UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS

FACULTAD DE INGENIERÍA

BUSINESS PREDICTIVE ANALYTICS (SI709)

TRABAJO FINAL

PROFESOR:

Caldearon Sulca, Carlos Felipe

SECCIÓN: SS92

AUTORES: GRUPO 2

Integrantes:

- Bendezú Jiménez, Luis Ángel	U202022364
- Caqui Pizarro, Valeria Milagros	U20211C241
- Silvina Gonzales, Gianella Lucía	U202019652
- Trujillo Neyra Neil Eduardo	U202020118
- Vásquez Castro, André Nicolas	U20211B559

Lima, 27 de junio de 2024

RESUMEN

El presente informe detalla el desarrollo y la fundamentación de nuestro proyecto enfocado en la Clínica Mayo, con el objetivo de abordar la necesidad de detección y manejo tempranos de las enfermedades cardiovasculares (ECV). Para ello, hemos utilizado un conjunto de datos que incluye información clínica relevante, como factores de riesgo cardiovascular, resultados de pruebas de laboratorio y diagnósticos médicos.

Nuestro análisis exhaustivo de los datos tiene como finalidad proyectar de manera precisa el riesgo de desarrollar enfermedades cardiovasculares en pacientes atendidos en la Clínica Mayo. Este enfoque permitirá identificar a las personas con mayor riesgo de ECV antes de que ocurran eventos graves, como ataques cardíacos o accidentes cerebrovasculares.

ÍNDICE

OBJETIVO DEL ESTUDIANTE (STUDENT OUTCOME)

ABET - EAC - Student Outcome 6: La capacidad para comprender y brindar soporte para el uso, entrega y gestión de sistemas de información dentro de un entorno de sistemas de información.

Para alcanzar este objetivo, entendemos y gestionamos los sistemas de información utilizando el dataset que hemos investigado previamente. De esta manera, organizamos adecuadamente su información. Además, ofreceremos soporte basado en nuestro conocimiento en este ámbito.

ABET - CAC - Student Outcome 7: La capacidad de adquirir y aplicar nuevos conocimientos según sea necesario, utilizando estrategias de aprendizaje apropiadas.

Para alcanzar este objetivo, utilizamos machine learning en el desarrollo del proyecto, lo que nos permite gestionar de manera eficaz nuestro dataset y su información. También entendemos las necesidades específicas de la empresa seleccionada, en este caso Netflix, y cómo nuestros conocimientos pueden ser de ayuda.

INTRODUCCIÓN

La Clínica Mayo, reconocida por su excelencia en el cuidado de la salud y la investigación médica, se enfrenta a un desafío crítico en el ámbito de la detección y manejo de las enfermedades cardiovasculares (ECV). Con el contexto global de las ECV como principal causa de mortalidad, cobrando 17,9 millones de vidas cada año y representando el 31% de todas las muertes en todo el mundo, surge una necesidad urgente de abordar la detección y el manejo temprano de estas enfermedades.

El problema radica en la identificación precoz de los factores de riesgo y la implementación de estrategias efectivas para prevenir y tratar las ECV. A pesar de los avances en la investigación y el tratamiento, la carga de morbilidad y mortalidad asociada con las ECV sigue siendo significativa, con un tercio de las muertes prematuras ocurriendo en personas menores de 70 años.

En este contexto, se empleará el análisis de datos y técnicas de machine learning para abordar este desafío. Utilizando un conjunto de datos que proporciona una visión detallada de las características clínicas y biomédicas relevantes para la predicción de enfermedades cardíacas, se buscará identificar patrones y correlaciones significativas. El objetivo es desarrollar modelos predictivos precisos y herramientas de apoyo a la toma de decisiones clínicas que contribuyan al avance de la detección precoz y el manejo efectivo de las enfermedades cardiovasculares; Con el respaldo y la colaboración de la Clínica Mayo, reconocida por su liderazgo en el cuidado de la salud cardiovascular, este proyecto tiene como meta mejorar los resultados para los pacientes y reducir la carga de enfermedad cardiovascular a nivel global.

CAPÍTULO 1: ENTENDIMIENTO DEL NEGOCIO

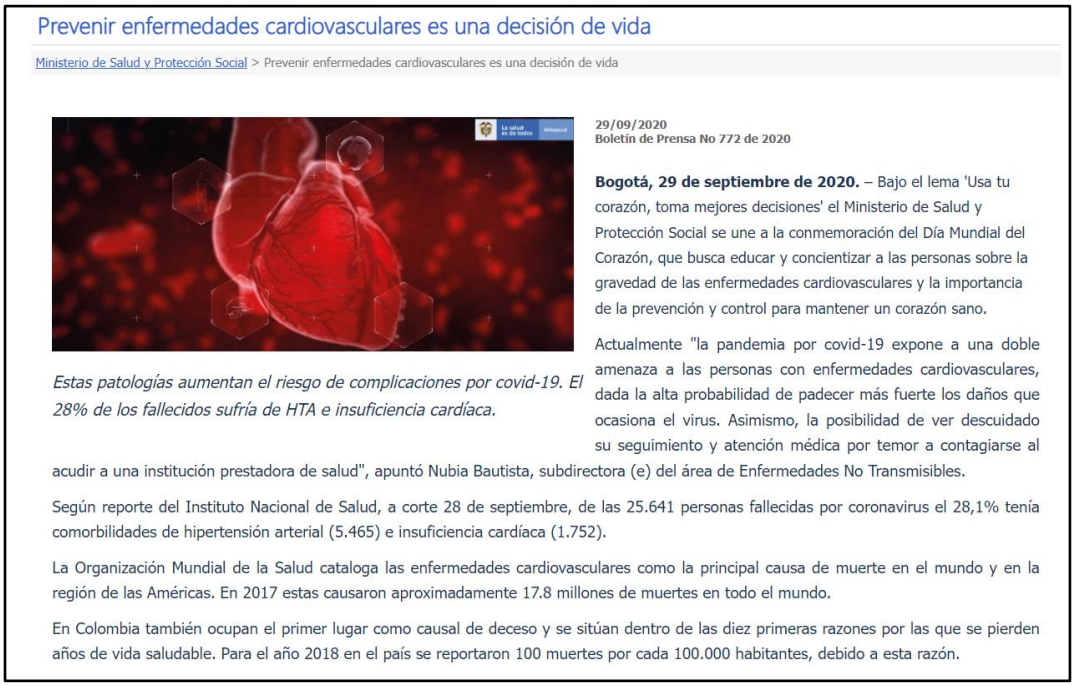
1.1. DEFINIR EL PROBLEMA

El principal desafío de negocio para la Clínica Mayo se centra en la necesidad de mejorar la detección y el manejo temprano de las enfermedades cardiovasculares (ECV). A pesar de su reconocida reputación en el cuidado de la salud cardiovascular, la clínica enfrenta dificultades para identificar de manera precoz los factores de riesgo y adoptar medidas preventivas efectivas contra las ECV. Esto se traduce en una limitada capacidad para anticipar y gestionar de manera proactiva las enfermedades cardiovasculares entre su población de pacientes.

Para abordar este desafío, es fundamental desarrollar e implementar soluciones innovadoras que mejoren la detección precoz, la gestión de los factores de riesgo y la adopción de medidas preventivas para las enfermedades cardiovasculares. Estas soluciones deben aprovechar tecnologías avanzadas, como el análisis de datos y el machine learning, para identificar patrones y correlaciones significativas en los datos clínicos de los pacientes.

Figura 1

Imagen sobre fuente de prevenir enfermedades cardiovasculares es una decisión de vida



Prevenir enfermedades cardiovasculares es una decisión de vida

Ministerio de Salud y Protección Social > Prevenir enfermedades cardiovasculares es una decisión de vida

Estas patologías aumentan el riesgo de complicaciones por covid-19. El 28% de los fallecidos sufría de HTA e insuficiencia cardíaca.

29/09/2020
Boletín de Prensa No 772 de 2020

Bogotá, 29 de septiembre de 2020. – Bajo el lema 'Usa tu corazón, toma mejores decisiones' el Ministerio de Salud y Protección Social se une a la conmemoración del Día Mundial del Corazón, que busca educar y concientizar a las personas sobre la gravedad de las enfermedades cardiovasculares y la importancia de la prevención y control para mantener un corazón sano.

Actualmente "la pandemia por covid-19 expone a una doble amenaza a las personas con enfermedades cardiovasculares, dada la alta probabilidad de padecer más fuerte los daños que ocasiona el virus. Asimismo, la posibilidad de ver descuidado su seguimiento y atención médica por temor a contagiarse al acudir a una institución prestadora de salud", apuntó Nubia Bautista, subdirectora (e) del área de Enfermedades No Transmisibles.

Según reporte del Instituto Nacional de Salud, a corte 28 de septiembre, de las 25.641 personas fallecidas por coronavirus el 28,1% tenía comorbilidades de hipertensión arterial (5.465) e insuficiencia cardíaca (1.752).

La Organización Mundial de la Salud cataloga las enfermedades cardiovasculares como la principal causa de muerte en el mundo y en la región de las Américas. En 2017 estas causaron aproximadamente 17.8 millones de muertes en todo el mundo.

En Colombia también ocupan el primer lugar como causal de deceso y se sitúan dentro de las diez primeras razones por las que se pierden años de vida saludable. Para el año 2018 en el país se reportaron 100 muertes por cada 100.000 habitantes, debido a esta razón.

Nota: Se puede ver la explicación de lo importante que es la prevención de las enfermedades cardiovasculares

1.2. EVALUAR Y ANALIZAR ESCENARIOS

Para el desarrollo del proyecto en la Clínica Mayo, se empleará un enfoque basado en el análisis predictivo utilizando técnicas de Machine Learning. El objetivo es determinar el trayecto de detección y manejo temprano de las enfermedades cardiovasculares (ECV) en los pacientes, evaluando cómo varía esta detección a lo largo del tiempo.

Para la implementación de Machine Learning, se utilizará Miniconda como gestor de paquetes de Python y Jupyter para llevar a cabo las tareas de extracción, transformación y carga de datos (ETL). Estas herramientas proporcionarán un entorno eficiente y flexible para el análisis de datos y el desarrollo de modelos predictivos en el contexto de la Clínica Mayo.

1.3. DEFINIR LOS OBJETIVOS DE ML

- Desarrollar un modelo predictivo que pueda identificar tempranamente la probabilidad de desarrollo de enfermedades cardiovasculares en pacientes, utilizando datos clínicos y biomédicos relevantes
- Diseñar algoritmos capaces de predecir la progresión de las enfermedades cardiovasculares en pacientes a lo largo del tiempo, considerando factores de riesgo y variables clínicas.
- Facilitar a los profesionales de la salud en la Clínica Mayo la identificación de patrones y correlaciones significativas entre las variables clínicas y biomédicas, para mejorar la toma de decisiones clínicas y el diseño de intervenciones preventivas.
- Desarrollar herramientas de apoyo a la toma de decisiones que permitan personalizar el manejo y tratamiento de los pacientes con riesgo de enfermedades cardiovasculares, optimizando así los recursos y mejorando los resultados clínicos.

