals with advanced tools that facilitate the early identification of cardiac pathologies, allowing faster and more personalized interventions. This Master's Thesis is positioned as a first link for the implementation of strategies that optimize the management of clinical data in hospital environments, promoting the creation of a centralized and accessible information bank.

In the following sections, the specific objectives, proposed methodologies and selected technologies to achieve the integration and optimization of clinical data, as well as the development of a predictive model applied to heart failure, will be explored.

Context

Cardiovascular disease (CVD) is the leading cause of death globally, claiming around 17.9 million lives annually, accounting for 31% of all deaths worldwide. Four out of 5 deaths attributable to CVD are related to heart attacks and strokes, and it is notable that approximately one-third of these deaths occur prematurely in individuals under 70 years of age.

Heart failure emerges as a common event associated with CVD, and the specific dataset addressed in this context consists of 11 characteristics that can be used to predict possible heart disease. Early detection and effective management of people with cardiovascular disease or who are at high cardiovascular risk (due to the presence of one or more risk factors such as hypertension, diabetes, hyperlipidemia, or already established disease) are imperative. In this context, a machine learning model can play a critical role in providing predictive tools that facilitate the early identification of these conditions and enable proactive management.

The application of machine learning techniques in the analysis of clinical data related to heart disease becomes a key strategy. Throughout the project, a dataset composed of 11 variables will be used, with which it seeks not only to better understand the incidence of heart disease, but also to develop an efficient predictive model that contributes to the early detection and, therefore, to the most effective management of cardiovascular diseases. This innovative approach highlights the importance of integrating data science into the medical field to improve care and reduce the global burden of cardiovascular disease [1,2].

Objectives

The main objective of the project is to make use of the amount of data stored by providing healthcare professionals with a reliable data source and useful and efficient tools: *dashboard* and predictive model. This objective can be broken down into the following parts:

- Define a data warehouse design strategy for hospital and clinical data, specializing in heart failure data.
- Obtaining data from different sources and processing it to store it in a *Datawarehouse*
- Creation of *a dashboard* as a source of information visualization.
- Development of a predictive model from one of the sources collected
- Create an interactive application for professionals that allows its direct application with the developed model.

Material and methods

The material used has been selected based on the tools given throughout the course. Although a wide variety has been provided, those that meet two fundamental requirements have been selected: *open source* and great connectivity. The tools used are as follows:

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

It should be noted that, in order to improve the scalability and efficiency of the project in the future, some modifications in the materials and technologies used could be considered. The following are proposed adjustments that could contribute to greater scalability:

- Cloud Platform: Introducing cloud services, such as AWS (Amazon Web Services) or Azure, could enhance the scalability of the system.
- Container Technology: Using container technologies, such as Docker, could simplify application deployment and management, ensuring consistency and portability across diverse environments.
- *Big Data* Frameworks: Rather than being limited to SQL Server, consider including Big Data tools, such as Apache Spark, for large-scale data processing and analysis.
- Automation: Implement automation and orchestration tools, such as Apache Airflow, to automatically manage and schedule tasks related to data collection, processing, and analysis.

In terms of methods and procedures carried out, the following are named:

- Obtaining localized information in *APIs*.
- *Web scrapping* based on url.
- ETL development by returning XML file based on a CSV.
- Data processing in two different phases and, consequently, with two different tools: Python and Spoon.
- Data cleansing for better representation in Power BI.
- Definition of different predictive models with comparison.

- Development of an interactive web application based on the most optimal model.

Results

Acquiring Data Sources

In data acquisition, the use of different techniques has prevailed to have heterogeneous sources, thus allowing more work on data processing. Specifically, three common sources have been chosen and used in companies:

- CSV file
- Apis
- Web scrapping

Similarly, a different data source has been obtained for each of the aforementioned techniques.

Hospital Data

It collects all the information on hospitals registered with the Ministry of Health. Although the information appears in other extensions, *web scraping* has been chosen. Firstly, the page corresponding to centres and services of the National Health System has been accessed [3].

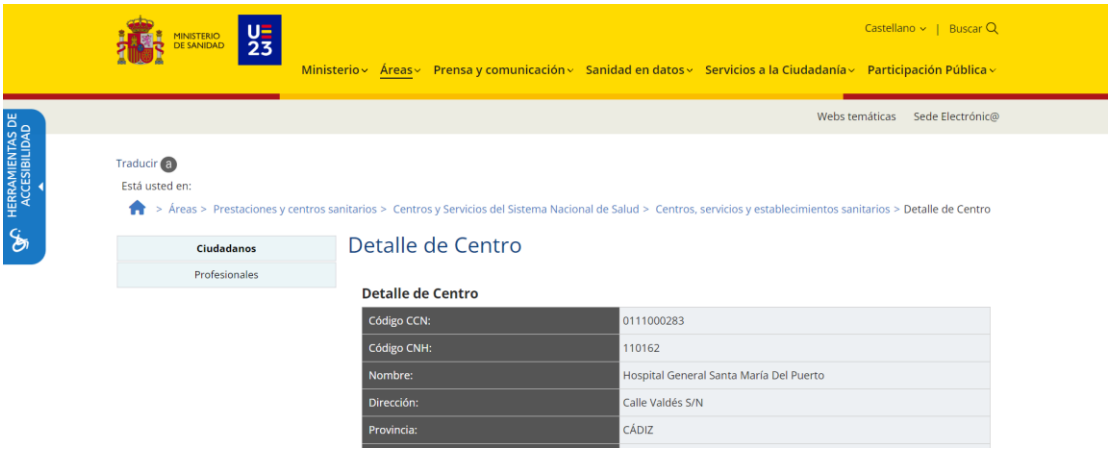

Figure 1.- *Health centres, services and establishments, Ministry of Health.*

Analyzing the URL, it is observed that the last digits correspond to the CNH Code of the hospital. Therefore, the code of the hospitals with the most available beds to carry out the study has been collected and an R code has been developed to perform the collection function.

Firstly, the libraries that will be used in the development have been imported and each of the identifiers of the hospitals with which we want to work have been stored in a list.

```r
# 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()
```

Figure 2.- *Web scraping: Libraries and variables.*

Next, the loop is defined which will collect all the information from each requested hospital. The website is accessed, the table in which the information is located is located and, if it exists, the information is extracted from the entire table. To end the loop, each hospital data is stored in the list created earlier.

```r
# 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")
  }
}
```

Figure 3.- *Web scraping: Loop in charge of collecting information.*

Finally, a *dataframe* is created with all the data and stored in a CSV file.

```r
# 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")
```

Figure 4.- *Web scraping: Store results in CSV file.*

All of this development is in the file called *WebScrapingFinal.R*.

