Universidad de los Andes Colombia — Acreditación institucional de alta calidad 10 años	**Ingeniería de Sistemas y Computación** ISIS-3301 – Inteligencia de Negocios Primer parcial	ABET Engineering Accreditation Commission

Código de honor

“Al entregar la solución de este parcial, yo, con código me comprometo a no conversar durante el desarrollo de este examen con ninguna persona que no sea el profesor del curso, sobre aspectos relacionados con el parcial; tampoco utilizaré algún medio de comunicación por voz, texto o intercambio de archivos, para consultar o compartir con otros, información sobre el tema del parcial. Soy consciente y acepto las consecuencias que acarreará para mi desempeño académico cometer fraude en este parcial”.

Contexto

El Hospital de los Alpes como entidad de salud es consciente de la responsabilidad que tiene en la atención a sus usuarios y es por ello, que busca formas de adaptar sus servicios a las características de los usuarios, principalmente de los pacientes que llegan a sus instalaciones. Es en este punto donde ha visto la oportunidad de analizar información que actualmente posee de los pacientes y ver si una solución basada en analítica con el uso de aprendizaje automático puede ser apropiada para **clasificar** a los pacientes que llegan al hospital.

Parte 1: Definicion del enfoque analítico (10%)

Con base en el contexto de la empresa, los datos suministrados y el objetivo establecido se sugiere abordar este problema por medio de un modelo de aprendizaje automatico con las siguientes caracteristicas:

- Tipo de aprendizaje: Supervisado
- Tarea de aprendizaje: Clasificacion
- Algoritmo de aprendizaje: Arbol de decision

  Se recomienda el aprendizaje supervisado porque en este contexto, se tiene acceso a un conjunto de datos etiquetado en el que se conoce la variable objetivo, que en este caso es "enfermedad_cardiaca" (1 para sí, 0 para no). Dado que el hospital quiere principalmente buscar formas de adaptar sus servicios a las características de los usuarios es un problema de clasificacion, ya que se quiere predecir la variable objetivo que es categorica. Por ultimo, se recomienda el algoritmo de arbol de decision porque es un algoritmo que se puede utilizar para clasificacion y es fácil de interpretar.

Librerias

In [1]:
```
# Librerías para manejo de datos
import pandas as pd
pd.set_option('display.max_columns', 25) # Número máximo de columnas a mostrar
pd.set_option('display.max_rows', 50) # Numero máximo de filas a mostar
import numpy as np
np.random.seed(1234)
from sklearn.metrics import accuracy_score
import pandas as pd
# Para preparar los datos
from sklearn.preprocessing import LabelEncoder
# Para crear el arbol de decisión
from sklearn.tree import DecisionTreeClassifier
# Para usar KNN como clasificador
from sklearn.neighbors import KNeighborsClassifier
# Para realizar la separación del conjunto de aprendizaje en entrenamiento y te
from sklearn.model_selection import train_test_split
# Para evaluar el modelo
from sklearn.metrics import confusion_matrix, classification_report, precision_
# Para búsqueda de hiperparámetros
from sklearn.model_selection import GridSearchCV
# Para la validación cruzada
from sklearn.model_selection import KFold
#Librerías para la visualización
import matplotlib.pyplot as plt
# Seaborn
import seaborn as sns
from sklearn import tree
import ydata_profiling as pp
```

```
c:\Users\lgome\.conda\envs\bi\lib\site-packages\numba\core\decorators.py:262:
NumbaDeprecationWarning: numba.generated_jit is deprecated. Please see the doc
umentation at: https://numba.readthedocs.io/en/stable/reference/deprecation.ht
ml#deprecation-of-generated-jit for more information and advice on a suitable
replacement.
  warnings.warn(msg, NumbaDeprecationWarning)
C:\Users\lgome\AppData\Roaming\Python\Python38\site-packages\visions\backends
\shared\nan_handling.py:51: NumbaDeprecationWarning: The 'nopython' keyword ar
gument was not supplied to the 'numba.jit' decorator. The implicit default val
ue for this argument is currently False, but it will be changed to True in Num
ba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.ht
ml#deprecation-of-object-mode-fall-back-behaviour-when-using-jit for details.
  def hasna(x: np.ndarray) -> bool:
```

Carga de los Datos

In [2]:
```
# carga del archivo completo
dataMed = pd.read_csv('Datos_pacientes_etapa1.csv', sep=';',encoding="ISO-8859-
# carga del diccionario
diccio = pd.read_excel('Diccionario_pacientes.xlsx')
```

Como paso incial para el entendimiento de la información se realiza la lectura del diccionario.

In [3]:
```
diccio
```

Out[3]:

	Columna	Tipo	Descripción
0	Edad	numérica	> 17
1	Sexo	categórica	NaN
2	tipo_dolor_de_pecho	categórica	1,2,3,4
3	presion_sanguinea	numérica	> 50
4	colesterol	numérica	NaN
5	frecuencia_cardiaca_maxima	numérica	NaN
6	ejercicio_angina	categórica	NaN
7	glucosa_plasma	categórica	NaN
8	grosor_piel	numérica	NaN
9	insulina	numérica	NaN
10	bmi	numérica	índice de masa corporal
11	diabetes_pedigree	numérica	NaN
12	enfermedad_cardiaca	categórica	1 (SI), 0 (NO)
13	tipo_de_residencia	categórica	Urban, Rural
14	estado_de_fumador	categórica	Smoker, Non-Smoker, Unknow

Se observa que las variables dadas pueden influir en si un paciente presenta o no (binario) una enfermedad cardiaca. Se observa que las variables son de tipo numerico y categorico. Muchas de las columnas proporcionadas en los datos, como la edad, el género, las mediciones cardiovasculares y otros factores de salud, tienen una relación conocida con las enfermedades cardíacas. Utilizando estas variables como características, un modelo de aprendizaje automático, como un árbol de decisión, podría aprender a identificar patrones que indiquen si un paciente tiene una enfermedad cardíaca o no y se atienda al paciente de acuerdo a ello. Asimismo, a diferencia del análisis inicial ahora en el diccioanrio se incluye que la edad *OBLIGATORIAMENTE* debe ser superior a 17 años y que el tipo de dolor de pecho debe SER {1,2,3,4}.

Entendimiento de los datos

Antes de comenzar a explorar el dataset es importante dividirlo en dos partes: una para entrenamiento *train* y otra para pruebas *test*. Esto es importante para poder evaluar el modelo que se construya. Para ello se utiliza la función train_test_split de la libreria sklearn.model_selection. Únicamente se realizará el análisis sobre el conjunto de entrenamiento.

Subdivisión en Train y Test

Incialmente, se dividen los datos en entrenamiento y en test.

In [4]:
```
# Seleccionar variable objetivo
Y = dataMed['enfermedad_cardiaca']
# Eliminar variable objetivo de los datos
X = dataMed.drop(['enfermedad_cardiaca'], axis=1)
# Dividir los datos en entrenamiento y test
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, rando
```

Una vez hecha la subdivisión de los datos, se procede a realizar el análisis exploratorio de los datos. Para ello se utiliza la libreria pandas_profiling, la cual permite realizar un análisis exploratorio de los datos de forma rápida y sencilla en conjunto con la librería pandas que permite obtener información estadística de los datos. Inicialmente, se revisan los valores que toman las columnas en algunos registros:

In [5]:
```
# Univer temporalmente
train_data = pd.concat([X_train, Y_train], axis=1)
```

In [6]:
```
train_data.head()
```

Out[6]:

	id	edad	sexo	tipo_dolor_de_pecho	presion_sanguinea	colesterol	frecuencia_cardiac
150	3809	22	NaN	1	150	253	
922	828	49	0.0	2	130	260	
117	3097	49	0.0	4	143	137	
582	3315	44	0.0	1	101	264	
679	1557	18	1.0	2	142	132	

In [7]:
```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 800 entries, 150 to 323
Data columns (total 15 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   id                         800 non-null    int64
 1   edad                       800 non-null    int64
 2   sexo                       727 non-null    float64
 3   tipo_dolor_de_pecho        800 non-null    int64
 4   presion_sanguinea          800 non-null    int64
 5   colesterol                 800 non-null    int64
 6   frecuencia_cardiaca_maxima 800 non-null    int64
 7   ejercicio_angina           800 non-null    int64
 8   glucosa_plasma             723 non-null    float64
 9   grosor_piel                724 non-null    float64
 10  insulina                   716 non-null    float64
 11  bmi                        800 non-null    float64
 12  diabetes_pedigree          800 non-null    float64
 13  tipo_de_residencia         737 non-null    object
 14  estado_de_fumador          800 non-null    object
dtypes: float64(6), int64(7), object(2)
memory usage: 100.0+ KB
```

Se revisa el tipo de dato de cada atributo. Inicialmente, se observa que el tipo de dato de cada atributo sí concuerda con lo presentado en el diccionario de datos.