1.4. PLAN DE PROYECTO

	EDT	Nombre de tarea	Duración	Comienzo	Fin	Nombre de los recursos
1	1	Trabajo Final	48 horas	Viernes 22/03/2024	Jueves 27/06/2024	Grupo 02
2	1.1	Capítulo 1: Entendimiento del negocio	7 horas	Viernes 22/03/2024	Jueves 27/06/2024	
3	1.1.1	Definir el problema	2 horas	Viernes 22/03/2024	Jueves 27/06/2024	André/Valeria/Luis/Gianella/Neil
4	1.2	Evaluar y analizar escenarios	1 hora y 30 minutos	Viernes 22/03/2024	Jueves 27/06/2024	André/Valeria/Luis/Gianella/Neil
5	1.3	Definir los objetivos de ML	1 hora y 30 minutos	Viernes 22/03/2024	Jueves 27/06/2024	André/Valeria/Luis/Gianella/Neil
6	1.4	Plan de proyecto	2 horas	Viernes 22/03/2024	Jueves 27/06/2024	André/Valeria/Luis/Gianella/Neil

7	2	Capítulo 2: Pre-Procesamiento de los Datos	16 horas	Miércoles 10/04/2024	Jueves 27/06/2024	
8	2.1	Colocar los datos	2 horas	Miércoles 10/04/2024	Jueves 27/06/2024	André Nicolas Vásquez Castro
9	2.2	Calidad y Limpieza de datos	6 horas	Miércoles 10/04/2024	Jueves 27/06/2024	André Nicolas Vásquez Castro
10	2.3	EDA	4 horas	Sábado 20/04/2024	Jueves 27/06/2024	Luis Bendezú/Neil Trujillo
11	2.4	Transformación de los datos	4 horas	Viernes 03/05/2024	Jueves 27/06/2024	Luis Bendezú/Neil Trujillo
12	3	Capítulo 3: Resultados sobre análisis de datos	4 horas	Sábado 20/04/2024	Jueves 27/06/2024	
13	3.1	Insights	4 horas	Sábado 20/04/2024	Jueves 27/06/2024	Valeria Caqui/Gianella Silvina/Neil Trujillo/Luis Bendezú

14	4	Capítulo 4: Modelización y Optimización	15 horas	Sábado 11/05/2024	Jueves 27/06/2024	
15	4.1	Modelización	8 horas	Sábado 11/05/2024	Jueves 27/06/2024	André Nicolas Vásquez Castro
16	4.2	Optimización	7 horas	Viernes 17/05/2024	Jueves 27/06/2024	Valeria Milagros Caqui Pizarro/Gianella Silvina
17	5	Capítulo 5: Resultados de modelización	6 horas	10/06/2024	Jueves 27/06/2024	
18	5.1	Presentación de resultados finales	6 horas	10/06/2024	Jueves 27/06/2024	Luis Ángel Bendezú Jimenez
19	6	Aportes	2 horas	15/06/2024	Jueves 27/06/2024	André Nicolas Vásquez Castro
20	7	Conclusiones	2 horas	18/06/2024	Jueves 27/06/2024	Valeria Milagros Caqui Pizarro

21	8	Recomendaciones	2 horas	23/06/2024	Jueves 27/06/2024	Neil Trujillo
22	9	Glosario	1 hora	24/06/2024	Jueves 27/06/2024	Luis Ángel Bendezú Jimenez
23	10	Bibliografía	2 horas	25/06/2024	Jueves 27/06/2024	Gianella Silvina

CAPÍTULO 2: PRE-PROCESAMIENTO DE LOS DATOS

2.1: COLECTAR LOS DATOS

PRE-PROCESAMIENTO DE LOS DATOS

Para poder recolectar los datos necesarios, hemos tomado un conjunto de datos o también llamado dataset con la información de varias variables necesarias para su análisis del año 2023, estas variables son diagnósticos médicos, factores biomédicos, entre otros.

Conjunto de datos sobre enfermedades cardiacas

Durante nuestro proceso de investigación, hemos identificado un valioso conjunto de datos alojado en Kaggle que contiene información detallada sobre enfermedades cardíacas. Este dataset, recopilado en el año 2023, proporciona una amplia gama de variables relevantes para el estudio y la predicción de enfermedades cardiovasculares. Entre los datos incluidos, como habíamos dicho antes, se encuentran diagnósticos médicos, factores biomédicos y clínicos, así como información demográfica de los pacientes. La disponibilidad de este conjunto de datos nos brinda la oportunidad de realizar un análisis profundo y detallado, así como de aplicar técnicas de aprendizaje automático para desarrollar modelos predictivos precisos.

- **CARGA DE DATOS:**

Importamos las bibliotecas y cargamos el drive:

```
import pandas as pd
import pickle
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline

#Automcompletar rápido
%config IPCompleter.greedy=True
#Desactivar la notación científica
pd.options.display.float_format = '{:.3f}'.for

from IPython.display import display
```

```
[2] #Google Drive
    from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive
```

Después hacemos la carga de datos del dataset correspondiente:

```
[14] # Cargar el DataFrame desde el archivo CSV
     df_heart_disease_dirty5 = pd.read_csv("/content/drive/My Drive/BPA/TP/heart_disease_dirty5.csv", encoding='ISO-8859-1', delimiter=";")

     # Ordenar filas y columnas
     df_heart_disease_dirty5 = df_heart_disease_dirty5.sort_index(axis=0).sort_index(axis=1)

     # Mostrar las primeras filas del DataFrame ordenado
     df_heart_disease_dirty5.head(5)
```

	BMI	BPMeds	Gender	Heart_ stroke	age	cigsPerDay	currentSmoker	diaBP	diabetes	education	glucose	heartRate	prevalentHyp	prevalentStroke	sysBP	totChol
0	26.97	0	Male	No	39.000	0	0.000	70.000	0.000	postgraduate	77	80.000	0.000	no	106.000	195
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	25.34	0	Male	No	48.000	20	1.000	80.000	0.000	uneducated	70	75.000	0.000	no	127.500	245
3	28.58	0	Female	yes	61.000	30	1.000	95.000	0.000	graduate	103	65.000	1.000	no	150.000	225
4	23.1	0	Female	No	46.000	23	1.000	84.000	0.000	graduate	85	85.000	0.000	no	130.000	285

2.2 CALIDAD Y LIMPIEZA DE DATOS

En esta parte del informe, vamos a aplicar la limpieza de datos.

- **CORRECCIÓN DE CABECERAS O HEADERS**

Vamos a empezar con la limpieza de datos, primero la corrección de cabeceras:

```
[15] columnas_renombrar =  {'BPMeds':'blood_pressure_medications','BMI':'body_mass_index','Gender':'gender','Heart_ stroke':'heart_disease','cigsPerDay':'cigarettes_per_day','currentSmoker
     df_heart_disease_dirty5.rename(columns = columnas_renombrar, inplace = True)
```

```
df_heart_disease_dirty5.head(3)
```

	body_mass_index	blood_pressure_medications	gender	heart_disease	age	cigarettes_per_day	is_current_smoker	diastolic_blood_pressure	diabetes	education	glucose	heartRate
0	26.97	0	Male	No	39.000	0	0.000	70.000	0.000	postgraduate	77	80.000
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	25.34	0	Male	No	48.000	20	1.000	80.000	0.000	uneducated	70	75.000

- **VERIFICACIÓN DE INCONSISTENCIAS TEXTUALES**

Primero, verificamos las inconsistencias textuales del dataset:

Verificación de Inconsistencias Textuales:

```
# Verificar inconsistencias textuales en datos de tipo string
def verificar_textos(df):
    for column in df.select_dtypes(include=['object']).columns:
        print(f"\nVerificación en columna '{column}':")
        print(f"Valores en mayúsculas: {df[column][df[column].str.isupper()].unique()}")
        print(f"Valores en minúsculas: {df[column][df[column].str.islower()].unique()}")

# Llamar a la función de verificación de textos
verificar_textos(df_cat)
```

```
Valores en mayúsculas: []
Valores en minúsculas: []

Verificación en columna 'blood_pressure_medications':
Valores en mayúsculas: []
Valores en minúsculas: []

Verificación en columna 'gender':
Valores en mayúsculas: ['FEMALE' 'MALE']
Valores en minúsculas: ['male' 'female']

Verificación en columna 'heart_disease':
Valores en mayúsculas: ['YES' 'NO']
Valores en minúsculas: ['yes' 'no']

Verificación en columna 'cigarettes_per_day':
Valores en mayúsculas: []
Valores en minúsculas: []

Verificación en columna 'education':
Valores en mayúsculas: ['PRIMARYSCHOOL' 'UNEDUCATED' 'GRADUATE' 'POSTGRADUATE']
Valores en minúsculas: ['postgraduate' 'uneducated' 'graduate' 'primaryschool']

Verificación en columna 'glucose':
Valores en mayúsculas: []
Valores en minúsculas: []

Verificación en columna 'has_prevalent_stroke':
Valores en mayúsculas: ['NO' 'YES']
Valores en minúsculas: ['no' 'yes']

Verificación en columna 'total_cholesterol':
Valores en mayúsculas: []
Valores en minúsculas: []
```

Y ahora, hacemos las correcciones de las inconsistencias textuales:

Corrección de Inconsistencias Textuales:

```
[ ]  # Corregir inconsistencias textuales en datos de tipo string
     def corregir_textos(df):
         for column in df.select_dtypes(include=['object']).columns:
             df[column] = df[column].str.title()  # Convertir a formato Título para uniformidad

     # Llamar a la función de corrección de textos
     corregir_textos(df_cat)
```

Como se puede ver, ya no hay inconsistencias textuales:

```
Verificación en columna 'body_mass_index':
Valores en mayúsculas: []
Valores en minúsculas: []

Verificación en columna 'blood_pressure_medications':
Valores en mayúsculas: []
Valores en minúsculas: []

Verificación en columna 'gender':
Valores en mayúsculas: []
Valores en minúsculas: []

Verificación en columna 'heart_disease':
Valores en mayúsculas: []
Valores en minúsculas: []

Verificación en columna 'cigarettes_per_day':
Valores en mayúsculas: []
Valores en minúsculas: []

Verificación en columna 'education':
Valores en mayúsculas: []
Valores en minúsculas: []

Verificación en columna 'glucose':
Valores en mayúsculas: []
Valores en minúsculas: []

Verificación en columna 'has_prevalent_stroke':
Valores en mayúsculas: []
Valores en minúsculas: []

Verificación en columna 'total_cholesterol':
Valores en mayúsculas: []
Valores en minúsculas: []
```

- **IDENTIFICACIÓN DE CELDAS DUPLICADOS**

Ahora vamos a identificar las celdas duplicadas, y borraremos sus filas si se presentan:

DUPLICADOS

```
df_heart_disease_dirty5.duplicated().sum()
```

```
363
```

```
[18] # Identificar filas duplicadas en el DataFrame
filas_duplicadas = df_heart_disease_dirty5[df_heart_disease_dirty5.duplicated()]

# Mostrar las filas duplicadas
print(filas_duplicadas)
```

```
     body_mass_index blood_pressure_medications  gender heart_disease     age  \
15               NaN                        NaN     NaN           NaN     NaN
34               NaN                        NaN     NaN           NaN     NaN
43               NaN                        NaN     NaN           NaN     NaN
180              NaN                        NaN     NaN           NaN     NaN
184              NaN                        NaN     NaN           NaN     NaN
...              ...                        ...     ...           ...     ...
4410           29.48                          0    MALE            NO  43.000
4411             NaN                        NaN     NaN           NaN     NaN
4412           25.63                          0    Male            No  41.000
4413           25.46                          0  Female            No  46.000
4414           24.01                          0    Male            No  39.000
```