All the information collected will be mentioned below [3].

> REGCESS STANDARDISED CENTRE CODE (CCN): Code internal to the system of the General Register of Health Centres, Services and Establishments (REGCESS), assigned by it to each centre. This code is unambiguous and maintained over time for each centre.
>
> NOTE: This is the code intended to serve as a reference for the identification of each authorised centre in the different information systems.
>
> For each hospital, both the CNH code, with which it has been identified in the CNH, and the CCN code are offered.
>
> HISTORICALLY ASSIGNED CENTER CODE IN THE CNH (CODCNH): Refers to the 6-digit code that has been assigned to the hospital when it is first incorporated into the CNH. This code has been generated by the Ministry of Health and is maintained throughout the life of the hospital.
>
> NAME OF THE CENTRE: Name of the centre that appears in the register of the Autonomous Community, Cities with Statute of Autonomy or Ministry of Defence, which appears in its registration and authorisation and which has been transferred to REGCESS.
>
> BEDS: Beds installed on the date of data collection are considered to be those that constitute the hospital's fixed staff and that are ready to be used, although some of them may, for various reasons, not be in service on that date.
>
> TYPE OF CENTRE: Refers to any of the classes defined in the Classification of health centres, services and establishments to which it belongs, according to Annex I of Royal Decree 1277/2003 which establishes the general bases for the authorisation of health centres, services and establishments.
>
> It defines Hospitals (centres with inpatient stays) as health centres intended for the specialised and continuous care of patients in inpatient stay (at least one night), whose main purpose is the diagnosis or treatment of patients admitted to them, without prejudice to the fact that they also provide care on an outpatient basis.
>
> The School Class variable is equivalent to the Healthcare Purpose variable that has been used in previous editions of the CNH.

In order to facilitate the interpretation of the data for this variable, the following table shows the correspondence of the values for the purpose of care that have been used so far (2 columns on the left) with the current ones:

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

Figure 5.- *Purpose CNH, Health Centers.*

FUNCTIONAL DEPENDENCE: Functional dependence of a centre is understood to be the body or legal entity on which it depends, i.e. the natural or legal person that exercises the most immediate hierarchical or functional dominance or jurisdiction over the health establishment, regardless of its form of management.

Both the definition and the current values correspond to those defined in Order SCO/3866/2007, of 18 December, which establishes the content and structure of the General Register of health centres, services and establishments of the Ministry of Health and Consumer Affairs.

The following table shows the correspondences between the values that were previously used (two columns on the left) and the current ones:

Dependencia Funcional CNH (Hasta CNH_2019)		Dependencia Funcional REGCESS	
Código	Literal	Código	Literal
1	Instituto de Gestión Sanitaria-Ingesa	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

Figure 6.- *CNH Functional Unit, Health Centers.*

TEACHING ACCREDITATION: It informs that the centre has the teaching accreditation that qualifies it to provide specialised postgraduate health training and that at the time of data collection it had professionals in training.

SNS AGREEMENT: Informs if a private dependency center provides services to the Public Health Administration, regardless of whether the agreement is partial or substitute.

HIGH TECHNOLOGY. In each of the sections, the number of computers in operation in the centre is included, whether or not they are owned by the centre, and regardless of whether they are managed by companies or individuals outside the centre.

Patient Data

The next information acquired is the patient's personal information. To this end, a website for generating data from random users has been accessed that allows information to be collected via API [4].

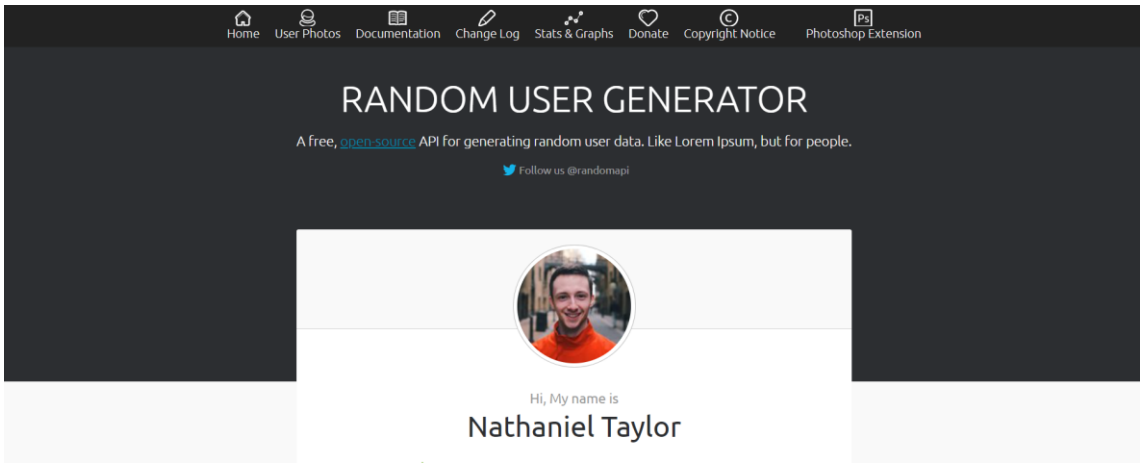

Figure 7.- *Random User Generator.*

To perform this functionality, a code in Python has been developed. This starts by importing the necessary libraries, entering the required url, and making a GET request to the API. A GET request is a type of HTTP (Hypertext Transfer Protocol) request that is used to retrieve data from a specific resource on a server. In simple terms, it is a request to obtain information from a resource on a web server.

```
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)
```

Figure 8.- *API Procedure: Import libraries and define variables.*

This development is developed in the file *ExtractionDeDatosPatients.ipynb*.

The url entered is:

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

When accessing the API, certain filters were applied to obtain limited data:

- results=918: Requests a total of 918 results (random users).
- nat=ES: Filter by Spanish nationality (ES).
- noseed: Requests random results without using a specific seed.

- format=gender,registered,location,phone: Defines the format of the response that includes information about gender, date of registration, location, and phone number.
- gender=both: Includes users of both genders (male and female).
- age=20-70: Filter by users between the ages of 20 and 70.