In [8]:
```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 800 entries, 150 to 323
Data columns (total 16 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   id                         800 non-null    int64  
 1   edad                       800 non-null    int64  
 2   sexo                       727 non-null    float64
 3   tipo_dolor_de_pecho        800 non-null    int64  
 4   presion_sanguinea          800 non-null    int64  
 5   colesterol                 800 non-null    int64  
 6   frecuencia_cardiaca_maxima 800 non-null    int64  
 7   ejercicio_angina           800 non-null    int64  
 8   glucosa_plasma             723 non-null    float64
 9   grosor_piel                724 non-null    float64
 10  insulina                   716 non-null    float64
 11  bmi                        800 non-null    float64
 12  diabetes_pedigree          800 non-null    float64
 13  tipo_de_residencia         737 non-null    object 
 14  estado_de_fumador          800 non-null    object 
 15  enfermedad_cardiaca        800 non-null    int64  
dtypes: float64(6), int64(8), object(2)
memory usage: 106.2+ KB
```

Hay variables que de acuerdo al registro son numéricas pero de acuerdo al diccionario de datos son categóricas. Entre ellas está: Sexo, tipo_dolor_pecho, ejercicio_angina, glucosa_plasma. Se transforman a tipo categórico en una copia del dataset para poder realizar el análisis exploratorio de los datos.

In [9]:
```
vars_to_convert = ["sexo", "tipo_dolor_de_pecho", "ejercicio_angina", "glucosa_

train_data_copy = train_data.copy()

for col in vars_to_convert:
    train_data_copy[col] = train_data_copy[col].astype('object')
train_data_copy.info()

for feature in train_data_copy.select_dtypes(include=['object']).columns:
    print(f"{feature}: {X_train[feature].unique()}")
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 800 entries, 150 to 323
Data columns (total 16 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   id                          800 non-null    int64  
 1   edad                        800 non-null    int64  
 2   sexo                        727 non-null    object 
 3   tipo_dolor_de_pecho         800 non-null    object 
 4   presion_sanguinea           800 non-null    int64  
 5   colesterol                  800 non-null    int64  
 6   frecuencia_cardiaca_maxima  800 non-null    int64  
 7   ejercicio_angina            800 non-null    object 
 8   glucosa_plasma              723 non-null    object 
 9   grosor_piel                 724 non-null    float64
 10  insulina                    716 non-null    float64
 11  bmi                         800 non-null    float64
 12  diabetes_pedigree           800 non-null    float64
 13  tipo_de_residencia          737 non-null    object 
 14  estado_de_fumador           800 non-null    object 
 15  enfermedad_cardiaca         800 non-null    int64  
dtypes: float64(4), int64(6), object(6)
memory usage: 106.2+ KB
sexo: [nan  0.  1.]
tipo_dolor_de_pecho: [1 2 4 3]
ejercicio_angina: [1 0]
glucosa_plasma: [206. 197. 178. 193. 185.  nan 152.  87. 209. 128. 123. 165. 2
10. 218.
 232. 208. 175. 238. 215. 119.  72. 141. 109. 231. 164. 129. 122. 145.
 161. 223. 171. 158. 106.  86. 191. 137. 221. 220. 117. 198. 204.  71.
 173. 181. 176. 102. 207. 136. 230. 187. 159. 203.  96. 113. 135. 195.
 143. 188. 242.  97. 104. 229. 224. 167. 177. 131. 211.  94. 216. 100.
  80. 213.  88. 153. 228. 169. 134. 125. 222.  78. 115. 249.  82. 139.
 162. 190. 150. 121. 205.  90. 149. 118. 112. 179. 246. 107. 235.  79.
 151. 127. 108. 194. 103.  75. 180. 146. 110.  83. 189. 192. 237. 105.
 247. 155. 183. 120. 248. 250. 202. 234. 240. 111.  73. 243.  84. 226.
 138. 142. 154.  74. 233. 186. 140. 241.  70. 239. 126. 133.  98. 212.
 101. 156.  77. 219.  81.  95. 245.  91. 166. 168. 214. 157. 236.  93.
 174. 144. 184. 200. 148. 116. 114. 217.  89.  92. 163. 147. 225.  85.
 196. 201.  76. 227. 130.  99. 160. 172. 132. 170. 244. 124. 182.]
tipo_de_residencia: ['Urban' nan 'Rural']
estado_de_fumador: ['Smoker' 'Non-Smoker' 'Unknown']
```

Se observa que las columnas categóricas de sexo, tipo de residencia, tiene valores faltantes. Se procede a imputar los valores faltantes con la media.

Se contabiliza el número de registros y de atributos.

In [10]:
```
total_rows_count = train_data.shape[0]
total_cols_count = train_data.shape[1]
numeric_cols = train_data.select_dtypes(['number']).columns
categorical_cols = train_data.select_dtypes(['object']).columns

print(f"El dataset has a total of {total_rows_count} rows and {total_cols_coun
print(f"and {len(categorical_cols)} categorical in train_data")
```

```
El dataset has a total of 800 rows and 16 columns: 14 numerical
l in train_data
```

Se muestran a continuación los estadísticos descriptivos de las variables numéricas:

In [11]:
```
num_cols_str = [num_col for num_col in numeric_cols]
train_data[num_cols_str].describe()
```

Out[11]:

	id	edad	sexo	tipo_dolor_de_pecho	presion_sanguinea	colest
count	800.000000	800.000000	727.000000	800.000000	800.000000	800.000
mean	2975.220000	52.926250	0.469051	2.531250	134.560000	208.865
std	1753.665656	21.390483	0.499385	1.117177	47.880498	54.387
min	43.000000	18.000000	0.000000	1.000000	0.000000	120.000
25%	1460.250000	34.000000	0.000000	2.000000	111.000000	162.000
50%	2913.000000	52.000000	0.000000	3.000000	137.000000	210.500
75%	4479.250000	72.000000	1.000000	4.000000	160.000000	258.000
max	5998.000000	90.000000	1.000000	4.000000	299.000000	300.000

En esta ocasión, se puede observar que la variable edad sí cumple con los requisitos básicos de que la edad debe ser superior a 17. Esto se sabe debido a la gráfica superior en la que edl valor mínimo para la edad es 18.