Ahora, eliminamos las celdas duplicadas:

```
#Corrección: Eliminar las filas duplicadas
df_heart_disease_dirty5.drop_duplicates(inplace=True)
df_heart_disease_dirty5.shape
```

```
(4052, 16)
```

- **VERIFICACIÓN DE VALORES NULOS**

Vamos a verificar y eliminar los valores nulos del dataset, primero, identificamos

VERIFICACIÓN DE VALORES NULOS EN DATASET

```
[22] df_cat = df_heart_disease_dirty5.select_dtypes(include=['object']).copy()
     df_cat
```

	body_mass_index	blood_pressure_medications	gender	heart_disease	cigarettes_per_day	education	glucose	has_prevalent_stroke	total_cholesterol
0	26.97	0	Male	No	0	postgraduate	77	no	195
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	25.34	0	Male	No	20	uneducated	70	no	245
3	28.58	0	Female	yes	30	graduate	103	no	225
4	23.1	0	Female	No	23	graduate	85	no	285
...	...	...	...	...	...	...	...	...	...
4200	25.97	0	Male	yes	1	uneducated	86	no	313
4201	19.71	0	MALE	NO	43	GRADUATE	68	NO	207
4202	22	NaN	Female	No	20	primaryschool	86	no	248
4203	19.16	0	Female	No	15	uneducated	NaN	no	210
4204	21.47	0	Female	No	0	primaryschool	107	no	269

4052 rows × 9 columns

Con porcentajes y saber la cantidad de nulos en cada columna de las variables:

```
#df_cat.isna().sum().sort_values(ascending=False)
df_cat.isnull().sum().sort_values(ascending=False)
```

```
glucose                        381
education                      127
blood_pressure_medications      82
total_cholesterol               82
cigarettes_per_day              60
body_mass_index                 53
heart_disease                   37
has_prevalent_stroke            37
gender                           1
dtype: int64
```

```
[24] df_cat.isnull().sum().sort_values(ascending=False) * 100 / len(df_cat)
```

```
glucose                        9.403
education                      3.134
blood_pressure_medications     2.024
total_cholesterol              2.024
cigarettes_per_day             1.481
body_mass_index                1.308
heart_disease                  0.913
has_prevalent_stroke           0.913
gender                         0.025
dtype: float64
```

Ahora, eliminamos esos valores nulos y verifica

```
[34] # Eliminar filas que contienen valores nulos
df_cat.dropna(inplace=True)

# Verificar si quedan valores nulos después de la eliminación
df_cat.isnull().sum().sort_values(ascending=False) * 100 / len(df_cat)
```

```
BMI              0.000
BPMeds           0.000
Gender           0.000
Heart_ stroke    0.000
cigsPerDay       0.000
education        0.000
glucose          0.000
prevalentStroke  0.000
totChol          0.000
dtype: float64
```

```
[43] # Verificar si quedan valores nulos después de la eliminación
df_cat.isnull().sum().sort_values(ascending=False) * 100 / len(df_cat)
```

```
BMI              0.000
BPMeds           0.000
Gender           0.000
Heart_ stroke    0.000
cigsPerDay       0.000
education        0.000
glucose          0.000
prevalentStroke  0.000
totChol          0.000
dtype: float64
```

Ahora verificamos y eliminamos de las variables numéricas, primero, identificamos:

```
df_num = df_heart_disease_dirty5.select_dtypes(include='number').copy()
```

```
print(df_num.shape)
df_num.isna().sum().sort_values(ascending=False)
```

```
(4052, 7)
heartRate                 38
age                       37
is_current_smoker         37
diastolic_blood_pressure  37
diabetes                  37
hypertension              37
systolic_blood_pressure   37
dtype: int64
```

```
#Analizamos con estadísticos básicos como se comporta el promedio y la media
df_num.describe().T
```

	count	mean	std	min	25%	50%	75%	max
age	4015.000	49.552	8.561	32.000	42.000	49.000	56.000	70.000
is_current_smoker	4015.000	0.496	0.500	0.000	0.000	0.000	1.000	1.000
diastolic_blood_pressure	4015.000	82.873	11.839	48.000	75.000	82.000	89.500	142.500
diabetes	4015.000	0.026	0.159	0.000	0.000	0.000	0.000	1.000
heartRate	4014.000	75.944	12.015	44.000	68.000	75.000	83.000	143.000
hypertension	4015.000	0.308	0.462	0.000	0.000	0.000	1.000	1.000
systolic_blood_pressure	4015.000	132.265	21.869	83.500	117.000	128.000	143.500	295.000

```
[39] valor_heartRate  = df_num['heartRate'].median()
     valor_age    = df_num['age'].median()
     valor_is_current_smoker    = df_num['is_current_smoker'].median()
     valor_hypertension    = df_num['hypertension'].median()
     valor_diabetes    = df_num['diabetes'].median()
     valor_systolic_blood_pressure    = df_num['systolic_blood_pressure'].median()
     valor_diastolic_blood_pressure    = df_num['diastolic_blood_pressure'].median()

     print(valor_heartRate)
     print(valor_age)
     print(valor_is_current_smoker)
     print(valor_hypertension)
     print(valor_diabetes)
     print(valor_systolic_blood_pressure)
     print(valor_diastolic_blood_pressure)
```

```
75.0
49.0
0.0
0.0
0.0
128.0
82.0
```

Ahora, eliminamos los valores nulos de las variables numéricas:

```
[40] df_num['heartRate']  = df_num['heartRate'].fillna(valor_heartRate)
     df_num['age']    = df_num['age'].fillna(valor_age)
     df_num['is_current_smoker']  = df_num['is_current_smoker'].fillna(valor_is_current_smoker)
     df_num['hypertension']    = df_num['hypertension'].fillna(valor_hypertension)
     df_num['diabetes']  = df_num['diabetes'].fillna(valor_diabetes)
     df_num['systolic_blood_pressure']    = df_num['systolic_blood_pressure'].fillna(valor_systolic_blood_pressure)
     df_num['diastolic_blood_pressure']  = df_num['diastolic_blood_pressure'].fillna(valor_diastolic_blood_pressure)
```

```
[42] df_num.isna().sum().sort_values(ascending=False)
```

```
age                      0
is_current_smoker        0
hypertension             0
diabetes                 0
systolic_blood_pressure  0
diastolic_blood_pressure 0
heartRate                0
dtype: int64
```

Aquí tenemos otro procedimiento adicional para la verificación de valores nulos y variantes para categóricas y numéricas:

Verificación de Valores Nulos y Variantes:

```
# Verificar valores nulos y sus variantes en el DataFrame
def verificar_valores_nulos(df):
    print("Recuento de valores nulos por columna:")
    print(df.isnull().sum())
    print("\nRecuento de 'NA', 'N/A', y espacios en blanco por columna:")
    variantes_nulas = ['NA', 'N/A', 'na', '', ' ']
    for variante in variantes_nulas:
        print(f"Recuento de '{variante}': {(df == variante).sum()}")

# Llamar a la función de verificación con el DataFrame
verificar_valores_nulos(df_num)
```

```
Recuento de valores nulos por columna:
age                          0
is_current_smoker            0
diastolic_blood_pressure     0
diabetes                     0
heartRate                    0
hypertension                 0
systolic_blood_pressure      0
dtype: int64

Recuento de 'NA', 'N/A', y espacios en blanco por columna:
Recuento de 'NA': age                          0
is_current_smoker            0
diastolic_blood_pressure     0
diabetes                     0
heartRate                    0
hypertension                 0
systolic_blood_pressure      0
dtype: int64
Recuento de 'N/A': age                          0
is_current_smoker            0
diastolic_blood_pressure     0
diabetes                     0
heartRate                    0
hypertension                 0
systolic_blood_pressure      0
```

```
# Verificar valores nulos y sus variantes en el DataFrame
def verificar_valores_nulos(df):
    print("Recuento de valores nulos por columna:")
    print(df.isnull().sum())
    print("\nRecuento de 'NA', 'N/A', y espacios en blanco por columna:")
    variantes_nulas = ['NA', 'N/A', 'na', '', ' ']
    for variante in variantes_nulas:
        print(f"Recuento de '{variante}': {(df == variante).sum()}")

# Llamar a la función de verificación con el DataFrame
verificar_valores_nulos(df_cat)
```

```
Recuento de valores nulos por columna:
body_mass_index              0
blood_pressure_medications   0
gender                       0
heart_disease                0
cigarettes_per_day           0
education                    0
glucose                      0
has_prevalent_stroke         0
total_cholesterol            0
dtype: int64
```

Y con esto se hace la corrección de esos valores nulo y variantes para categóricas y numéricas también:

Corrección de Valores Nulos y Variantes:

```
# Corregir valores nulos y sus variantes en el DataFrame
def corregir_valores_nulos(df):
    variantes_nulas = ['NA', 'N/A', 'na', '', ' ']
    for variante in variantes_nulas:
        df.replace(variante, pd.NA, inplace=True)  # Reemplaza con un NA que pandas puede manejar mejor
    df.dropna(inplace=True)  # Opcional: eliminar filas con valores NA si necesario

# Llamar a la función de corrección con el DataFrame
corregir_valores_nulos(df_cat)
```

```
[ ] # Corregir valores nulos y sus variantes en el DataFrame
def corregir_valores_nulos(df):
    variantes_nulas = ['NA', 'N/A', 'na', '', ' ']
    for variante in variantes_nulas:
        df.replace(variante, pd.NA, inplace=True)  # Reemplaza con un NA que pandas puede manejar mejor
    df.dropna(inplace=True)  # Opcional: eliminar filas con valores NA si necesario

# Llamar a la función de corrección con el DataFrame
corregir_valores_nulos(df_num)
```

Como se puede ver, ya no hay valores nulos:

```
Recuento de 'NA', 'N/A', y espacios en blanco por columna:
Recuento de 'NA': body_mass_index              0
blood_pressure_medications    0
gender                        0
heart_disease                 0
age                           0
cigarettes_per_day            0
is_current_smoker             0
diastolic_blood_pressure      0
diabetes                      0
education                     0
glucose                       0
heartRate                     0
hypertension                  0
has_prevalent_stroke          0
systolic_blood_pressure       0
total_cholesterol             0
dtype: int64
Recuento de 'N/A': body_mass_index              0
blood_pressure_medications    0
gender                        0
heart_disease                 0
age                           0
cigarettes_per_day            0
is_current_smoker             0
diastolic_blood_pressure      0
diabetes                      0
education                     0
glucose                       0
heartRate                     0
hypertension                  0
has_prevalent_stroke          0
systolic_blood_pressure       0
total_cholesterol             0
dtype: int64
Recuento de 'na': body_mass_index              0
```

Ejecutamos este código también:

```
[ ] df_cat_num = pd.concat([df_cat,df_num], axis =1 ) #concatenar los dos dataframe, con las variables de posicion
    df_cat_num.dropna(inplace=True)
    df_cat_num
```

	body_mass_index	blood_pressure_medications	gender	heart_disease	cigarettes_per_day	education	gluco
0	26.97	0	Male	No	0	Postgraduate	
2	25.34	0	Male	No	20	Uneducated	
3	28.58	0	Female	Yes	30	Graduate	1
4	23.1	0	Female	No	23	Graduate	
5	30.3	0	Female	No	0	Primaryschool	
...	...	...	...	...	...	...	
4198	24.96	0	Male	No	0	Graduate	
4199	23.14	0	Male	Yes	0	Uneducated	
4200	25.97	0	Male	Yes	1	Uneducated	
4201	19.71	0	Male	No	43	Graduate	
4204	21.47	0	Female	No	0	Primaryschool	1