Once accessed, the following structure is used:

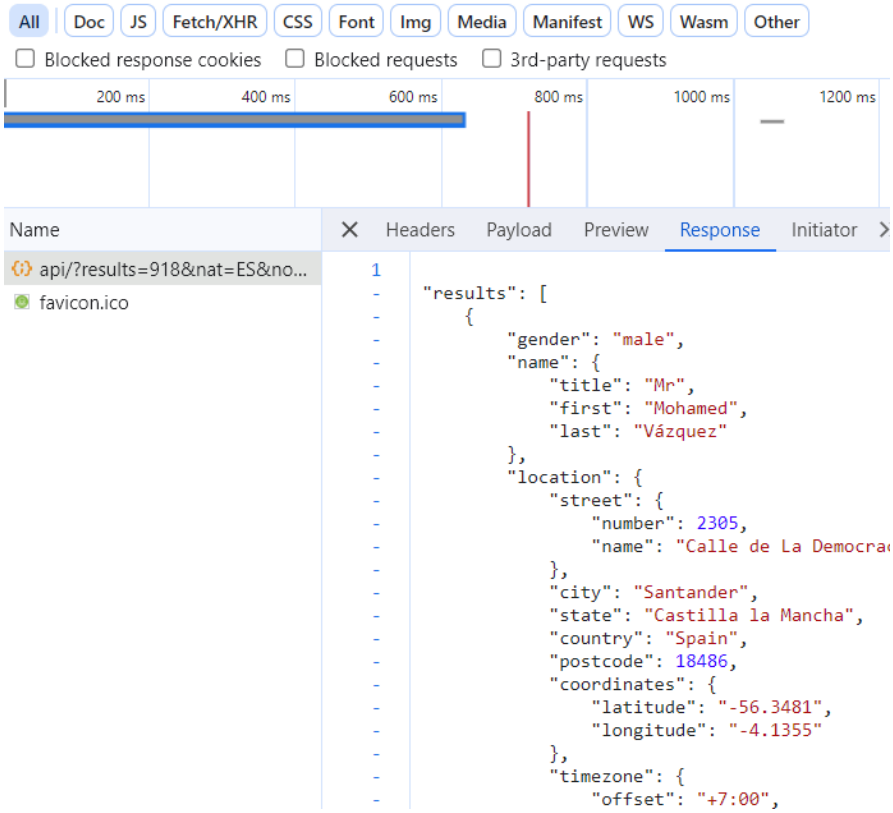

Figure 9.- *HTML structure that contains the information.*

The next step is to check if it can be accessed, status = 200, and if so, a CSV file is generated and the name of each of the variables is defined.

```python
# 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()
```

Figure 10.- *API Procedure: Generate CSV file and name the variables.*

Next, the necessary information is stored based on the structure previously proposed. In the event of an error, a message is printed on the screen.

```python
        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)
```

Figure 11.- *API procedure: Store information in the CSV file.*

Additionally, and to avoid problems with rare characters, the type of encoding stored has been confirmed.

```python
# 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","
st":"Ruiz"},"location":{"street":{"number":6756,"name":"Calle Mota"},"cit
y":"Torrejón de Ardoz","state":"Ceuta","country":"Spain","postcode":9211
1,"coordinates":{"latitude":"64.6681","longitude":"84.7161"},"timezone":{
ffset":"+10:00","description":"Eastern Australia, Guam, Vladivostok"}},"e
il":"milagros.ruiz@example.com","login":{"uuid":"ad7faed6-ecd5-4650-aab1-
ab8e1ccba3","username":"bigpeacock980","password":"randolph","salt":"UYuz
zN","md5":"87f89d6432caace927975a0f085ea649","sha1":"c93f29158e6e7a2ef93a
b7051cddc034e8ecd9","sha256":"be839bdc05f1d462eea21f1f935a5f038feef6a56c9
dbaae9452eab178eae6"},"dob":{"date":"1982-02-27T14:47:36.236Z","age":41},
```

Figure 12.- *API Procedure: First Format Acknowledgment.*

```python
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', 'regi
age': '17', 'location-country': 'Spain', 'location-state': 'Asturias',
ion-city': 'Lugo', 'location-street-name': 'Calle de Téllez', 'locatio
t-number': '7523', 'location-postcode': '10032', 'email': 'gregorio.ra
xample.com', 'phone': '937-375-703'}
{'id-value': '01371896-E', 'name-first': 'Miguel', 'name-last': 'Moral
ender': 'male', 'registered-date': '2012-08-07T01:06:32.736Z', 'regist
```

Figure 13.- *API Procedure: Second Format Acknowledgment.*

Clinical data: Heart failure

Finally, a *dataset* of heart failures has been accessed. This has been downloaded directly in CSV format [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								

Figure 14.- *CSV file with heart failure data.*

It is interesting to mention that it was created by combining different datasets that were already available independently, but had not been previously combined. In this dataset, five datasets related to heart disease have been merged across 11 common characteristics [2]. The five datasets used for its creation are:

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

The information collected refers to the following attributes [2]:

- Age: Age of the patient [years]
- Sex: Patient's sex [M: Male, F: Female]
- Type of Chest Pain: Type of chest pain [AT: Typical Angina, ATA: Atypical Angina, NAP: Non-Anginal Pain, ASY: Asymptomatic]
- Resting Blood Pressure: Resting blood pressure [mm Hg]
- Cholesterol: Serum cholesterol [mm/dl]

- Fasting Blood Glucose: Fasting Blood Glucose [1: if Fasting Blood Glucose > 120 mg/dl, 0: Otherwise]
- Resting Electrocardiogram: Resting electrocardiogram results [Normal: Normal, ST: with ST-T wave abnormality (T-wave inversions and/or elevation or depression of > 0.05 mV), LVH: showing probable or definitive left ventricular hypertrophy according to Estes' criteria]
- Maximum Heart Rate: Maximum heart rate achieved [Numerical value between 60 and 202]
- Exercise-Induced Angina: Exercise-Induced Angina [Y: Yes, N: No]
- Old Depression: Old Depression = ST [Numerical value measured in depression]
- ST Wave Slope at Maximum Exercise: The slope of the ST segment at the peak of the exercise [Top: ascending, Flat: flat, Bottom: descending]
- Heart Disease: Exit Class [1: Heart Disease, 0: Normal]

Data preprocessing and variable selection

Hospital data

Manually, the longitude and latitude of the location has been added for a more accurate display in the future and the accents have been removed to avoid formatting errors again. This can be seen in the *CleanupOfCharacters.ipynb file*.