Se revisan las categorías y la frecuencia de ocurrencia de cada categoría para las variables categóricas:

In [12]:
```
text_cols_str = [text_col for text_col in categorical_cols]
train_data[text_cols_str].describe()
```

Out[12]:

	tipo_de_residencia	estado_de_fumador
count	737	800
unique	2	3
top	Rural	Smoker
freq	384	371

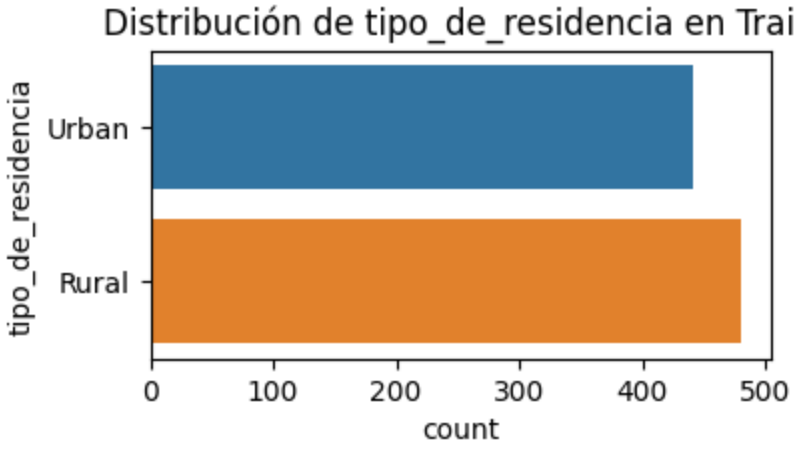
In [13]:
```
for col in text_cols_str:
    print(f"Categorías y frecuencia de la columna {col}:")
    print(train_data[col].value_counts(), "\n")
```

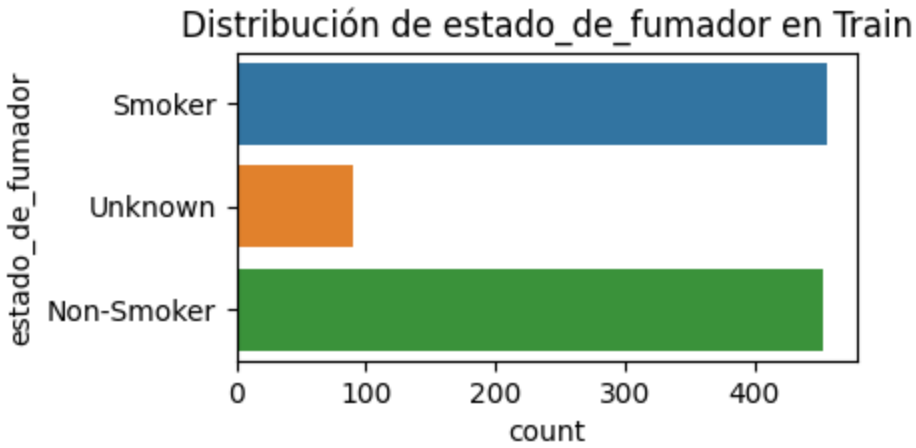
```
Categorías y frecuencia de la columna tipo_de_residencia:
tipo_de_residencia
Rural    384
Urban    353
Name: count, dtype: int64

Categorías y frecuencia de la columna estado_de_fumador:
estado_de_fumador
Smoker        371
Non-Smoker    356
Unknown        73
Name: count, dtype: int64
```

Se observa que tanto la variable de tipo_de_residencia como estado_de_fumador, también cumplen con el diccionario dado que solo tienen dos categorías (Rural, Uraban) en el caso del _tipo_de*residencia* y tres (Smoker, Non-smoker, Unknown) en el caso del _estado_de*fumador*.

In [14]:
```
for col in X_train.select_dtypes(include='object').columns:
    plt.figure(figsize=(4, 2))
    sns.countplot(y=dataMed[col])
    plt.title(f'Distribución de {col} en Train')
    plt.show()
```





Verificación de Supuestos

A continuación se determinan los supuestos de consistencia, validez, unicidad y completitud de los datos. 1) Consistencia: Eliminar los registros que presentan inconsistencia semántica en la columna de acuerdo al diccionario de datos. 2) Validez: Eliminar los registros que presentan valores atípicos en las columnas de acuerdo al análisis de los estadísticos descriptivos. 3) Unicidad: Eliminar los registros duplicados. 4) Completitud: Eliminar los registros que presentan valores nulos en las columnas. 5) Variables a utilizar: Se eliminan las variables que no se utilizarán en el modelo. Eliminar las columnas que son identificadores únicos de los datos: id.

Consistencia

Hay variables que de acuerdo al registro son numéricas pero de acuerdo al diccionario de datos son categóricas. Entre ellas está: Sexo, tipo_dolor_pecho, ejercicio_angina, glucosa_plasma. Esto se realizó en la parte superior del notebook. Para hacer el correcto análisis.

Unicidad

No se presentan duplicados en el dataset de train.

In [15]:
```
duplicados = train_data.duplicated(keep = False).sum()
print(f"Total de filas duplicadas en el dataframe: {duplicados}")
```

```
Total de filas duplicadas en el dataframe: 0
```

Completitud

Sí se presenta falta de completitud en algunas de las columnas del dataset de train. En particular, sexo, glucosa_plasma, grosor_piel, insulina, tipo de residencia. En esta ocasión, se puede tomar la decisión o de a) eliminar las columnas que presentan faltantes o de b) imputar los valores faltantes. En este caso, se decide imputar los valores faltantes con la media.

In [16]:
```
#Identificar el porcentaje de nulos por columna
print("Variables con valores nulos en el conjunto de entrenamiento Y")
print(Y_train.isnull().sum() / Y_train.shape[0])
print("Variables predictoras con valores nulos en el conjunto de entrenamiento
print(X_train.isnull().sum() / X_train.shape[0])
print("Variables con valores nulos en el conjunto de training")
print(train_data.isnull().sum() / train_data.shape[0])
```

```
Variables con valores nulos en el conjunto de entrenamiento Y
0.0
Variables predictoras con valores nulos en el conjunto de entrenamiento X
id                            0.00000
edad                          0.00000
sexo                          0.09125
tipo_dolor_de_pecho           0.00000
presion_sanguinea             0.00000
colesterol                    0.00000
frecuencia_cardiaca_maxima    0.00000
ejercicio_angina              0.00000
glucosa_plasma                0.09625
grosor_piel                   0.09500
insulina                      0.10500
bmi                           0.00000
diabetes_pedigree             0.00000
tipo_de_residencia            0.07875
estado_de_fumador             0.00000
dtype: float64
Variables con valores nulos en el conjunto de training
id                            0.00000
edad                          0.00000
sexo                          0.09125
tipo_dolor_de_pecho           0.00000
presion_sanguinea             0.00000
colesterol                    0.00000
frecuencia_cardiaca_maxima    0.00000
ejercicio_angina              0.00000
glucosa_plasma                0.09625
grosor_piel                   0.09500
insulina                      0.10500
bmi                           0.00000
diabetes_pedigree             0.00000
tipo_de_residencia            0.07875
estado_de_fumador             0.00000
enfermedad_cardiaca           0.00000
dtype: float64
```

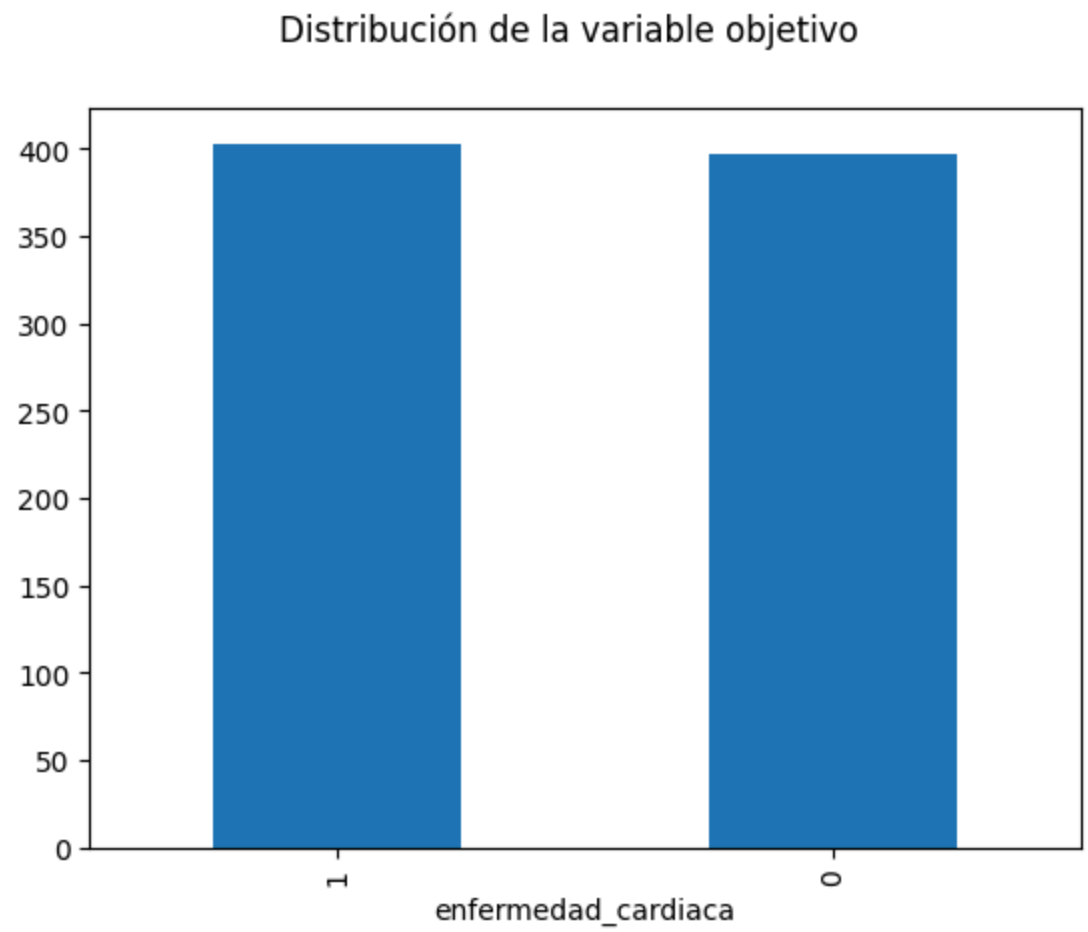
Se observa que hay valores nulos en las columnas "sexo" y "glucosa_plasma". Es posible que más adelante se decida eliminar eliminar los registros que presentan valores nulos en estas columnas o reemplazar por la media de la columna.