Y por último, grabamos el fin de fase:

GRABAR FIN DE FASE

```
[ ] df_cat.to_pickle('/content/drive/My Drive/BPA/TP/df_heart_disease_cat_fin_quality_cleaning.pickle')
    df_num.to_pickle('/content/drive/My Drive/BPA/TP/df_heart_disease_num_fin_quality_cleaning.pickle')
    df_cat_num.to_pickle('/content/drive/My Drive/BPA/TP/df_heart_disease_cat_num_fin_quality_cleaning.pickle')
```

2.3 EDA

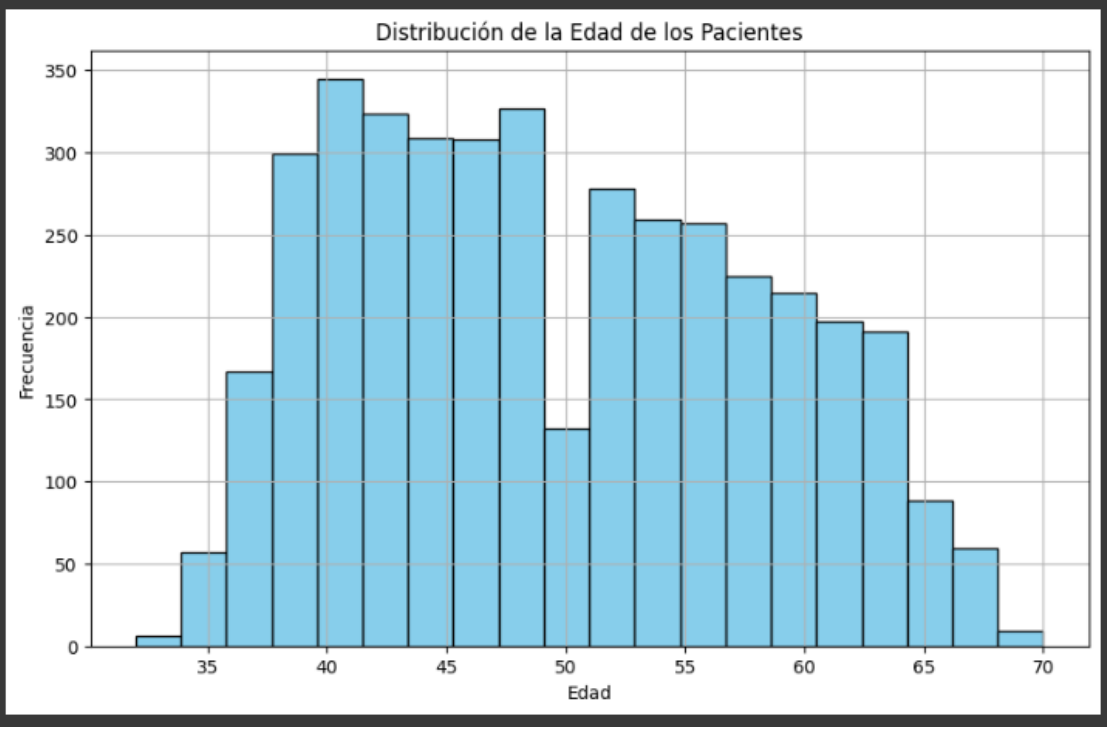
Realizamos un análisis exploratorio de datos para examinar los datos obtenidos de la Clínica Mayo con el fin de comprender la estructura y las características que presentan estos mismos. El EDA es importante en este trabajo, ya que nos permite conocer información valiosa que puede guiarnos en la selección de modelos y técnicas de análisis de datos.

1. **¿Cuál es la distribución de la edad de los pacientes en el conjunto de datos?**

   La distribución de la edad de los pacientes en nuestro conjunto de datos proporciona una visión general de cómo se distribuyen las edades dentro de nuestra muestra. Este análisis nos permite comprender mejor la composición demográfica de nuestra población de pacientes. Al observar la distribución de la edad, podemos identificar tendencias.

DISTRIBUCIÓN DE LA EDAD DE LOS PACIENTES

```
[ ] import matplotlib.pyplot as plt
    # Cargar el DataFrame desde el archivo CSV

    edad_pacientes = df_num['age']

    plt.figure(figsize=(10, 6))
    plt.hist(edad_pacientes, bins=20, color='skyblue', edgecolor='black')
    plt.title('Distribución de la Edad de los Pacientes')
    plt.xlabel('Edad')
    plt.ylabel('Frecuencia')
    plt.grid(True)
    plt.show()
```

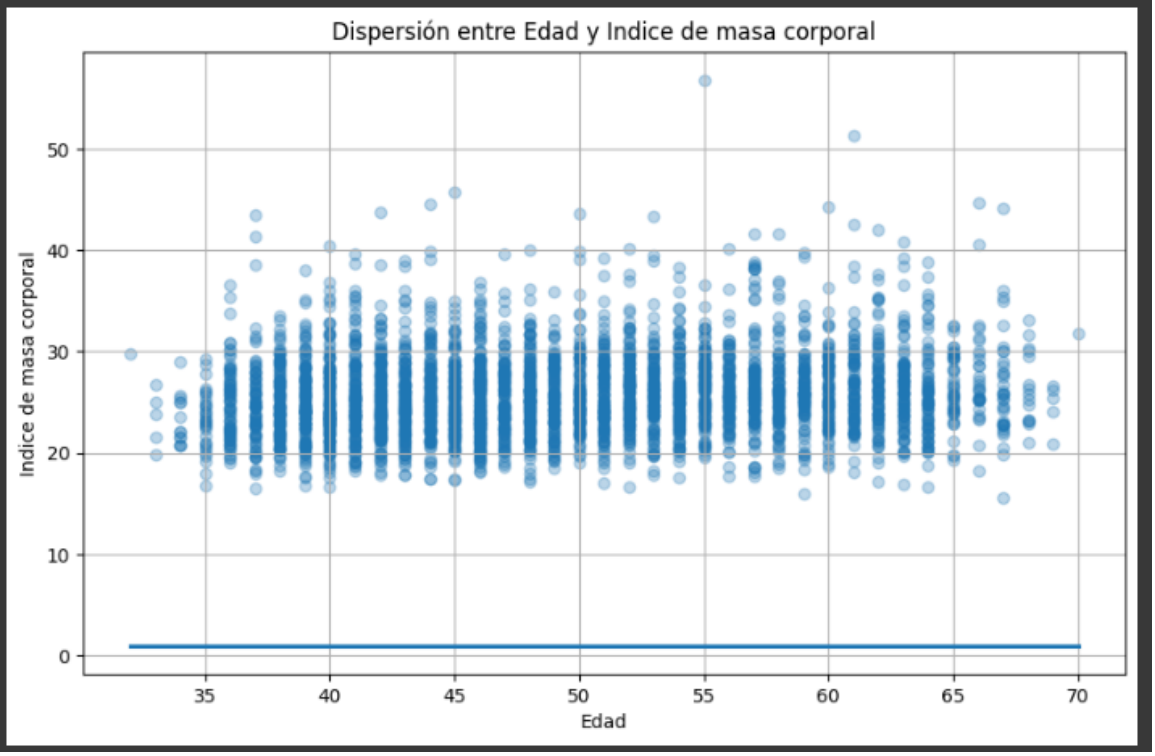


2. ¿Cuál es la distribución del índice de masa corporal (BMI) en la población?

La distribución del índice de masa corporal (BMI) en nuestra población nos brinda información sobre la variabilidad del peso corporal en nuestra muestra. El BMI es una medida importante que nos ayuda a evaluar el estado nutricional y el riesgo de enfermedades relacionadas con el peso. Al examinar su distribución, podemos identificar patrones, como la concentración de valores en rangos específico

DISPERSIÓN ENTRE EDAD Y ÍNDICE DE MASA CORPORAL

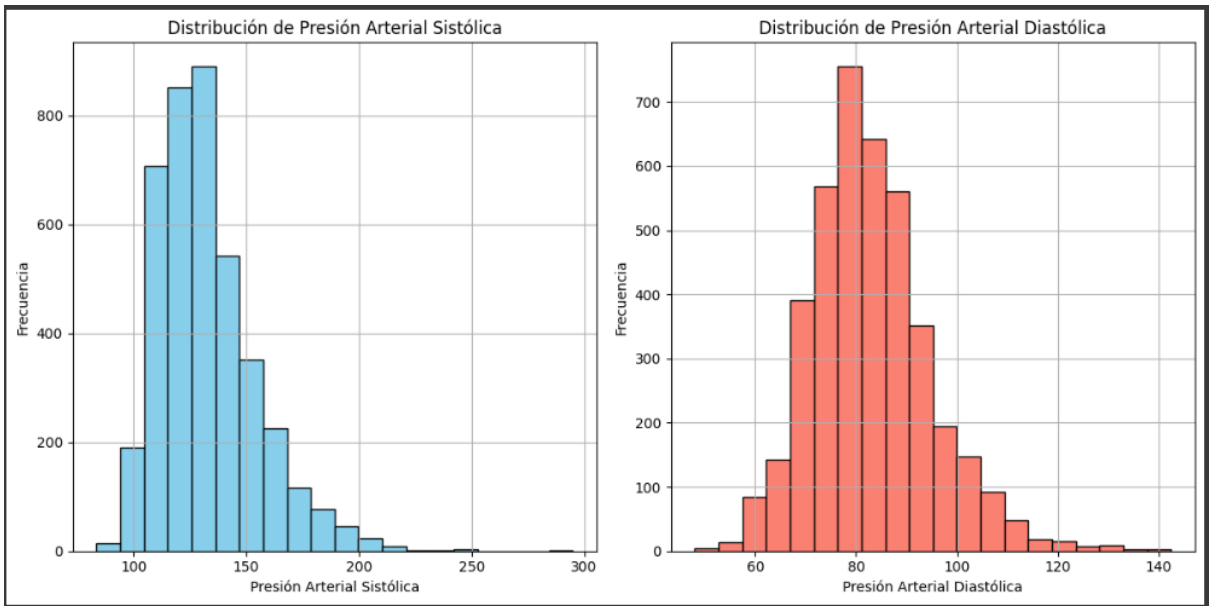
```
import seaborn as sns
df_cat_num['body_mass_index'] = pd.to_numeric(df_cat_num['body_mass_index'], errors='coerce')

plt.figure(figsize=(10, 6))
sns.regplot(x='age', y='body_mass_index', data=df_cat_num, logistic=True, scatter_kws={'alpha':0.3})
plt.title('Dispersión entre Edad y Indice de masa corporal')
plt.xlabel('Edad')
plt.ylabel('Indice de masa corporal')
plt.grid(True)
plt.show()
```



3. ¿Cómo se distribuyen los niveles de presión arterial sistólica y diastólica?