For the removal of accents, the *unicodedata library has been used*.

```
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 funcion
quitar_tildes_en_csv('datos_hospitales_completo.csv', 'datos_hospitales_final.csv')
```

Figure 15.- *Character cleanup: Removal of accents.*

Next, an ETL process has been applied, called *ETLHospitales.ktr*, which has the following structure:

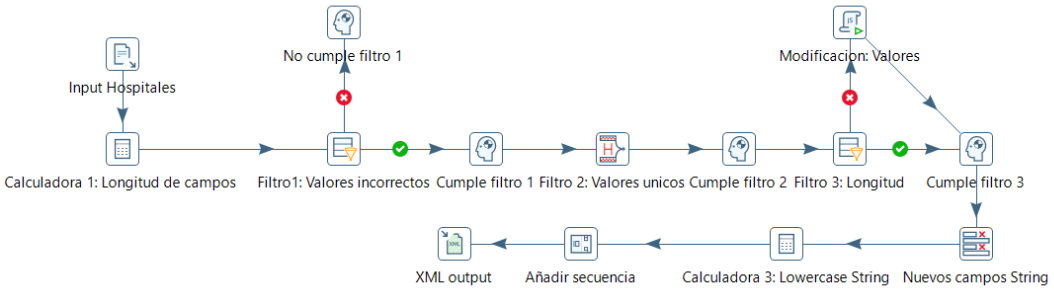

Figure 16.- *Hospital ETL process.*

It contains:

- Format filtering of: CCN Code, CNH Code, zip code and number of beds.

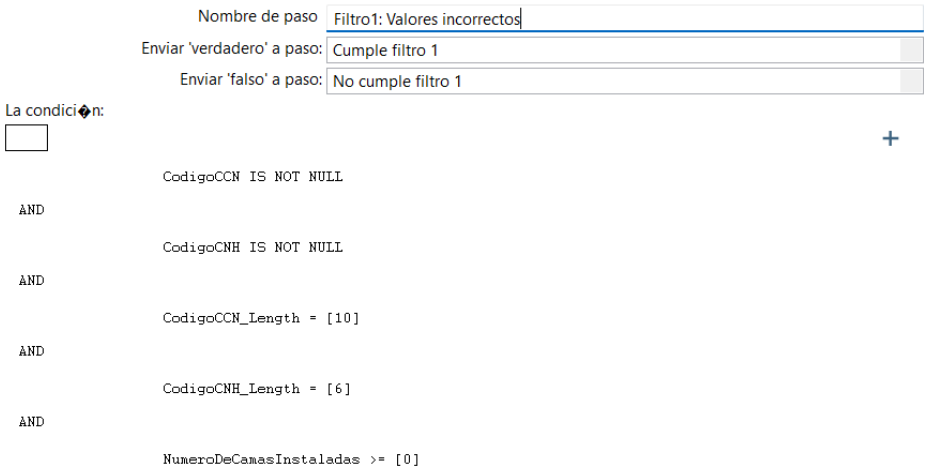

Figure 17.- *Hospital ETL Process: Format Filtering.*

- CCN Code and CNH Code are unique values.
- Correction of formatting in postal codes, with the leading zeros being omitted.

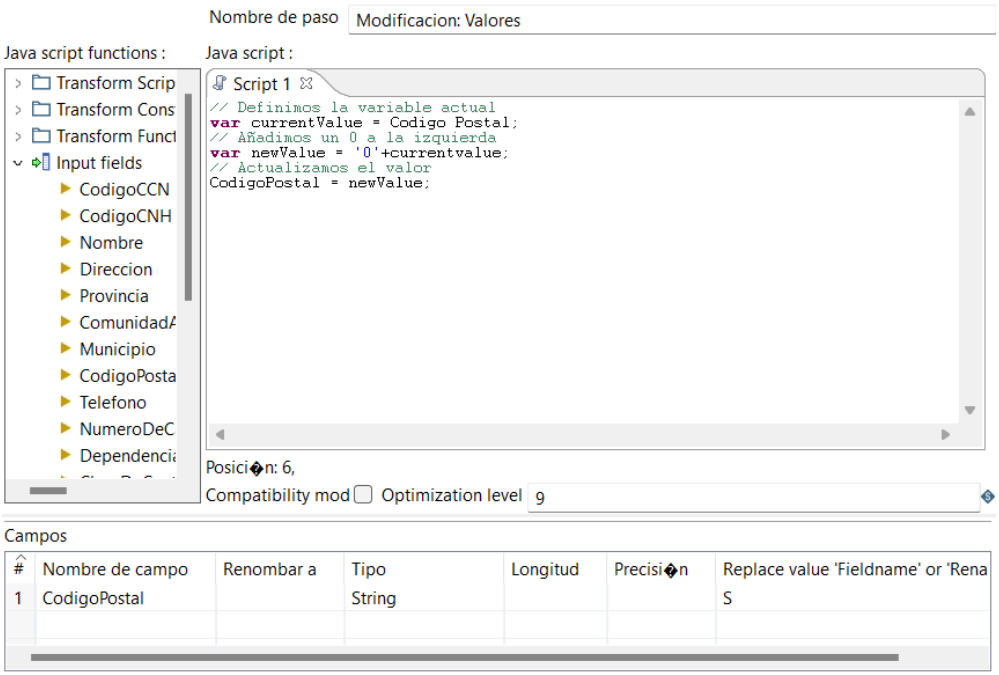

Figure 18.- *Hospital ETL Process: Correction of the format of postal codes using java script.*

- Lowercase String *typology fields* to unify texts.

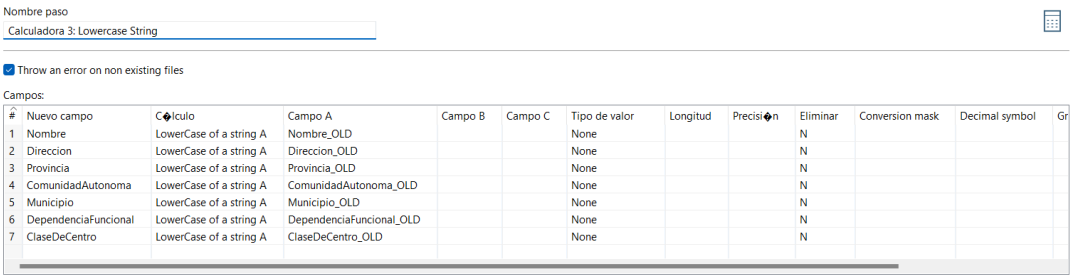

Figure 19.- *Hospital ETL Process: Unify string, lowercase.*

- IDHospital added as a primary key.