Distribucion de la variables y Visualización

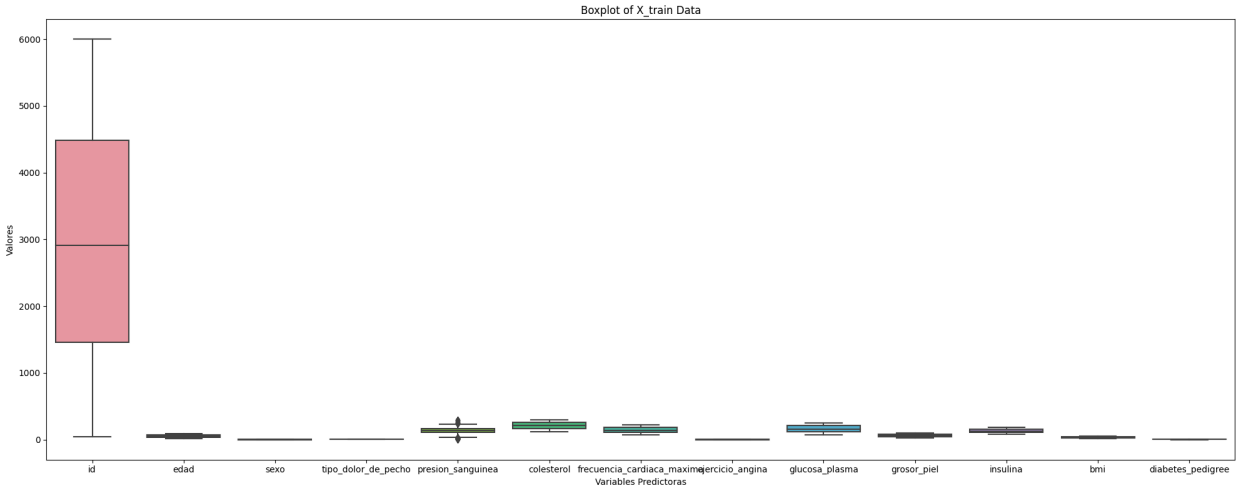
Se observa que ambas variables para predecir si un paciente sufre o no de una enfermedad cardiaca están balanceadas. Por lo tanto, el problema es un problema binario de clasificación

In [17]:
```
Y_train.value_counts().plot(kind='bar')
plt.suptitle('Distribución de la variable objetivo')
```

Out[17]:
```
Text(0.5, 0.98, 'Distribución de la variable objetivo')
```



In [18]:
```
# Grafico de caja para analizar los datos y su dispersion
figura = plt.figure(figsize=(20, 8))
ax = sns.boxplot(data=X_train, orient="v")
ax.set_title("Boxplot of X_train Data")
ax.set_xlabel("Variables Predictoras")
ax.set_ylabel("Valores")
figura.tight_layout()
```



Finalmente, se procede a realizar el informe general con Pandas Profiling. Este reporte muestra en total ocho alertas: 1) La columna de sexo presenta datos faltantes. Esto ya se había detectado en la subsección de completitud. 2) El identificador es una columna que

identifica a cada fila. Además, el bmi y diabetes_pedigree son únicos también para todos los pacientes. No se repite.

In [19]:
```
#Reporte de pandas_profiling como complemento para entender mejor los datos
profile = pp.ProfileReport(X_train, title="Pandas Profiling Report")
profile.to_notebook_iframe()
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Overview

Dataset statistics

Number of variables	15
Number of observations	800
Missing cells	373
Missing cells (%)	3.1%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	100.0 KiB
Average record size in memory	128.0 B

Variable types

Numeric	10
Categorical	5

Alerts

`sexo` has 73 (9.1%) missing values	**Missing**
`glucosa_plasma` has 77 (9.6%) missing values	**Missing**
`grosor_piel` has 76 (9.5%) missing values	**Missing**
`insulina` has 84 (10.5%) missing values	**Missing**

Parte 2: Limpieza y preparacion de los datos (30%)

Tras el entendimiento y analisis de los datos se detectaron varios problemas:

- Hay nulos

- No hay duplicados (No obstante no se sabe si en el data de test si los hay) Con el objetivo de asegurar la calidad de los datos se decidió el siguiente curso de acción:
- Reemplazar los valores nulos por la media de la columna en la que se encuentran
- Eliminar los duplicados
- Reemplazar los valores fuera de rango por la media de la columna
- Convertir las variables categoricas a numericas
- Eliminar columnas que no se utilizaran en el modelo

Se eliminan las columnas que no se utilizarán en el modelo. Eliminar las columnas que son identificadores únicos de los datos: id. Asimismo, se decide elimar el tipo de dolor del pecho dado que presenta ordinalidad. Para este análisis no se va a considerar.

In [20]:
```
cols_to_drop = ["id",'tipo_dolor_de_pecho']
dataLimpio = train_data.drop(cols_to_drop, axis=1).copy()
```

Se eliminan las filas duplicadas. Dado que no hay filas duplicadas no se espera variación en los resultados obtenidos.

In [21]:
```
# Se eliminan los datos duplicados
print(f"Data antes de eliminar duplicados: {dataLimpio.shape}")
dataLimpio = dataLimpio.drop_duplicates()
print(f"Data después de eliminar duplicados: {dataLimpio.shape}")
```

```
Data antes de eliminar duplicados: (800, 14)
Data después de eliminar duplicados: (800, 14)
```

In [22]:
```
dataLimpio.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 800 entries, 150 to 323
Data columns (total 14 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   edad                       800 non-null    int64  
 1   sexo                       727 non-null    float64
 2   presion_sanguinea          800 non-null    int64  
 3   colesterol                 800 non-null    int64  
 4   frecuencia_cardiaca_maxima 800 non-null    int64  
 5   ejercicio_angina           800 non-null    int64  
 6   glucosa_plasma             723 non-null    float64
 7   grosor_piel                724 non-null    float64
 8   insulina                   716 non-null    float64
 9   bmi                        800 non-null    float64
 10  diabetes_pedigree          800 non-null    float64
 11  tipo_de_residencia         737 non-null    object 
 12  estado_de_fumador          800 non-null    object 
 13  enfermedad_cardiaca        800 non-null    int64  
dtypes: float64(6), int64(6), object(2)
memory usage: 126.0+ KB
```

Antes de continuar, es importante recalcar que hay variables categóricas que no pueden se float dado que representan una categoría en particular como lo es el caso del sexo. Estas columnas debe estar en formato int64 y en float64.Por otro lado, es importante recalcar que

glucosa plasma presenta valores numéricos por lo que no se considerará como varibale categórica.

In [27]:
```
all_cols = dataLimpio.columns
col_categoricas = ['sexo', 'ejercicio_angina','tipo_de_residencia','estado_de_
num_cols = list(set(all_cols) - set(col_categoricas))
```

In [28]:
```
for col in col_categoricas:
    try:
        dataLimpio[col] = dataLimpio[col].astype('int64')
    except:
        continue
```

In [29]:
```
for col in col_categoricas:
    print(f"Categorías y frecuencia de la columna {col}:")
    print(train_data[col].value_counts(), "\n")
```