La distribución de los niveles de presión arterial sistólica y diastólica en nuestra población proporciona información crucial sobre la salud cardiovascular de nuestros pacientes. Estos dos valores son fundamentales para evaluar la función cardíaca y el riesgo de enfermedades cardiovasculares. Al analizar su distribución, podemos identificar la gama de valores en los que se encuentran los pacientes, así como posibles anomalías o tendencias

```python
import matplotlib.pyplot as plt


presion_sistolica = df_num['systolic_blood_pressure']
presion_diastolica = df_num['diastolic_blood_pressure']


plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.hist(presion_sistolica, bins=20, color='skyblue', edgecolor='black')
plt.title('Distribución de Presión Arterial Sistólica')
plt.xlabel('Presión Arterial Sistólica')
plt.ylabel('Frecuencia')
plt.grid(True)

plt.subplot(1, 2, 2)
plt.hist(presion_diastolica, bins=20, color='salmon', edgecolor='black')
plt.title('Distribución de Presión Arterial Diastólica')
plt.xlabel('Presión Arterial Diastólica')
plt.ylabel('Frecuencia')
plt.grid(True)


plt.tight_layout()
plt.show()
```

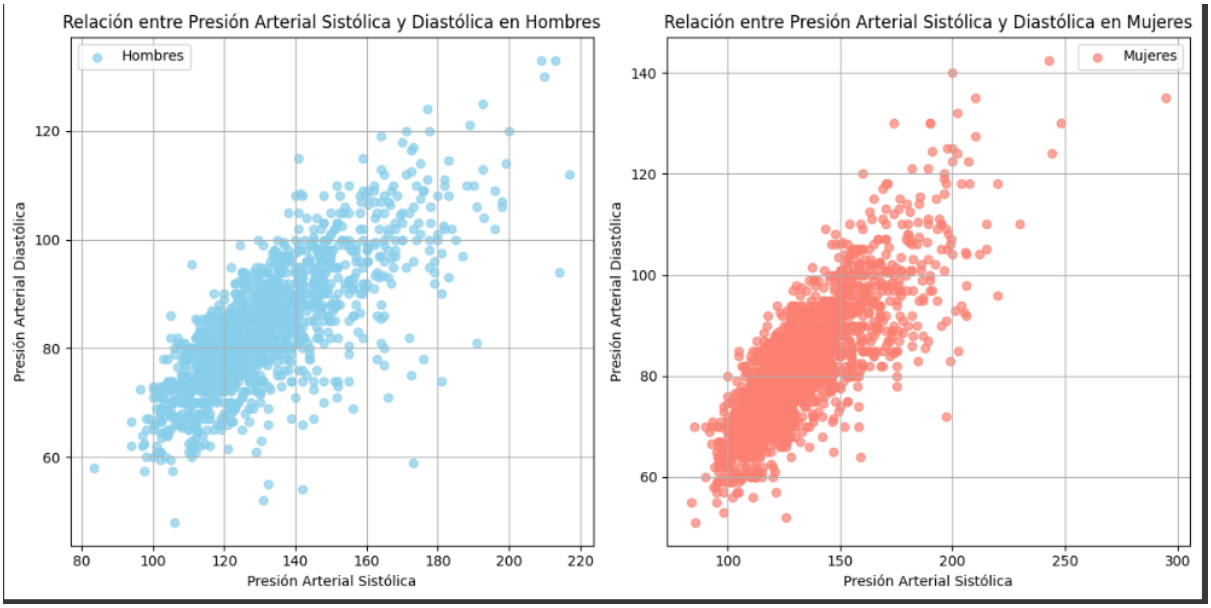


4. ¿Hay alguna correlación entre el género de los pacientes y la prevalencia de enfermedades cardiovasculares?

La correlación entre el género de los pacientes y la prevalencia de enfermedades cardiovasculares es un área de investigación crucial en la salud cardiovascular. Este análisis busca explorar si existe alguna relación entre el género de los pacientes y la probabilidad de desarrollar enfermedades cardiovasculares, como ataques cardíacos o accidentes cerebrovasculares

RELACIÓN ENTRE PRESIÓN ARTERIAL SISTÓLICA Y DIASTÓLICA EN HOMBRES Y MUJERES

```python
import matplotlib.pyplot as plt

# Filtrar datos por género
df_male = df_cat_num[df_cat_num['gender'] == 'Male']
df_female = df_cat_num[df_cat_num['gender'] == 'Female']

plt.figure(figsize=(12, 6))

# Gráfico de dispersión para la relación entre presión arterial sistólica y diastólica en hombres
plt.subplot(1, 2, 1)
plt.scatter(df_male['systolic_blood_pressure'], df_male['diastolic_blood_pressure'], color='skyblue', label='Hombres', alpha=0.7)
plt.title('Relación entre Presión Arterial Sistólica y Diastólica en Hombres')
plt.xlabel('Presión Arterial Sistólica')
plt.ylabel('Presión Arterial Diastólica')
plt.legend()
plt.grid(True)

# Gráfico de dispersión para la relación entre presión arterial sistólica y diastólica en mujeres
plt.subplot(1, 2, 2)
plt.scatter(df_female['systolic_blood_pressure'], df_female['diastolic_blood_pressure'], color='salmon', label='Mujeres', alpha=0.7)
plt.title('Relación entre Presión Arterial Sistólica y Diastólica en Mujeres')
plt.xlabel('Presión Arterial Sistólica')
plt.ylabel('Presión Arterial Diastólica')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```



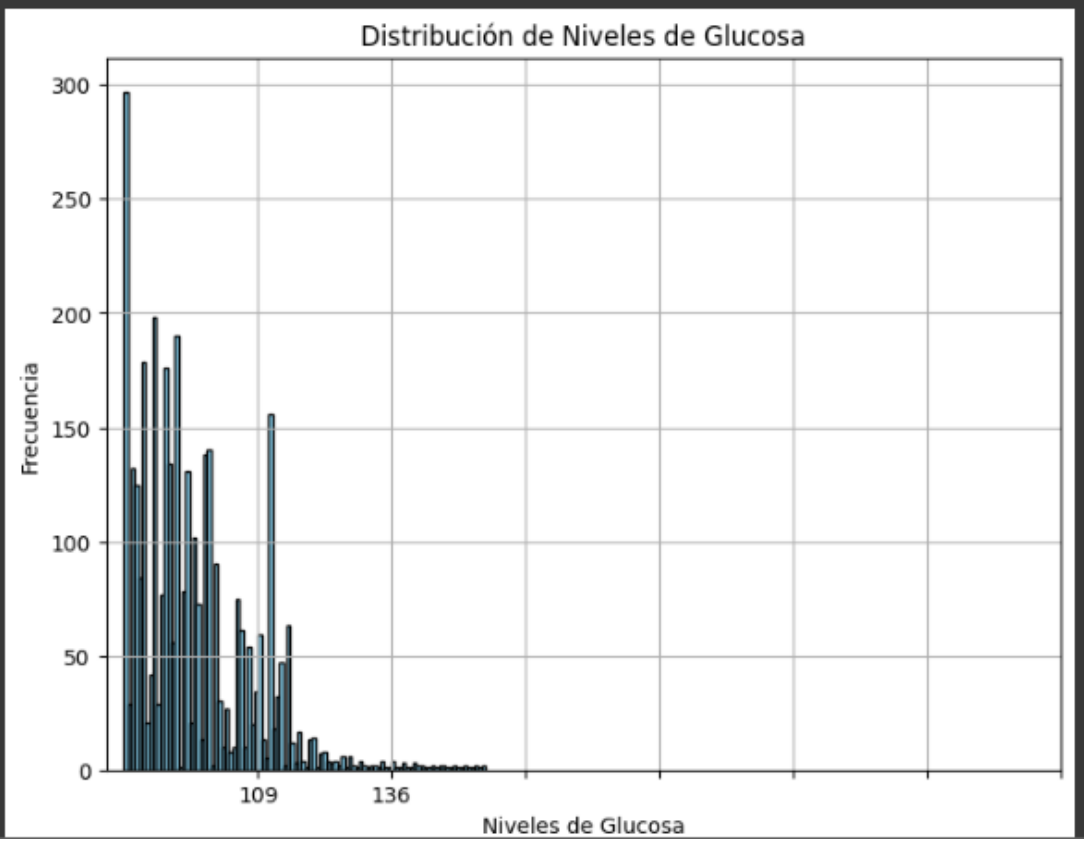
5. ¿Qué género tiene mayores niveles de glucosa en la sangre?

DISTRIBUCIÓN DE LOS NIVELES DE GLUCOSA

En este gráfico de histogramas se aprecia los niveles de glucosa que existe entre hombres y mujeres.

DISTRIBUCIÓN DE LOS NIVELES DE GLUCOSA

```
[ ]  import numpy as np
     # Configurar el tamaño del gráfico
     plt.figure(figsize=(8, 6))

     # Crear el histograma de los niveles de glucosa
     plt.hist(df_cat['glucose'], bins=100, color='skyblue', edgecolor='black')
     plt.grid(True)

     # Agregar etiquetas y título al gráfico
     plt.xlabel('Niveles de Glucosa')
     plt.ylabel('Frecuencia')
     plt.title('Distribución de Niveles de Glucosa')
     plt.xticks(np.arange(50, 400, 50))

     # Mostrar el gráfico
     plt.show()
```



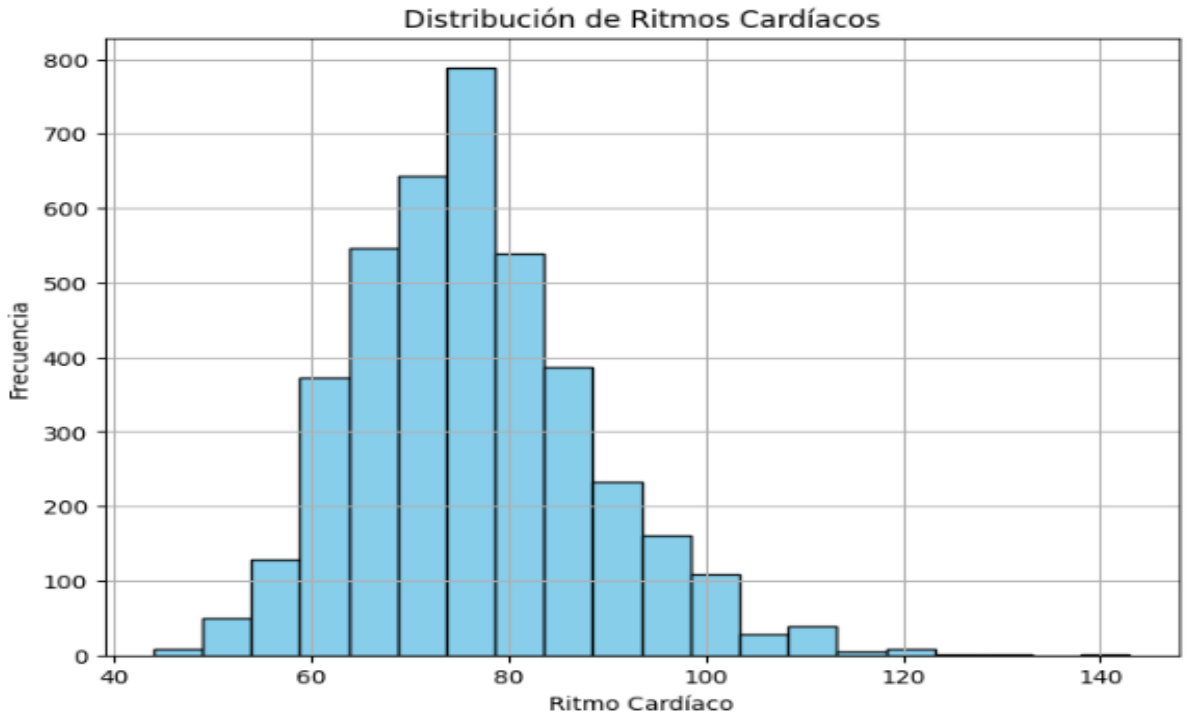
6. ¿Cuántos cigarrillos consumidos por día afectan el nivel acelerado del ritmo cardíaco?

DISTRIBUCIÓN DE RITMOS CARDÍACOS

En este gráfico de histogramas se aprecia la distribución de ritmos cardíacos que tienen los pacientes en donde mayormente predominan los niveles en un intervalo de 60 a 100.

DISTRIBUCIÓN DE RITMOS CARDÍACOS