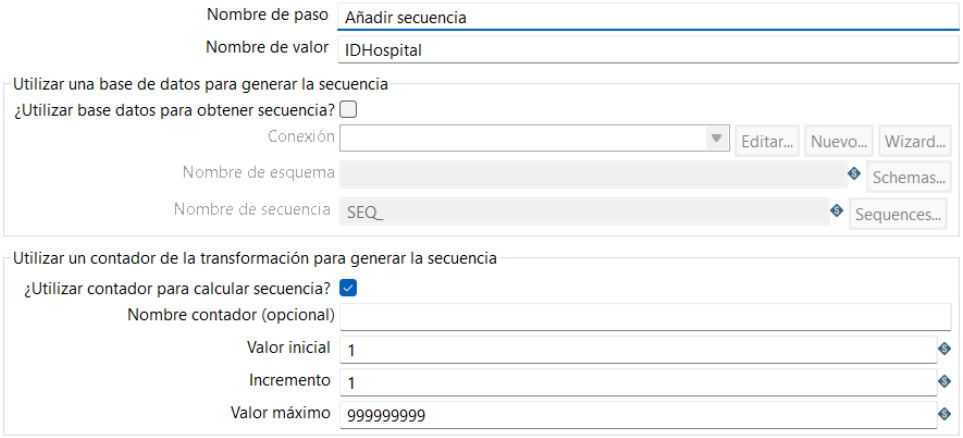

Figure 20.- *Hospital ETL Process: Add IDHospital as the primary key.*

Finally, the generated file is called *DatosHospitales.xml*.

Patient data

In this case, the accents have been eliminated to avoid UTF8 format errors as in hospital data, and the Patient-Hospital link has been made randomly. To carry out the second step, the enumeration proposed in the hospital data has been taken into account.

```
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)
```

Figure 21.- *Patient-hospital linkage.*

This link is shown in the *PatientLinkingHospital.ipynb file* shown in the annex.

Subsequently, the ETL process called *ETLPacientes.ktr* has been applied, which is shown below.

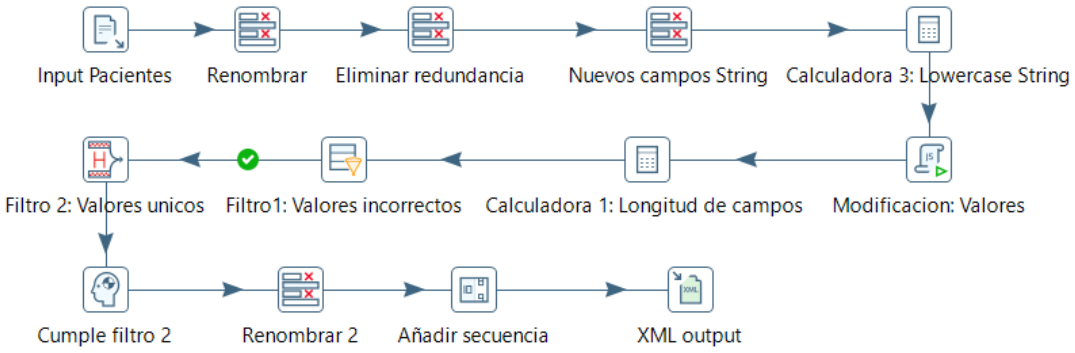

Figure 22.- *Patient ETL process.*

It contains:

- Renaming columns and eliminating field redundancy. Field deletion is caused by their appearance in another data source or by redundancy of related fields.

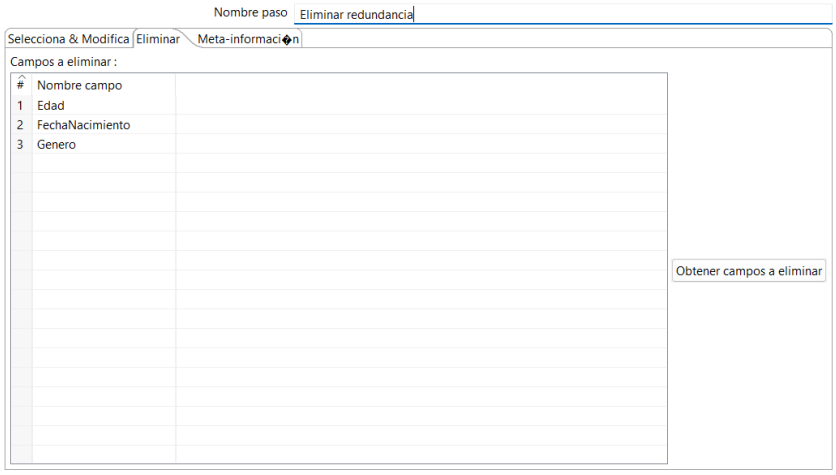

Figure 23.- *Patient ETL process: Elimination of variables that cause redundancy.*

- Lowercase String *typology fields* to unify texts.
- Delete characters within *String*.

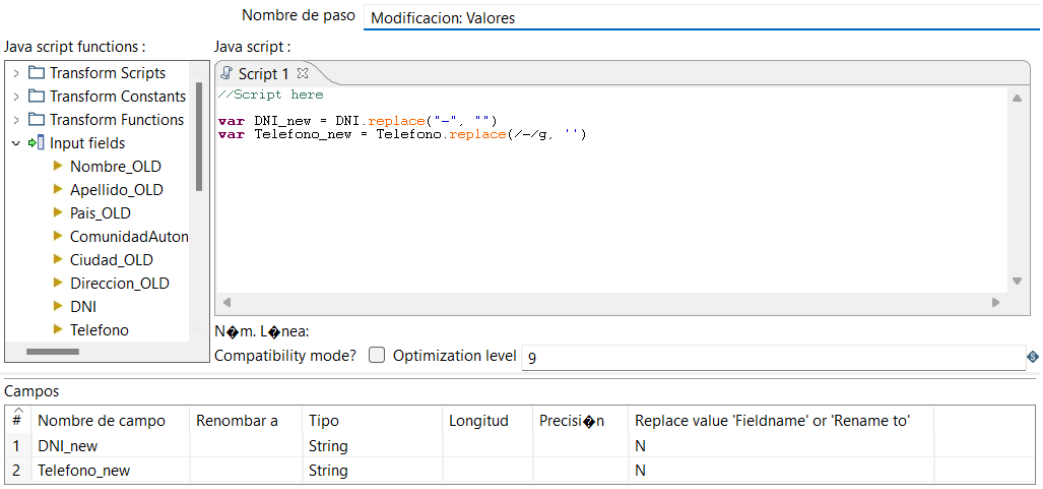

Figure 24.- *Patient ETL process: Removal of string characters.*

- Unique value and formatting filter.
- Add Parent ID as a primary key.