```
Categorías y frecuencia de la columna sexo:
sexo
0.0    386
1.0    341
Name: count, dtype: int64

Categorías y frecuencia de la columna ejercicio_angina:
ejercicio_angina
0    413
1    387
Name: count, dtype: int64

Categorías y frecuencia de la columna tipo_de_residencia:
tipo_de_residencia
Rural    384
Urban    353
Name: count, dtype: int64

Categorías y frecuencia de la columna estado_de_fumador:
estado_de_fumador
Smoker        371
Non-Smoker    356
Unknown        73
Name: count, dtype: int64
```

Reemplazo de nulos por la media de la columna en las variables numéricas y por la moda en las variables categóricas.

In [30]:
```
from sklearn.impute import SimpleImputer

# Replace missing values in numerical columns with the mean
num_imputer = SimpleImputer(strategy='mean')
dataLimpio[num_cols] = num_imputer.fit_transform(dataLimpio[num_cols])

# Replace missing values in categorical columns with the mode
cat_imputer = SimpleImputer(strategy='most_frequent')
dataLimpio[col_categoricas] = cat_imputer.fit_transform(dataLimpio[col_categor
```

In [31]:
```
col_num = dataLimpio.select_dtypes(include='number').columns
int_cols = dataLimpio.select_dtypes(include=np.int64).columns
for col in col_num:
    if col != 'enfermedad_cardiaca':
        dataLimpio[col] = dataLimpio[col].fillna(dataLimpio[col].mean())
# Se redondean los valores de las columnas enteras
dataLimpio[int_cols] = dataLimpio[int_cols].round(0).astype(np.int64)
dataLimpio
```

Out[31]:

	edad	sexo	presion_sanguinea	colesterol	frecuencia_cardiaca_maxima	ejercicio_angina
150	22.0	0.0	150.0	253.0	74.0	1
922	49.0	0.0	130.0	260.0	87.0	0
117	49.0	0.0	143.0	137.0	187.0	1
582	44.0	0.0	101.0	264.0	77.0	0
679	18.0	1.0	142.0	132.0	70.0	1
...	...	...	...	...	...	...
977	44.0	0.0	118.0	152.0	202.0	1
220	19.0	0.0	134.0	278.0	131.0	0
34	60.0	1.0	227.0	201.0	214.0	1
862	50.0	0.0	140.0	197.0	148.0	1
323	52.0	0.0	299.0	283.0	93.0	1

800 rows × 14 columns

Posteriormente, se procede a imputar los outliers. Se considera como outlier a cualquier valor que se encuentre +- 1.5 del IQR.

In [32]:
```
col_num = dataLimpio.select_dtypes(include='number').columns
for col in col_num:
    Q1 = dataLimpio[col].quantile(0.25)
    Q3 = dataLimpio[col].quantile(0.75)
    IQR = Q3 - Q1
    dataLimpio[col] = np.where(dataLimpio[col] < (Q1 - 1.5 * IQR), dataLimpio[
    dataLimpio[col] = np.where(dataLimpio[col] > (Q3 + 1.5 * IQR), dataLimpio[
```

Despues de haber corregido los problemas de los datos, unicamente restaba convertir las variables categoricas en numerica para asi asegurar un modelamiento correcto. Incialmente se genera una copia.

In [33]:
```
from sklearn.preprocessing import OneHotEncoder

def one_hot_encoder(df):
    original_df = df.copy()
    original_df.to_csv('original_data.csv', index=False)

    cat_cols = df.select_dtypes(include='object').columns

    if len(cat_cols)==0:
```

```
        numeric_columns = df.select_dtypes(include='number').copy()
        return numeric_columns, original_df


    for col in cat_cols:
        encoder = OneHotEncoder()
        transformed = encoder.fit_transform(df[[col]])
        new_column_names = [f"{col}_{col_name}" for col_name in encoder.catego
        df[new_column_names] = transformed.toarray()
        df.drop([col], axis=1, inplace=True)

    print("Completado OneHot")
    return df, original_df

dataLimpio, original_data = one_hot_encoder(dataLimpio)
dataLimpio.info()
```

```
Completado OneHot
<class 'pandas.core.frame.DataFrame'>
Index: 800 entries, 150 to 323
Data columns (total 19 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   edad                            800 non-null    float64
 1   presion_sanguinea               800 non-null    float64
 2   colesterol                      800 non-null    float64
 3   frecuencia_cardiaca_maxima      800 non-null    float64
 4   glucosa_plasma                  800 non-null    float64
 5   grosor_piel                     800 non-null    float64
 6   insulina                        800 non-null    float64
 7   bmi                             800 non-null    float64
 8   diabetes_pedigree               800 non-null    float64
 9   enfermedad_cardiaca             800 non-null    float64
 10  sexo_0.0                        800 non-null    float64
 11  sexo_1.0                        800 non-null    float64
 12  ejercicio_angina_0              800 non-null    float64
 13  ejercicio_angina_1              800 non-null    float64
 14  tipo_de_residencia_Rural        800 non-null    float64
 15  tipo_de_residencia_Urban        800 non-null    float64
 16  estado_de_fumador_Non-Smoker    800 non-null    float64
 17  estado_de_fumador_Smoker        800 non-null    float64
 18  estado_de_fumador_Unknown       800 non-null    float64
dtypes: float64(19)
memory usage: 157.3 KB
```

In [34]:
```
dataLimpio.head()
```

Out[34]:

	edad	presion_sanguinea	colesterol	frecuencia_cardiaca_maxima	glucosa_plasma	grosor
150	22.0	150.0	253.0	74.0	206.0	66.00
922	49.0	130.0	260.0	87.0	197.0	46.00
117	49.0	143.0	137.0	187.0	178.0	37.00
582	44.0	101.0	264.0	77.0	193.0	81.00
679	18.0	142.0	132.0	70.0	185.0	59.22

Se vuelve a realizar el análisis exploratorio de los datos para verificar que los problemas hayan sido corregidos.

In [35]:
```
#Perfil de los datos ya limpios y preprocesados
profile = pp.ProfileReport(dataLimpio)
profile.to_notebook_iframe()
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Overview

Dataset statistics

Number of variables	19
Number of observations	800
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	157.3 KiB
Average record size in memory	201.3 B

Variable types

Numeric	9
Categorical	10

Alerts

`sexo_0.0` is highly overall correlated with `sexo_1.0`	**High correlation**
`sexo_1.0` is highly overall correlated with `sexo_0.0`	**High correlation**
`ejercicio_angina_0` is highly overall correlated with `ejercicio_angina_1`	**High correlation**