```
[ ]  # Eliminar filas con valores faltantes en la columna 'heartRate' y convertir a tipo numérico
     df_cleaned = df_num.dropna(subset=['heartRate'])
     df_cleaned['heartRate'] = pd.to_numeric(df_cleaned['heartRate'], errors='coerce')

     # Configurar el tamaño del gráfico
     plt.figure(figsize=(8, 6))

     # Crear el histograma de los ritmos cardíacos
     plt.hist(df_cleaned['heartRate'], bins=20, color='skyblue', edgecolor='black')

     # Agregar cuadrículas al gráfico
     plt.grid(True)

     # Agregar etiquetas y título al gráfico
     plt.xlabel('Ritmo Cardíaco')
     plt.ylabel('Frecuencia')
     plt.title('Distribución de Ritmos Cardíacos')

     # Mostrar el gráfico
     plt.show()
```

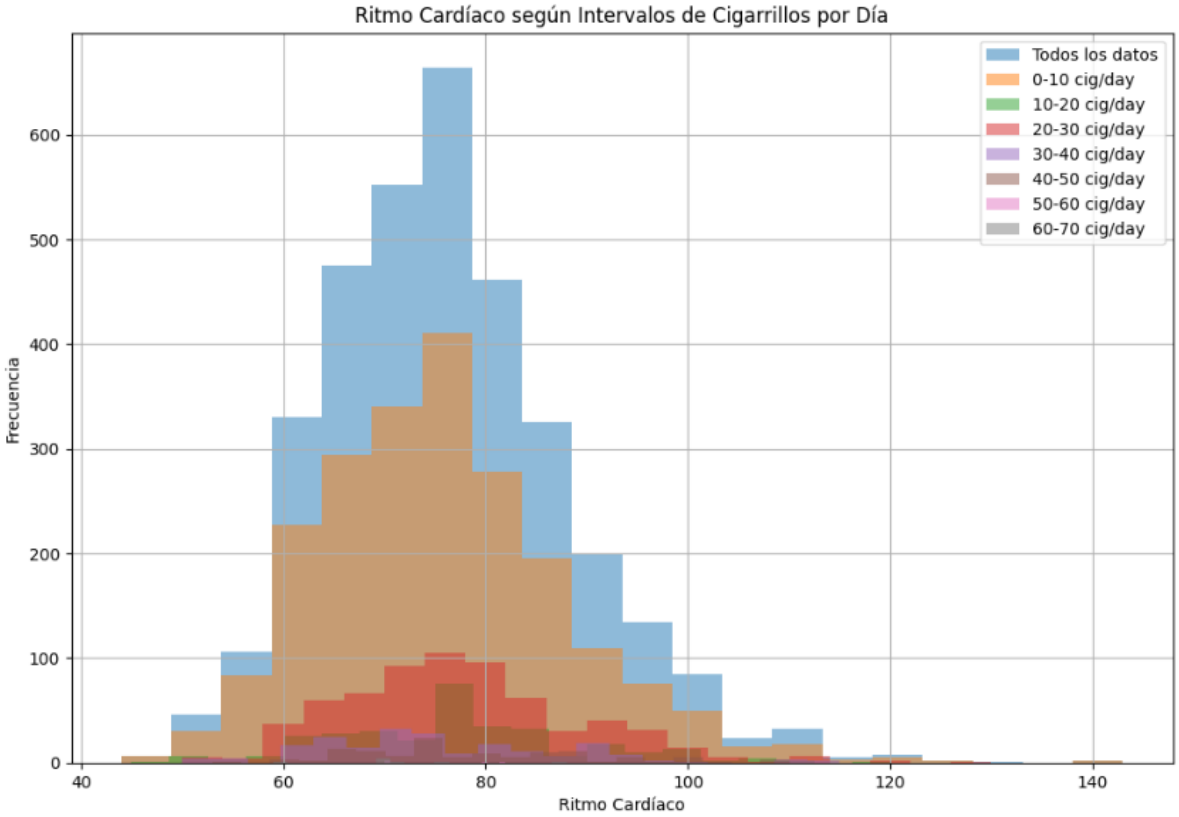


RITMO CARDÍACO SEGÚN INTERVALOS DE CIGARRILLOS POR DÍA

Analizando los datos, observamos que las diferentes capas de color determinan si hay una relación entre la cantidad de cigarrillos fumados y el ritmo cardíaco. Por ejemplo, si las capas de colores más oscuros (que representan un mayor consumo de cigarrillos) tienden a aparecer más en el lado derecho del histograma, esto podría indicar un ritmo cardíaco más alto asociado con un mayor consumo de cigarrillos, lo cual conlleva a una posible situación donde exista una enfermedad cardiovascular.

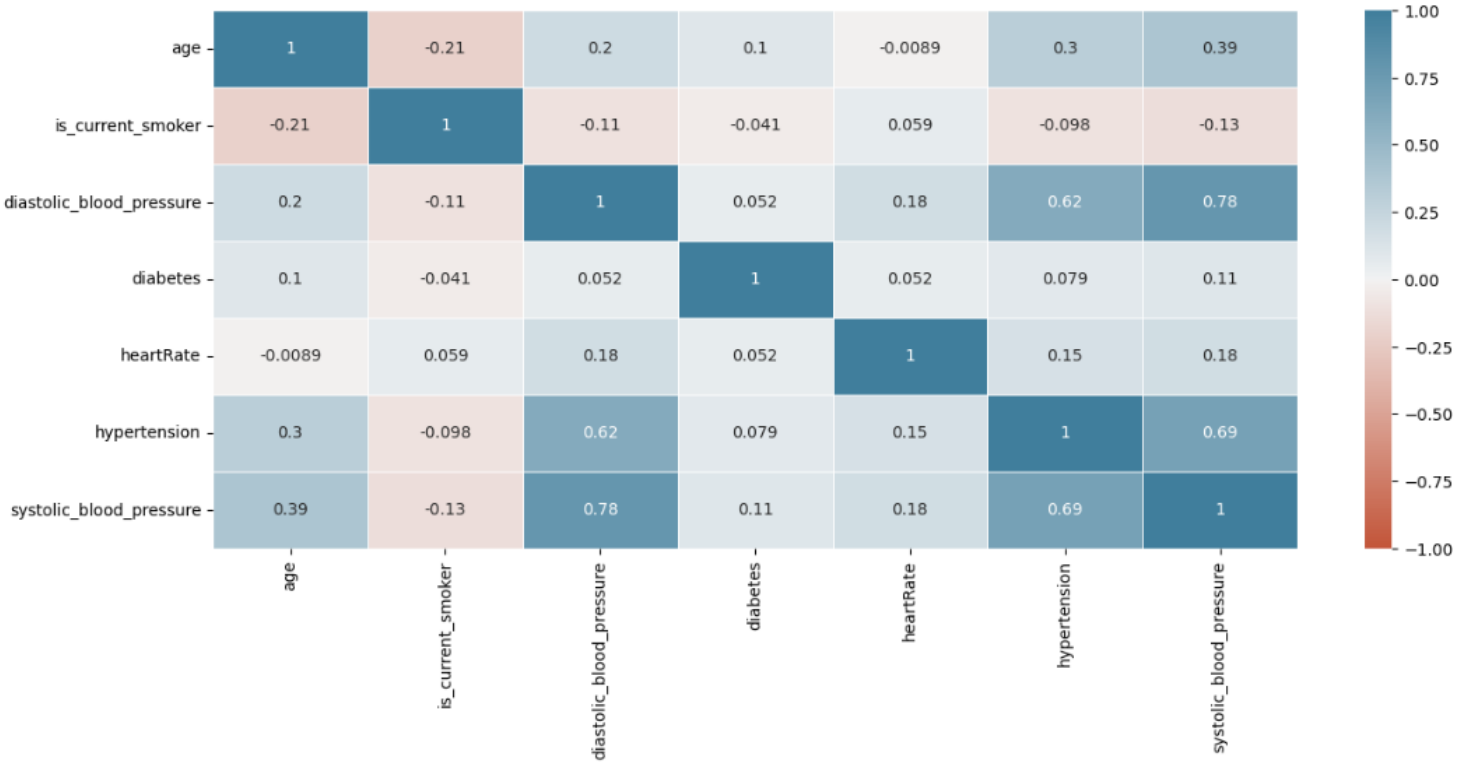
RITMO CARDIACO SEGUN INTERVALOS DE CIGARRILLOS POR DIA

```
[ ]  import numpy as np

     # Eliminar filas con valores faltantes en las columnas de interés
     df_cleaned = df_cat_num.dropna(subset=['heartRate', 'cigarettes_per_day'])
     df_cleaned['heartRate'] = pd.to_numeric(df_cleaned['heartRate'], errors='coerce')
     df_cleaned['cigarettes_per_day'] = pd.to_numeric(df_cleaned['cigarettes_per_day'], errors='coerce')

     # Definir los intervalos para el número de cigarrillos por día
     bins = np.arange(0, df_cleaned['cigarettes_per_day'].max() + 10, 10)  # Intervalos de 0 a máximo + 10, cada 10 cigarrillos

     # Configurar el tamaño del gráfico
     plt.figure(figsize=(12, 8))

     # Crear histogramas para cada intervalo de cigarrillos por día
     plt.hist(df_cleaned['heartRate'], bins=20, alpha=0.5, label='Todos los datos')  # Histograma para todos los datos

     for i in range(len(bins) - 1):
         lower_bound = bins[i]
         upper_bound = bins[i + 1]

         # Filtrar datos dentro del intervalo de cigarrillos por día
         subset_data = df_cleaned[(df_cleaned['cigarettes_per_day'] >= lower_bound) & (df_cleaned['cigarettes_per_day'] < upper_bound)]

         # Crear histograma para el subconjunto de datos
         plt.hist(subset_data['heartRate'], bins=20, alpha=0.5, label=f'{lower_bound}-{upper_bound} cig/day')

     # Agregar cuadrículas al gráfico
     plt.grid(True)

     # Agregar etiquetas y título al gráfico
     plt.xlabel('Ritmo Cardíaco')
     plt.ylabel('Frecuencia')
     plt.title('Ritmo Cardíaco según Intervalos de Cigarrillos por Día')
     plt.legend()