Similarly, a final file called DatosPacientes.xml is generated.

Clinical data: Heart failure

Firstly, the Patient-HeartFailure link has been made between the Heartfailure CSV file and the XML generated previously using the following code.

```python
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.")
```

Figure 25.- *Patient-heart failure linkage.*

This development is implemented in the *LinkingCancerPatient.ipynb file*.

This data is formatted correctly and only the name of the columns has been renamed to maintain the uniformity of the Spanish language. It can be found in the archive is *ETLCancer.ktr* from the annex.

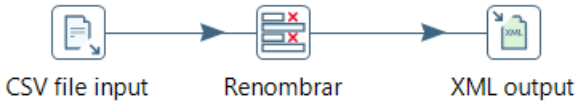

Figure 26.- *ETL process heart failures.*

As in the previous cases, a file called DatosCancer.xml has been obtained.

Big data strategy

The structure to be developed is based on Inmon*'s data warehouse*, i.e. a *top-down* design is proposed. A centralized architecture is sought that can serve as a single source and from which other development projects for different clinical problems can be based. The objective is to

consolidate the data from the various sources mentioned, and more can be added if necessary. That is why possible redundancies have been eliminated, improving the consistency of the information stored. In this way, a great capacity for scalability is obtained to handle growing volumes of data and needs that may be changed.

In short, a scalable structure is created that can be adapted to various approaches as needed. This focuses on clinical data related to heart failure and presents two developments: *dashboard* and predictive model with web application.

Figure 27.- *Big data strategy.*

In the project, the part of adding to the *datawarehouse* as well as the development of the *data marts* will be omitted, since it is not among the objectives of the project. Consequently, and as previously shown, the data will arrive in the XML format totally optimal to be stored, but instead, it will be used directly.

Applying machine learning models

Based on the data collected and the possible solutions, positive (1) and negative (0), several models suitable for binary classification have been taken into account: Logistic Regression, Decision Trees and Support Vector Machines. The goal of having multiple options is to see which one brings results with better performance. All this development is collected in its entirety in the *Modelling.ipynb file*.

We have started by defining the libraries that will be used in the first instance, establishing the current directory and selecting the file with which we are going to work.

```
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

Figure 28.- *Predictive model: Selection of libraries and data source.*

Below is the typology and description of all the existing data.

```
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

Figure 29.- *Predictive model: Analysis of the variables present.*

Data pre-processing

The following processes have been carried out:

Check that there are no 0 values that imply complete deletion of the row because it is inconsistent data.

```
# 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
```

Figure 30.- *Predictive model: Null value check.*

Study of categorical variables for subsequent treatment.

```
# 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
```

Figure 31.- *Predictive model: Analysis of categorical variables.*

Next, the categorical variables are treated. To do this, two possible treatments will be followed.

1. The first is by using *Label Ecoding*, which is useful when you have ordinal categorical variables, i.e. categories with an inherent order. This method is efficient and useful when the order of categories matters for the model, especially in algorithms based on decision trees.

```
# 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)
```

Figure 32.- *Predictive model: Treatment of categorical variables with LabelEncoder().*

2. The second is to binarize the data using *get_dummies*, a format that can be used for linear models and vector support machines, among others. This method is appropriate when the categories do not have an intrinsic order and are nominally distinct. Although it is true that it generates a large number of columns, it has been chosen to have a greater variety of results and consequently, a variety of yields.

```
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 × 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  0.183673       0.65     0.469320        0.0  0.267606  0.295455
```

Figure 33.- *Predictive model: Treatment of categorical variables with get_dummies().*

Logistic regression

Both for this model and for the subsequent ones that are going to be developed, a first modeling will be carried out where both the case with default values offered by the model will be

contemplated, as well as the case with hyperparameter analysis looking for the most optimal values for the performance of the model.

First, the model is exposed with default values or basic model. The steps followed with the following:

1. Import Libraries
2. Separation of values: Predictor variables and target variables.
3. Splitting the dataset: Training and testing.
4. Adjustment of the model, with predetermined parameters, using the training data.
5. Display of the parameters used.
6. Obtaining results
7. Performance visualization of both data sets.

```
#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))
```

Figure 34.- *Predictive model: Logistic regression, basic model.*

The results obtained are as follows:

```
{'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
```

Figure 35.- *Predictive model: Logistic regression result, basic model.*

gridsearchCV is then applied to perform hyperparameter tuning. Due to non-convergence errors, it has been decided to increase the maximum number of iterations to 10000. The hyperparameter values analyzed are:

- 'penalty': ['l1', 'l2', 'elasticnet']
- 'C': [0.001, 0.01, 0.1, 1, 10, 100]
- 'Solver': ['Newton-CG', 'LBFGS', 'Liblinear', 'SAG', 'SAGA']

```python
# 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))
```

Figure 36.- *Predictive model: Logistic regression, model with hyperparameter analysis.*

For the analysis, the largest possible set of options has been taken into account. This means that there may be incompatible combinations that cannot be analyzed. The results obtained are as follows.

```
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
```

Figure 37.- *Predictive model: Logistic regression result, model with hyperparameter analysis.*

Additionally, incompatible combinations can be checked using the following *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]
```

Figure 38.- *Predictive model: Logistic regression incompatibilities, model with hyperparameter analysis.*

Stand Vector Machines

As in the previous case, you start by importing the library to be used and reuse the two previously defined datasets. This is done because they have the same treatment of categorical variables.