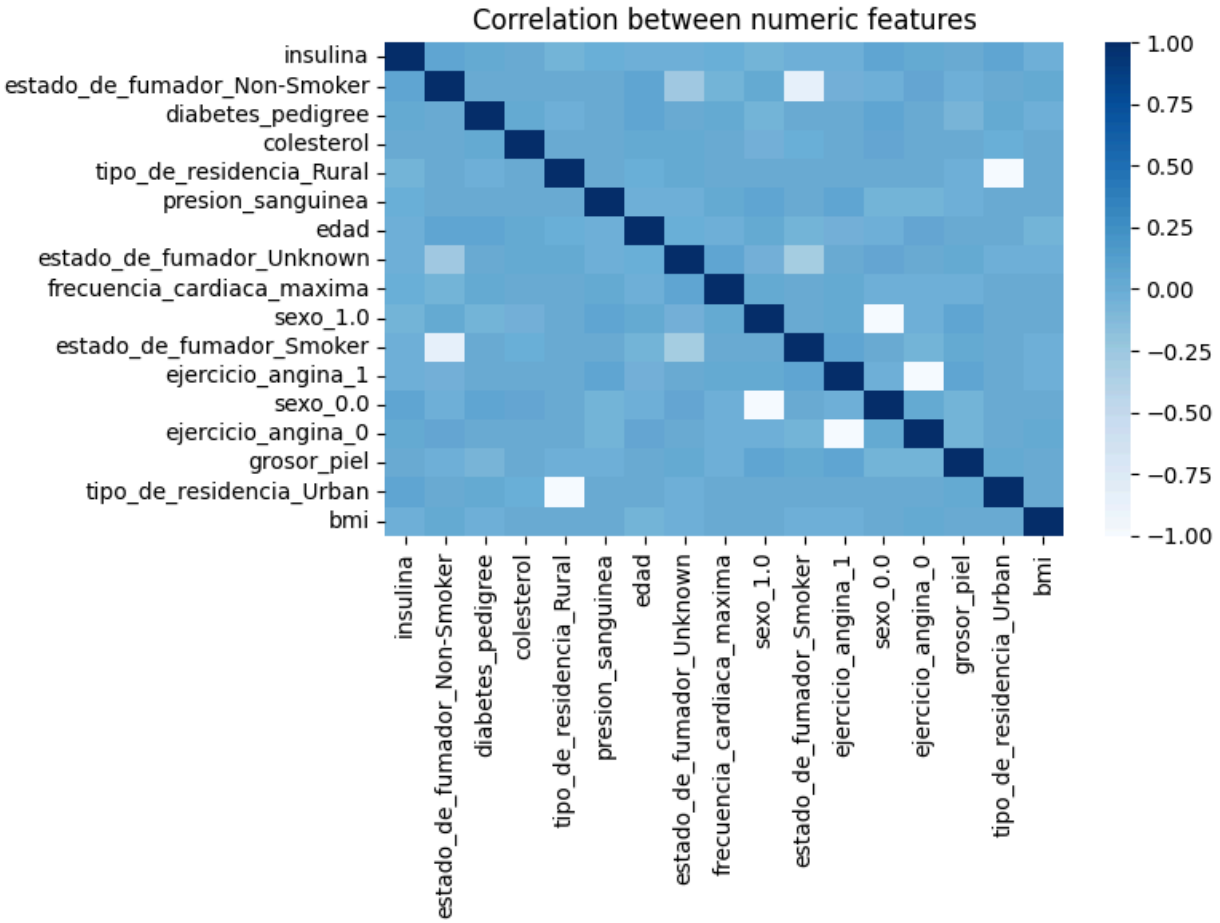
Se separa nuevamente el conjunto n las variables predictoras y la variable objetivo.

In [36]:
```
Y_train = dataLimpio['enfermedad_cardiaca']
X_train = dataLimpio.drop(['enfermedad_cardiaca'], axis=1)
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 800 entries, 150 to 323
Data columns (total 18 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   edad                            800 non-null    float64
 1   presion_sanguinea               800 non-null    float64
 2   colesterol                      800 non-null    float64
 3   frecuencia_cardiaca_maxima      800 non-null    float64
 4   glucosa_plasma                  800 non-null    float64
 5   grosor_piel                     800 non-null    float64
 6   insulina                        800 non-null    float64
 7   bmi                             800 non-null    float64
 8   diabetes_pedigree               800 non-null    float64
 9   sexo_0.0                        800 non-null    float64
 10  sexo_1.0                        800 non-null    float64
 11  ejercicio_angina_0              800 non-null    float64
 12  ejercicio_angina_1              800 non-null    float64
 13  tipo_de_residencia_Rural        800 non-null    float64
 14  tipo_de_residencia_Urban        800 non-null    float64
 15  estado_de_fumador_Non-Smoker    800 non-null    float64
 16  estado_de_fumador_Smoker        800 non-null    float64
 17  estado_de_fumador_Unknown       800 non-null    float64
dtypes: float64(18)
memory usage: 151.0 KB
```

Finalmente, se realizan estas mismas transformaciones sobre el conjunto de prueba. Esto es importante para poder evaluar el modelo que se construya de acuerdo con lo que se realice en el fit.

In [39]:
```
all_cols = X_train.columns
col_categoricas = ['sexo', 'tipo_dolor_de_pecho', 'ejercicio_angina', 'glucosa_
num_cols = list(set(all_cols) - set(col_categoricas))
# Verificar la correlación entre las variables numéricas
corr_matrix = X_train[num_cols].corr()
plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, cmap='Blues')
plt.title('Correlation between numeric features')
plt.tight_layout()
plt.show()
```



No se observan altas correlaciones entre variables. únicamente como se indició en el último reporte de pandas profiling entre aquellas variables que pertenecen a la misma categoría.

Aplicación del mismo Pipeline de transformación al conjunto de prueba

In [40]:
```
cols_to_drop = ["id",'tipo_dolor_de_pecho']
test_data = pd.concat([X_test, Y_test], axis=1)
```

In [41]:
```
dataLimpioTest =  test_data.drop(cols_to_drop, axis=1).copy()
```

In [42]:
```
# Se eliminan los datos duplicados
dataLimpioTest = dataLimpioTest.drop_duplicates()
```

In [ ]:
```
for col in col_categoricas:
    try:
        dataLimpio[col] = dataLimpio[col].astype('int64')
    except:
        continue
```

In [45]:
```
all_cols = dataLimpioTest.columns
col_categoricas = ['sexo', 'ejercicio_angina','tipo_de_residencia','estado_de_
num_cols = list(set(all_cols) - set(col_categoricas))
```

In [46]:
```
from sklearn.impute import SimpleImputer

g values in numerical columns with the mean
```

```
num_imputer = SimpleImputer(strategy='mean')
dataLimpioTest[num_cols] = num_imputer.fit_transform(dataLimpioTest[num_cols])

# Replace missing values in categorical columns with the mode
cat_imputer = SimpleImputer(strategy='most_frequent')
dataLimpioTest[col_categoricas] = cat_imputer.fit_transform(dataLimpioTest[col_
```

In [47]:
```
col_num_test = dataLimpioTest.select_dtypes(include='number').columns
int_cols_test = dataLimpioTest.select_dtypes(include=np.int64).columns
for col in col_num_test:
    if col != 'enfermedad_cardiaca':
        dataLimpioTest[col] = dataLimpioTest[col].fillna(dataLimpio[col].mean(
# Se redondean los valores de las columnas enteras
dataLimpioTest[int_cols_test] = dataLimpioTest[int_cols_test].round(0).astype(
dataLimpioTest
```

Out[47]:

	edad	sexo	presion_sanguinea	colesterol	frecuencia_cardiaca_maxima	ejercicio_angina
189	30.0	1.0	102.0	146.0	187.0	1
383	40.0	1.0	132.0	163.0	124.0	1
24	83.0	0.0	133.0	164.0	142.0	1
717	67.0	1.0	171.0	227.0	132.0	0
218	34.0	0.0	132.0	133.0	85.0	0
...	...	...	...	...	...	...
670	70.0	1.0	178.0	221.0	179.0	1
325	47.0	1.0	131.0	168.0	190.0	1
580	78.0	1.0	131.0	182.0	95.0	0
104	80.0	1.0	178.0	124.0	70.0	1
5	59.0	1.0	101.0	166.0	119.0	1

200 rows × 14 columns

In [48]:
```
col_num = dataLimpioTest.select_dtypes(include='number').columns
for col in col_num:
    Q1 = dataLimpioTest[col].quantile(0.25)
    Q3 = dataLimpioTest[col].quantile(0.75)
    IQR = Q3 - Q1
    dataLimpioTest[col] = np.where(dataLimpioTest[col] < (Q1 - 1.5 * IQR), data
    dataLimpioTest[col] = np.where(dataLimpioTest[col] > (Q3 + 1.5 * IQR), data
```