     # Mostrar el gráfico
     plt.show()
```



MATRIZ DE CORRELACIÓN

La matriz de correlación revela relaciones significativas entre varias variables de salud. Observamos que la presión arterial sistólica tiene una correlación positiva fuerte con la presión arterial diastólica (0.78) y con la hipertensión (0.69), indicando que a medida que aumenta la presión arterial diastólica, también lo hace la sistólica y la presencia de hipertensión. La edad también muestra una correlación positiva moderada con la hipertensión (0.30) y la presión arterial sistólica (0.39), sugiriendo que la presión arterial tiende a aumentar con la edad. Por otro lado, el hecho de ser fumador actual tiene una correlación negativa con la edad (-0.21), lo que podría indicar que las personas mayores tienden a fumar menos. También se observa que el ritmo cardíaco tiene una correlación baja con la mayoría de las variables, lo que sugiere que puede estar influenciado por factores no incluidos en esta matriz.



2.4 TRANSFORMACIÓN DE LOS DATOS

Primero importamos las librerías, conectamos con el drive, y dividimos las variables categóricas y numéricas, primero con las categóricas.

TRANSFORMACIÓN DE LOS DATOS

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy as scipy
import seaborn as sns

%matplotlib inline

#Automcompletar rápido
%config IPCompleter.greedy=True
#Desactivar la notación científica
pd.options.display.float_format = '{:.3f}'.format

pd.options.display.max_columns = None
```

```
[ ] df_heart_disease_cleaned = pd.read_pickle('/content/drive/MyDrive/BPA/TP/df_heart_cat_num_fin_EDA.pickle')
    df_heart_disease_cleaned.reset_index(inplace = True)
    df_heart_disease_cleaned.head(3)
```

```
df_cat = df_heart_disease_cleaned.select_dtypes(exclude='number').copy()
df_num = df_heart_disease_cleaned.select_dtypes(include='number').copy()
```

```
df_cat
```

	blood_pressure_medications	gender	heart_disease	cigarettes_per_day	education	glucose	has_prevalent_stroke	total_cholesterol
0	0	Male	No	0	Postgraduate	77	No	195
1	0	Male	No	20	Uneducated	70	No	245
2	0	Female	Yes	30	Graduate	103	No	225
3	0	Female	No	23	Graduate	85	No	285
4	0	Female	No	0	Primaryschool	99	No	228
...	...	...	...	...	...	...	...	...
3456	0	Male	No	0	Graduate	81	No	187
3457	0	Male	Yes	0	Uneducated	79	No	176
3458	0	Male	Yes	1	Uneducated	86	No	313
3459	0	Male	No	43	Graduate	68	No	207
3460	0	Female	No	0	Primaryschool	107	No	269

3461 rows × 8 columns

Vamos a convertir las variables categóricas en variables oneHotEncoder, en la imagen de abajo, mencionamos a la columna llamada gender para poder hacer la transformación.

```
[ ] #convertir variables categóricas en variables oneHotEncoder
    columnas = ['gender']
    df_cat[columnas]
```

	gender
0	Male
1	Male
2	Female
3	Female
4	Female
...	...
3456	Male
3457	Male
3458	Male
3459	Male
3460	Female

3461 rows × 1 columns

Ahora, una vez hecho ese paso, ahora si podemos hacer la transformación de esa variable a oneHotEncoder.

```
from sklearn.preprocessing import OneHotEncoder

#sparse=True, la salida será una matriz / drop='first: Elimina la primera columna / dtype='int64': Asignamos el tipo de dato entero
oneHE = OneHotEncoder(sparse = False, drop='first', dtype='int64')
df_oneHE = oneHE.fit_transform(df_heart_disease_cleaned[columnas])

df_oneHE = pd.DataFrame(data = df_oneHE , columns=oneHE.get_feature_names_out()) #input_features=cat))
df_oneHE
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_output
  warnings.warn(
```

	gender_Male
0	1
1	1
2	0
3	0
4	0
...	...
3456	1
3457	1
3458	1
3459	1
3460	0

3461 rows × 1 columns

Repetimos los mismos pasos para los siguientes procedimientos.

```
columnas = ['education']
df_cat[columnas]
```

	education
0	Postgraduate
1	Uneducated
2	Graduate
3	Graduate
4	Primaryschool
...	...
3456	Graduate
3457	Uneducated
3458	Uneducated
3459	Graduate
3460	Primaryschool

3461 rows × 1 columns

Ahora podemos hacer la transformación de la variable a OneHotEnconder.

```
#convertir variables categóricas en variables dummy
from sklearn.preprocessing import OneHotEncoder

#sparse=True, la salida será una matriz / drop='first: Elimina la primera columna / dtype='int64': Asignamos el tipo de dato entero
oneHE = OneHotEncoder(sparse = False, drop='first', dtype='int64')
df_oneHE = oneHE.fit_transform(df_heart_disease_cleaned[columnas])

df_oneHE = pd.DataFrame(data = df_oneHE , columns=oneHE.get_feature_names_out()) #input_features=cat))
df_oneHE
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_outpu
  warnings.warn(
```

	education_Postgraduate	education_Primaryschool	education_Uneducated
0	1	0	0
1	0	0	1
2	0	0	0
3	0	0	0
4	0	1	0
...	...	...	...
3456	0	0	0
3457	0	0	1
3458	0	0	1
3459	0	0	0
3460	0	1	0

3461 rows × 3 columns

Hacemos la misma transformación de oneHotEncoder a la variable has_prevalent_stroke.

```
#convertir variables categóricas en variables dummy
from sklearn.preprocessing import OneHotEncoder

#sparse=True, la salida será una matriz / drop='first: Elimina la primera columna / dtype='int64': Asignamos el tipo de dato entero
oneHE = OneHotEncoder(sparse = False, drop='first', dtype='int64')
df_oneHE = oneHE.fit_transform(df_heart_disease_cleaned[columnas])

df_oneHE = pd.DataFrame(data = df_oneHE , columns=oneHE.get_feature_names_out()) #input_features=cat))
df_oneHE
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_outpu
  warnings.warn(
```

	has_prevalent_stroke_Yes
0	0
1	0
2	0
3	0
4	0
...	...
3456	0
3457	0
3458	0
3459	0
3460	0

Lo mismo para la variable dependiente heart_disease.

```
#convertir variables categóricas en variables dummy
from sklearn.preprocessing import OneHotEncoder

#sparse=True, la salida será una matriz / drop='first: Elimina la primera columna / dtype='int64': Asignamos el tipo de dato entero
oneHE = OneHotEncoder(sparse = False, drop='first', dtype='int64')
df_oneHE = oneHE.fit_transform(df_heart_disease_cleaned[columnas])

df_oneHE = pd.DataFrame(data = df_oneHE , columns=oneHE.get_feature_names_out()) #input_features=cat))
df_oneHE
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_outpu
  warnings.warn(
```

	heart_disease_Yes
0	0
1	0
2	1
3	0
4	0
...	...
3456	0
3457	1
3458	1
3459	0
3460	0

Ahora, vamos con las variables numéricas, vamos a identificarlas.

```
df_num
```

	index	body_mass_index	age	is_current_smoker	diastolic_blood_pressure	diabetes	heartRate	hypertension	systolic_blood_pressure
0	0	26.970	39.000	0.000	70.000	0.000	80.000	0.000	106.000
1	2	25.340	48.000	1.000	80.000	0.000	75.000	0.000	127.500
2	3	28.580	61.000	1.000	95.000	0.000	65.000	1.000	150.000
3	4	23.100	46.000	1.000	84.000	0.000	85.000	0.000	130.000
4	5	30.300	43.000	0.000	110.000	0.000	77.000	1.000	180.000
...	...	...	...	...	...	...	...	...	...
3456	4198	24.960	58.000	0.000	81.000	0.000	80.000	1.000	141.000
3457	4199	23.140	68.000	0.000	97.000	0.000	60.000	1.000	168.000
3458	4200	25.970	50.000	1.000	92.000	0.000	66.000	1.000	179.000
3459	4201	19.710	51.000	1.000	80.000	0.000	65.000	0.000	126.500
3460	4204	21.470	52.000	0.000	83.000	0.000	80.000	0.000	133.500

3461 rows × 9 columns

Ahora, realizamos la transformación de las variables numércias a MinMaxScaler,

```
from sklearn.preprocessing import MinMaxScaler

columnas = ['age','blood_pressure_medications','body_mass_index','cigarettes_per_day',
            'diabetes','diastolic_blood_pressure','glucose','heartRate', 'hypertension','is_current_smoker','systolic_blood_pressure','total_cholesterol']

mms = MinMaxScaler()
mms.fit(df_heart_disease_cleaned[columnas])
df_mms = mms.transform(df_heart_disease_cleaned[columnas])

df_mms = pd.DataFrame(data=df_mms, columns=columnas)
df_mms
```

	age	blood_pressure_medications	body_mass_index	cigarettes_per_day	diabetes	diastolic_blood_pressure	glucose	heartRate	hypertension	is_current_smoker	systolic_blood_pressure	total_cholesterol
0	0.184	0.000	0.277	0.000	0.000	0.233	0.105	0.364	0.000	0.000	0.106	0.158
1	0.421	0.000	0.238	0.286	0.000	0.339	0.085	0.313	0.000	1.000	0.208	0.262
2	0.763	0.000	0.316	0.429	0.000	0.497	0.178	0.212	1.000	1.000	0.314	0.220
3	0.368	0.000	0.183	0.329	0.000	0.381	0.127	0.414	0.000	1.000	0.220	0.345
4	0.289	0.000	0.358	0.000	0.000	0.656	0.167	0.333	1.000	0.000	0.456	0.227
...	...	...	...	...	...	...	...	...	...	...	...	...
3456	0.684	0.000	0.228	0.000	0.000	0.349	0.116	0.364	1.000	0.000	0.272	0.141
3457	0.947	0.000	0.184	0.000	0.000	0.519	0.110	0.162	1.000	0.000	0.400	0.119
3458	0.474	0.000	0.253	0.014	0.000	0.466	0.130	0.222	1.000	1.000	0.452	0.403
3459	0.500	0.000	0.101	0.614	0.000	0.339	0.079	0.212	0.000	1.000	0.203	0.183
3460	0.526	0.000	0.144	0.000	0.000	0.370	0.189	0.364	0.000	0.000	0.236	0.312

3461 rows × 12 columns

Ahora vamos a hacer la integración de dataframes, que va a imprimir los dataframes que contienen las variables categóricas y númericas.

```
print(df_oneHE)
print(df_mms)
```

```
      heart_disease_Yes
0                     0
1                     0
2                     1
3                     0
4                     0
...                 ...
3456                  0
3457                  1
3458                  1
3459                  0
3460                  0

[3461 rows x 1 columns]
        age  blood_pressure_medications  body_mass_index  cigarettes_per_day  \
0     0.184                       0.000            0.277               0.000
1     0.421                       0.000            0.238               0.286
2     0.763                       0.000            0.316               0.429
3     0.368                       0.000            0.183               0.329
4     0.289                       0.000            0.358               0.000
...     ...                         ...              ...                 ...
3456  0.684                       0.000            0.228               0.000
3457  0.947                       0.000            0.184               0.000
3458  0.474                       0.000            0.253               0.014
3459  0.500                       0.000            0.101               0.614
3460  0.526                       0.000            0.144               0.000

      diabetes  diastolic_blood_pressure  glucose  heartRate  hypertension  \
0        0.000                     0.233    0.105      0.364         0.000
1        0.000                     0.339    0.085      0.313         0.000
2        0.000                     0.497    0.178      0.212         1.000
3        0.000                     0.381    0.127      0.414         0.000
4        0.000                     0.656    0.167      0.333         1.000
...        ...                       ...      ...        ...           ...
```