```
# 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))
```

Figure 39.- *Predictive model: Support vector machines, basic model.*

Optimal values for the use of the predictive model were again obtained.

```
{'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
```

Figure 40.- *Predictive model: Result of support vector machines, basic model.*

For the case with hyperparameter adjustment, the following values have been taken into account:

- '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]

In addition, we have opted for a cross-validation dividend the data into 5 different sets 'cv=5' and a classification based on accuracy '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))
```

Figure 41.- *Predictive model: Support vector machines, model with hyperparameter analysis.*

The following result was obtained.

```
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
```

Figure 42.- *Predictive model: Result of support vector machines, model with hyperparameter analysis.*

Decision Trees

Again, you start by importing the libraries to be used. In contrast, categorical variables will be treated as the first case of those raised, using *Label Ecoding*. It is then divided into two sets of data and the same procedure is performed.

```
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']

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

Figure 43.- *Predictive model: Decision trees, basic model.*

In this case, the *entropy* criterion will be taken and no depth or any other parameter will be imposed.

```
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_DecisionTree=DecisionTree_train.predict(x_tree_train)
y_test_pred_DecisionTree = 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_DecisionTree))
print('Exactitud: ', precision_score(y_tree_train,y_train_pred_DecisionTree))
print('Exhaustividad: ', recall_score(y_tree_train,y_train_pred_DecisionTree))

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

```
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
```

Figure 44.- *Predictive model: Result of decision trees, basic model.*

Obtaining such good results makes us suspect that the model is overadjusted. Therefore, hyperparameter tuning will be performed directly with *gridsearchCV.* The values that are analyzed are the following:

- '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]

```python
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))
```

Figure 45.- *Predictive model: Decision trees, model with hyperparameter analysis.*

The result is the following.

```
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
```

Figure 46.- *Predictive model: Result of decision trees, model with hyperparameter analysis.*

Conclusion

By averaging the results obtained, the following is obtained:

```
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
```

Figure 47.- *Conclusions of the predictive models developed.*

Regarding logistic regression, very similar values are found, but a slight improvement is observed in the results of the model with hyperparameter analysis. This may be due to the difference in values found in the 'C' and 'solver' hyperparameters.

Regarding the support vector machines, differences are found in the values of 'C' and 'gamma'. As in the previous case, very similar results are shown. Due to the slight improvement, and the hyperparameter comparison carried out, the analyzed one was selected as the best option.

In the case of the decision tree, the results with overfittings have not been taken into account, selecting the values obtained after the hyperparameter analysis as optimal.

To conclude, although excellent and similar performances have been obtained, a slight improvement is perceived in those who have undergone hyperparameter analysis, which is coherent given that numerous possible combinations are analyzed and is complemented by cross-validation. Therefore, it is decided to work with the Support Vector Machine model that has been subjected to hyperparameter analysis to make predictions.

Web application

Additionally, from the selected model, a web application has been developed that allows the user to enter the data of the variables and obtain the prediction (1 or 0). To do this, we start by importing all the libraries to be used and including the practical development carried out in the modeling.

```
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)
```

Figure 48.- *Web application: Libraries and predictive model.*

Next, a function is created to collect the values that the user will enter into the calculator and will be applied to the model already obtained. The data entered will be included in all the data already recorded and the categorization and normalization will be carried out with them. This step will allow the transformation of the data entered by the user into normalized and categorized data ready to be used by the model. Once applied, the information entered by the user is applied to the model.

```
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)}")
```

Figure 49.- *Web application: Categorization and normalization of the values entered by the user.*

Finally, the design of the pop-up window with which the user will work is done.

```
# 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)
```