In [49]:
```
dataLimpioTest, original_data_test = one_hot_encoder(dataLimpioTest)
dataLimpioTest.info()
```

```
Completado OneHot
<class 'pandas.core.frame.DataFrame'>
Index: 200 entries, 189 to 5
Data columns (total 19 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   edad                            200 non-null    float64
 1   presion_sanguinea               200 non-null    float64
 2   colesterol                      200 non-null    float64
 3   frecuencia_cardiaca_maxima      200 non-null    float64
 4   glucosa_plasma                  200 non-null    float64
 5   grosor_piel                     200 non-null    float64
 6   insulina                        200 non-null    float64
 7   bmi                             200 non-null    float64
 8   diabetes_pedigree               200 non-null    float64
 9   enfermedad_cardiaca             200 non-null    float64
 10  sexo_0.0                        200 non-null    float64
 11  sexo_1.0                        200 non-null    float64
 12  ejercicio_angina_0              200 non-null    float64
 13  ejercicio_angina_1              200 non-null    float64
 14  tipo_de_residencia_Rural        200 non-null    float64
 15  tipo_de_residencia_Urban        200 non-null    float64
 16  estado_de_fumador_Non-Smoker    200 non-null    float64
 17  estado_de_fumador_Smoker        200 non-null    float64
 18  estado_de_fumador_Unknown       200 non-null    float64
dtypes: float64(19)
memory usage: 31.2 KB
```

In [50]:
```
Y_test = dataLimpioTest['enfermedad_cardiaca']
X_test = dataLimpioTest.drop(['enfermedad_cardiaca'], axis=1)
```

Es importante recalcar que esta fase es la que hace parte de la fase de entendimiento de los datos de la metodología ASUM-DM. Esta metodología se centra en definir estrategias específicas para la transformación de los datos. En este caso, se decidió realizar la transformación de los datos de acuerdo con lo que se realizó en el conjunto de entrenamiento. Lo cual incluyó la verificación de los supuestos de consistencia, unicidad, completitud y validez. Asimismo, se decidió realizar la imputación de los valores faltantes con la media de la columna dependiendo si las variables eran categóricas o numéricas. Por último, se decidió realizar la transformación de las variables categóricas a numéricas. Una vez hecha la transformación a ambos datasets se puede proseguir con la siguiente parte. Es decir, con el modelamiento.

Parte 3: Modelamiento

Inicialmente, se plantea la búsqueda de hiperparámetros para el árbol de decisión. Para ello se utiliza la libreria sklearn.model_selection y la función GridSearchCV.

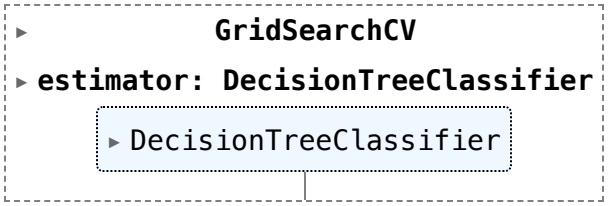
In [51]:
```
#Definicion del numero de particiones para la validacion cruzada asi como los p
particiones = KFold(n_splits=5, shuffle=True, random_state=1234)
param_grid = {'max_depth': np.arange(1, 20,2), 'criterion': ['gini', 'entropy']
```

Posteriormente, se plantea el modelo con DecisionTreeClassifier con los hiperparámetros encontrados.

In [52]:
```
arbol = DecisionTreeClassifier(random_state=1234)
```

In [53]:
```
# Busqueda del mejor modelo con GridSearch
arbol_cv = GridSearchCV(arbol, param_grid, cv=particiones)
arbol_cv.fit(X_train, Y_train)
```

Out[53]:



Como se observa a continuación, el resultado más alto se obtuvo con el criterio "entropy" y con una profundidad de "max_depth" de 19. Esto tiene sentido dado que a mayor profundidad se espera que haya una mejor clasificación.

In [54]:
```
# Resultado de la busqueda
print("Mejor Score: %f" % arbol_cv.best_score_)
print("Mejores Hiperparámetros: ", arbol_cv.best_params_)
```

```
Mejor Score: 0.528750
Mejores Hiperparámetros:  {'criterion': 'entropy', 'max_depth': 19}
```

Posteriormente, se verifica que ambos sean del mismo tipo para no tener inconvenientes en los pasos que siguen.

In [55]:
```
print("Y_train data type:", Y_train.dtype)
print("Y_test data type:", Y_test.dtype)
```

```
Y_train data type: float64
Y_test data type: float64
```

Se observa en los resultados finales obtenidos que el modelo se está sobreajustando a los datos. Esto se muestra dado que las métricas obtenidas para el conjunto de prueba son mucho más bajas que para el conjunto de validación. En particular, por una diferencia de 40 puntos porcentuales.

In [56]:
```
# Construcción del árbol con los mejores hiperparámetros
mejor_arbol = arbol_cv.best_estimator_
y_pred_train = mejor_arbol.predict(X_train)
y_pred_test = mejor_arbol.predict(X_test)
print("Precisión en el conjunto de entrenamiento: ", precision_score(Y_train, y
print("Precisión en el conjunto de test: ", precision_score(Y_test, y_pred_tes
print("Accuracy en el conjunto de entrenamiento: ", accuracy_score(Y_train, y_
print("Accuracy en el conjunto de test: ", accuracy_score(Y_test, y_pred_test)
```

```
Precisión en el conjunto de entrenamiento:  0.9489795918367347
Precisión en el conjunto de test:  0.5148514851485149
Accuracy en el conjunto de entrenamiento:  0.93625
Accuracy en el conjunto de test:  0.51
```

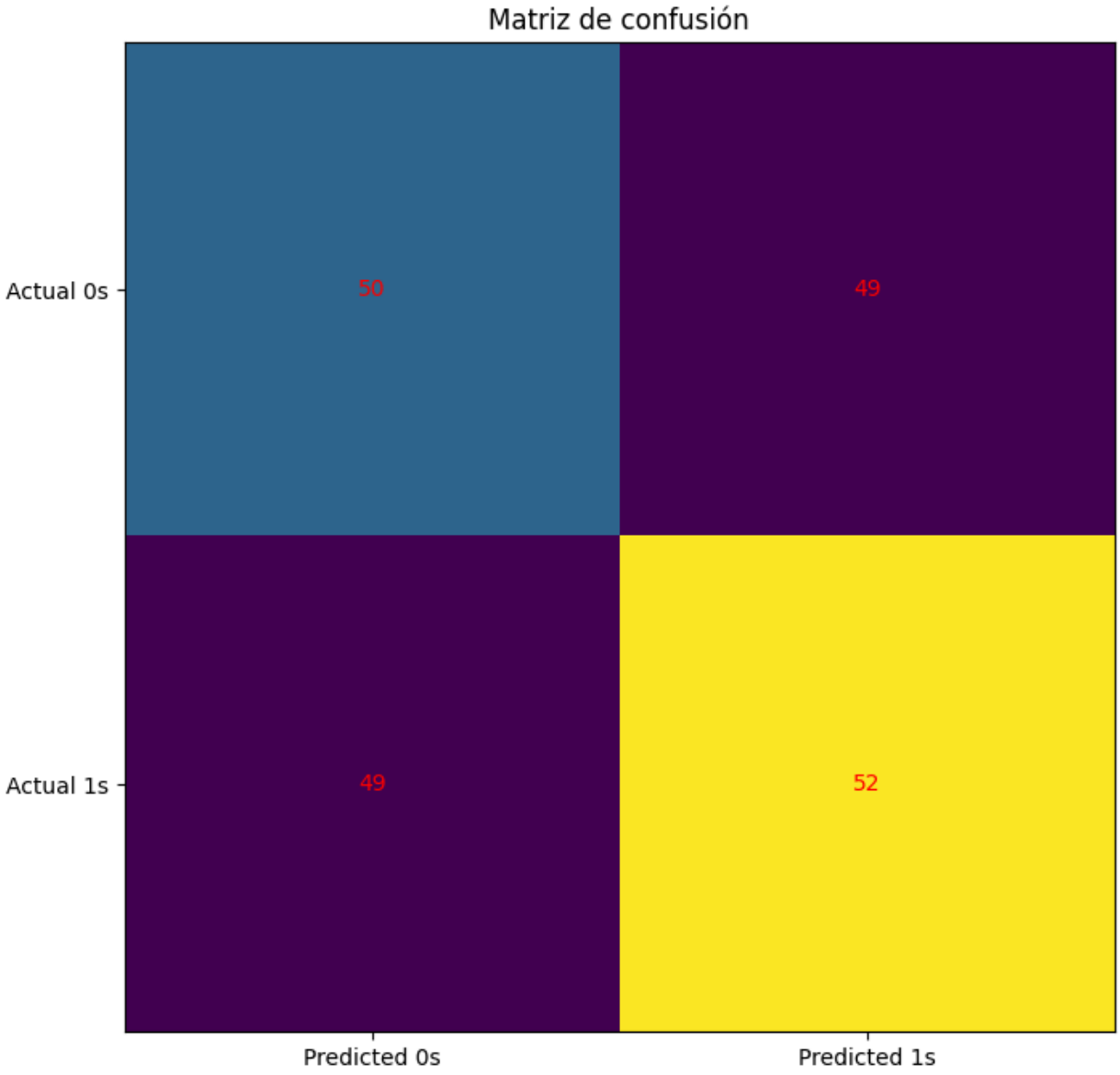
In [57]:
```
# Se genera la matriz de confusión
                                    (Y_test, y_pred_test)
```

Out[57]:
```
array([[50, 49],
       [49, 52]], dtype=int64)
```

Vale la pena recalcar que un paciente que tiene un label de 0 significa que NO presenta una enfermedad cardiaca. Por el contrario si presenta un 1 si tiene enfermedad cardiaca.

In [58]:
```
# plot confusion matrix
fig, ax = plt.subplots(figsize=(8, 8))
ax.imshow(confusion_matrix(Y_test, y_pred_test))
ax.grid(False)
ax.xaxis.set(ticks=(0, 1), ticklabels=('Predicted 0s', 'Predicted 1s'))
ax.yaxis.set(ticks=(0, 1), ticklabels=('Actual 0s', 'Actual 1s'))

for i in range(2):
    for j in range(2):
        ax.text(j, i, confusion_matrix(Y_test, y_pred_test)[i, j], ha='center'
plt.title("Matriz de confusión")
plt.show()
```



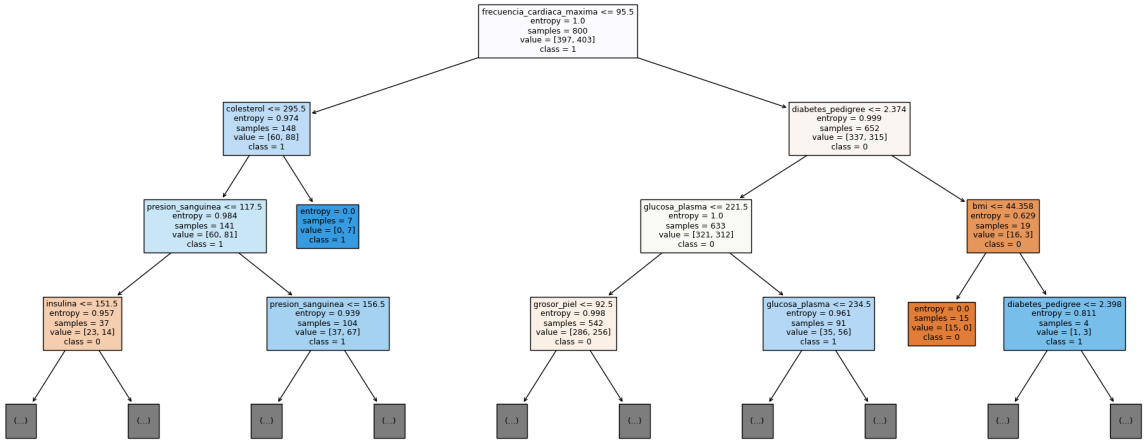
In [59]:
```
# se genera el reporte de clasificación
tion_report(Y_test, y_pred_test))
```

	precision	recall	f1-score	support
0.0	0.51	0.51	0.51	99
1.0	0.51	0.51	0.51	101
accuracy			0.51	200
macro avg	0.51	0.51	0.51	200
weighted avg	0.51	0.51	0.51	200

A continuación, se muestra el árbol de decisión a la que llegó el modelo luego de realizar un grid search con valores de 'max_depth': np.arange(1, 20,2), 'criterion': ['gini', 'entropy']. Este es un modelo altamente interpretable por lo que le puede servir a la organización para saber cómo clasificar a sus pacientes. Además, en particular, en el trigae inicial médico se puede utilizar en caso de que se requiere clasificar rápidamente a las personas que tienen o no una enfermedad cardiovascular.

In [60]:
```
from sklearn import tree
from matplotlib import pyplot as plt

fig = plt.figure(figsize=(25,10))
_ = tree.plot_tree(mejor_arbol, max_depth=3, feature_names=X_train.columns, cla
```



Parte 4: Resulados Finales (15%)

Finalmente, se analizan los resultados obtenidos a lo largo de esta implementación.

1) En primer lugar, se realizó la definicón del enfoque analítico para este proyecto en cuestión. Se decidió utilizar clasificación dado que la empresa Hospital de los Alpes requiere "buscar formas de adaptar sus servicios a las características de los usuarios, principalmente de los pacientes que llegan a sus instalaciones". Por lo que dada una variable objetivo binaria o multiclase se puede utilizar un modelo de clasificación para predecir si un paciente tiene o no una enfermedad cardiaca en este caso. Asimismo, se decidió utilizar un modelo de aprendizaje supervisado dado que se tiene acceso a un conjunto de datos etiquetado en el que se conoce la variable objetivo. Por último, se decidió

 decisión dado que es un algoritmo que se puede utilizar para

clasificación y es fácil de interpretar en particular en el ámbito médico donde cada una de las decisiones puede impactar la vida de una persona por lo que estas son *high stake decissions* en las que se necesita que la interpretabilidad del modelo sea alta.

2) Se realizó la limpieza de los datos recibidos haciendo imputación de los valores nulos con la modo o la media dependiendo si era una variable categórica o numérica. Asimismo, se exploró la posibilidad de que hubieran valores fuera del rango establecido por el diccionario no obstante no se encontraron caso para este dataset basandose en el conjunto de datos de entrenamiento.

1. Se realizó el mismo pipeline de limpieza para el conjunto de datos de prueba. Si bien no se construyó un pipeline como tal el proceso seguido fue el mismo.

1. Se valido el modelo obtenido con un grid search sobre los valores param_grid = {'max_depth': np.arange(1, 20,2), 'criterion': ['gini', 'entropy']}. Se encontró que los mejores resultados´para las métricas se obtenían con un criterio de entropía y una profundidad de 19. Esto tiene sentido dado que a mayor profundidad se espera que haya una mejor clasificación.

1. Se realizó la evaluación del modelo donde el modelo clasificó de manera correcta a 52 pacientes que efectivamente tenían enfermedad cardiaca y a 50 pacientes que no tenían enfermedad cardiaca. No obstante no clasificó de manera correcta a 98 pacientes. Lo que le da un accuracy al modelo de 51%. Esto implica para el negocio que este modelo, al menos en la clasificación de pacientes que pueden llegar al hospital con una enfermedad cardiaca, no es el más adecuado. Aún la métrica se encuentra muy por debajo del accuracy humano y muy probablemente puede llegar a tener grandes efectos si una persona llega y no se le clasifica adecuadamente.

En conclusión, el modelo parece estar overfitting los datos de entrenamiento y a la hora de probar el modelo ante datos con posibles *distribution shifs* las métricas no son las mejores. Se requiere una mayor exploración y experimentación con otros métodos, por ejemplo, KNN para que el modelo pueda ser efectivamente *deployed* en estas circunstancias de alto riesgo. Igualmente, la clasificación es una buena técnica que puede considerar el hospital para ayudar al profesional de la salud a guiar sus decisiones con múltiples variables no únicamente con pacientes que sufren o no de enfermedad cardiaca.