Figure 50.- *Application pop-up design (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()
```

Figure 51.- *Application pop-up design (II).*

The result is the following representation.

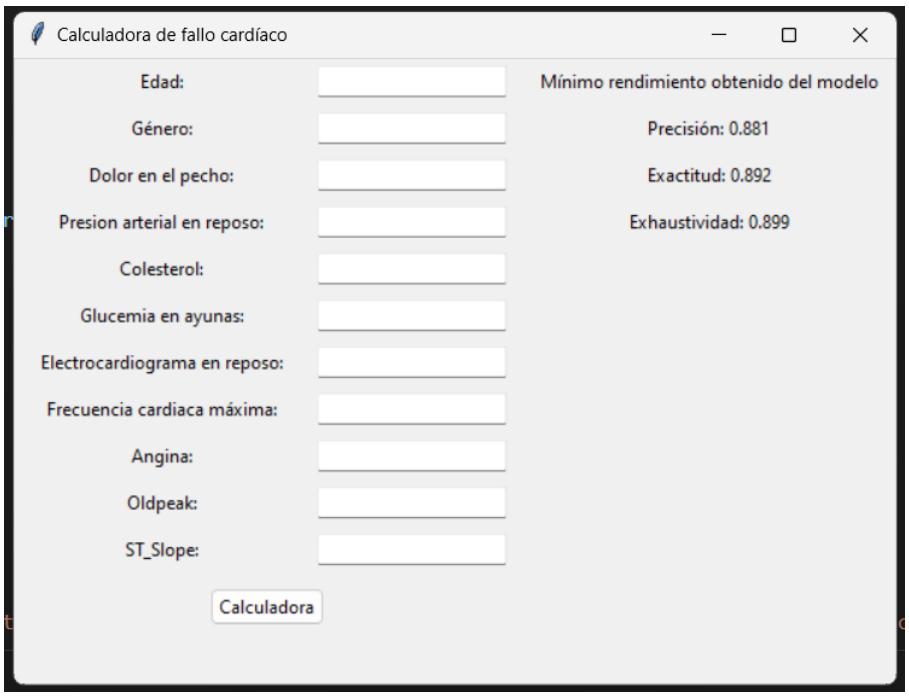

Figure 52.- *Web application interface.*

The development of this application is available in file *AplicacionLaptop.py* of the annex.

Dasboard

Since the part of entering the data into the databases was omitted, the generated XML files are used directly for the elaboration of the *dasboard*. Subsequently, some format modifications and improvements have been made for better visualization. Specifically:

- Change of variable typology.
- Replacement of variables.
- Add counter.
- Add custom columns.
- Renaming columns.

The results are shown below, and appear in the *Dashboard.pbix file*.

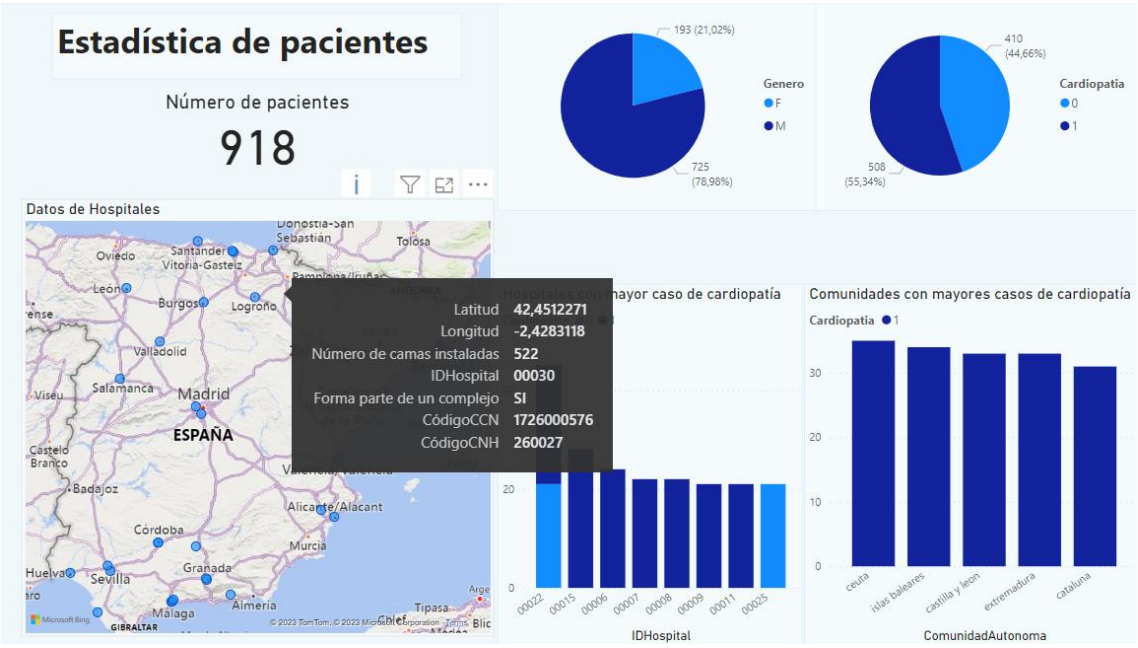

Figure 53.- *Dashboard: First sheet.*

First sheet, showing all the hospitals used in the study. Positioning yourself over each one, a pop-up window appears with detailed information, providing a complete view of the hospital network involved. This interactive feature allows easy access to key details of each medical center, such as its location, capacity, and relevant specialties.

In addition, the following crucial statistics can be observed quickly and intuitively for strategic decision-making:

- The total number of patients who participated in the study, providing an overall perspective on the size of the sample.
- The gender distribution of patients, facilitating a quick understanding of possible disparities and specific needs.
- The proportion of patients with heart disease, offering an overview of the prevalence of the disease in the population studied.
- The hospitals where the highest number of positive cases have been treated, highlighting those centers with a significant burden of patients with heart disease.
- The communities where the highest number of cases are found, identifying geographical areas with a higher incidence of heart disease.

These key indicators provide a solid basis for the efficient allocation of funds for heart disease treatment. The information presented in this first sheet is intended as a valuable resource for senior leadership, providing a comprehensive and visually accessible perspective that can effectively inform strategic decisions and resource planning in the cardiovascular health arena.

Figure 54.- *Dashboard: Second sheet.*

The second sheet, designed specifically for workers who use clinical data on a daily basis, has been conceived as a comprehensive tool that provides efficient access to all clinical and personal

patient data. This sheet is designed to facilitate the daily work of health professionals, allowing them to carry out simple searches and obtain information quickly and accurately.

The interface of the second sheet has been structured in an intuitive way, with a prominent filter that allows you to search for patients by their identification (ID). When entering the corresponding ID, the sheet responds instantly showing all the clinical and personal data associated with the patient in question. This feature significantly streamlines access to specific information, providing workers with the ability to retrieve relevant data immediately.

In addition, a clean and organized design has been incorporated that makes it easy to visualize detailed data without generating confusion. This usability-focused approach is intended to improve the efficiency and productivity of those professionals who rely on clinical data on a daily basis for diagnosis, treatment, and patient follow-up.

In conclusion, the second sheet of the *dashboard* is presented as an essential tool for medical staff, providing quick and simplified access to clinical and personal patient information, with a particular emphasis on the ability to search by ID to ensure operational efficiency in the day-to-day handling of clinical data.

Conclusions

Based on the project carried out, the following conclusions can be derived:

- The project has achieved the main objective of providing utility data from different sources. Specifically, a data warehouse design structure has been defined, a dashboard has been developed and an optimal predictive model has been obtained with a web application.
- Both the tools used and the methods have been efficient for data processing and analysis.
- The inclusion of data from different sources has enriched the heterogeneity of the dataset.
- Extensive data preprocessing has been performed, including cleaning, normalizing, and linking data from different sources.
- Various machine learning models with different configurations have been applied, and support vector machines have been selected as the final model due to the comparative results and performance improvement.
- A dashboard has been implemented that offers interactive visualizations for healthcare professionals, and a web application that allows predictions to be obtained based on the developed model.
- Adjustments have been proposed for future improvements in terms of scalability, cloud services, container technologies, and various *frameworks*.

In summary, the project has comprehensively addressed data management in the healthcare field, from acquisition and processing to predictive modeling and the creation of interactive tools for professionals. The conclusions highlight the achievement of the proposed objectives, the correct selection of tools and methods, as well as the attention to the quality and diversity of the data.

References

1- World Health Organization. Collected on November 12, 2023, from https://www.who.int/es/health-topics/cardiovascular-diseases#tab=tab_1

2- Soriano, Federico. Dataset for the prediction of heart failure. Collected on November 16, 2023, https://www.kaggle.com/datasets/fedesoriano/heart-failure-prediction/data

3- Ministry of Health. Centres and Services of the National Health System. Retrieved on November 18, 2023, from https://www.sanidad.gob.es/ciudadanos/centros.do?metodo=realizarDetalle&tipo=hospital&numero=110162

4- Hunt, Arron and Armstrong, Keith. Open source API for the generation of random users, Random User Generator. Collected on November 18, 2023, from https://randomuser.me/