Ahora, vamos a concatenar los dataframes.

```
df_transformationdata = pd.concat([df_oneHE,df_mms],axis=1)
```

Mostramos todas las variables ya transformadas en un nuevo dataframe.

df_transformationdata

	heart_disease_Yes	age	blood_pressure_medications	body_mass_index	cigarettes_per_day	diabetes	diastolic_blood_pressure	glucose	heartRate	hypertension	is_current_smoker	systolic_blood_pressure	total_cholesterol
0	0	0.184	0.000	0.277	0.000	0.000	0.233	0.105	0.364	0.000	0.000	0.106	0.158
1	0	0.421	0.000	0.238	0.286	0.000	0.339	0.085	0.313	0.000	1.000	0.208	0.262
2	1	0.763	0.000	0.316	0.429	0.000	0.497	0.178	0.212	1.000	1.000	0.314	0.220
3	0	0.368	0.000	0.183	0.329	0.000	0.381	0.127	0.414	0.000	1.000	0.220	0.345
4	0	0.289	0.000	0.358	0.000	0.000	0.656	0.167	0.333	1.000	0.000	0.456	0.227
...	...	...	...	...	...	...	...	...	...	...	...	...	...
3456	0	0.684	0.000	0.228	0.000	0.000	0.349	0.116	0.364	1.000	0.000	0.272	0.141
3457	1	0.947	0.000	0.184	0.000	0.000	0.519	0.110	0.162	1.000	0.000	0.400	0.119
3458	1	0.474	0.000	0.253	0.014	0.000	0.466	0.130	0.222	1.000	1.000	0.452	0.403
3459	0	0.500	0.000	0.101	0.614	0.000	0.339	0.079	0.212	0.000	1.000	0.203	0.183
3460	0	0.526	0.000	0.144	0.000	0.000	0.370	0.189	0.364	0.000	0.000	0.236	0.312

3461 rows × 13 columns

Por último, haremos el paso de serialización, de esa forma permite poder almacenar y recuperar los datos del dataframe en su estado original.

```
df_transformationdata.to_pickle('/content/drive/My Drive/BPA/TP/df_transformationdata_transformacionheart.pickle')
```

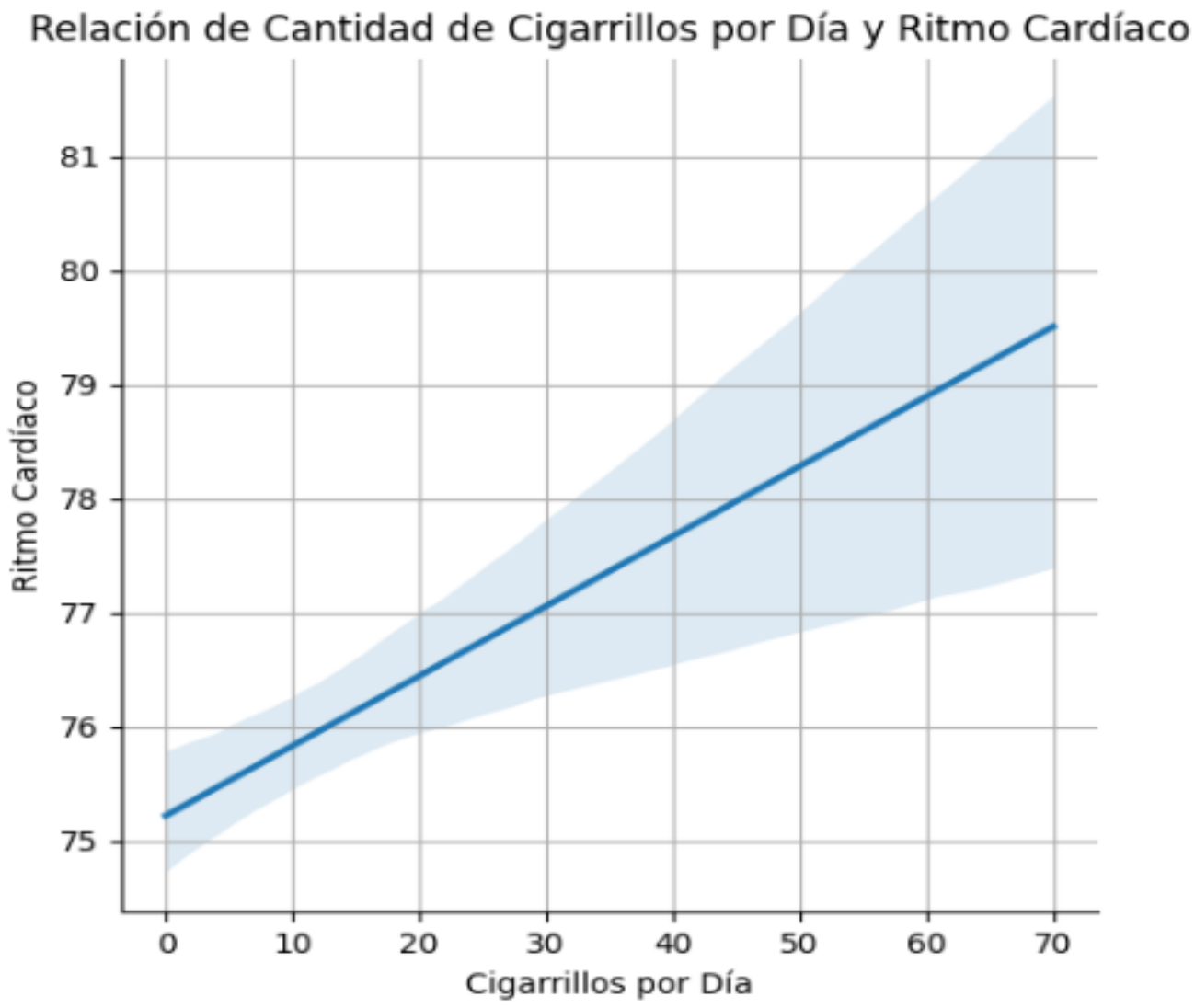
CAPÍTULO 3: RESULTADOS SOBRE ANÁLISIS DE DATOS

3.1 INSIGHTS

1) **Relación de tendencia de la cantidad de cigarrillos consumidos por día y ritmo cardíaco**

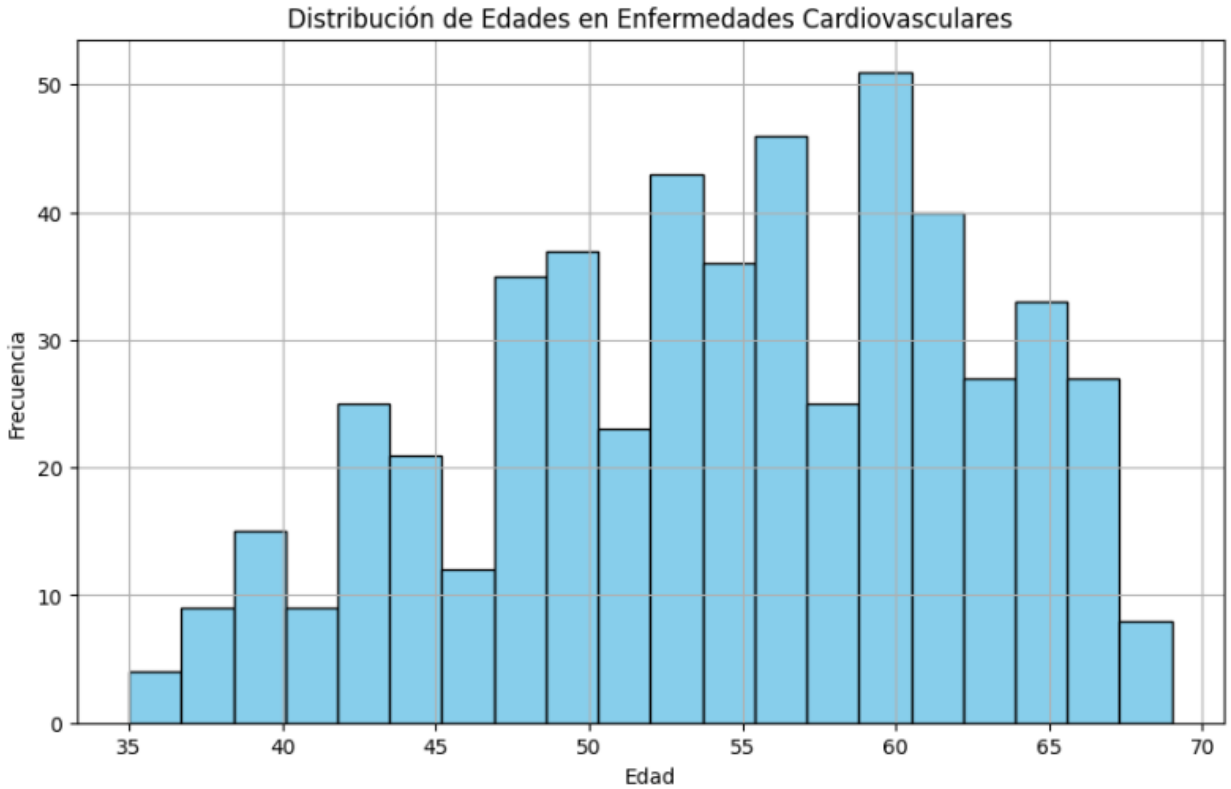
Este gráfico sugiere que hay una relación positiva entre la cantidad de cigarrillos fumados y el ritmo cardíaco, lo que significa que a medida que la cantidad de cigarrillos fumados por día aumenta, también lo hace el ritmo cardíaco promedio.

```python
import pandas as pd

# Convertir la columna 'cigarettes_per_day' a numérico, tratando errores
df_heart_disease_cleaned['cigarettes_per_day'] = pd.to_numeric(df_heart_disease_cleaned['cigarettes_per_day'], errors='coerce')

# Verificar los nuevos tipos de datos
print(df_heart_disease_cleaned['cigarettes_per_day'].dtype)

# Ahora puedes intentar crear el gráfico de nuevo, asegurándote que no hay valores NaN que puedan afectar la regresión
import seaborn as sns
import matplotlib.pyplot as plt

# Crear un gráfico de tendencia entre 'cigarettes_per_day' y 'heartRate'
sns.lmplot(x='cigarettes_per_day', y='heartRate', data=df_heart_disease_cleaned, scatter=False)

# Añadir título y etiquetas de ejes
plt.title('Relación de Cantidad de Cigarrillos por Día y Ritmo Cardíaco')
plt.xlabel('Cigarrillos por Día')
plt.ylabel('Ritmo Cardíaco')

# Agregar cuadrículas al gráfico
plt.grid(True)

# Mostrar el gráfico
plt.show()
```



2) Distribución de las edades de las personas afectadas por enfermedades cardiovasculares

En el gráfico que muestra la distribución de edades de las personas afectadas por enfermedades cardiovasculares en nuestro dataset, se destaca que la edad más común entre los afectados es de 60 años, mientras que la edad menos frecuente es de 36 años. Esta representación visual nos permite identificar claramente las edades con mayor incidencia de enfermedades cardiovasculares en nuestra muestra de datos. Al analizar detenidamente la distribución, se observa que la prevalencia de estas enfermedades tiende a aumentar con la edad, siendo más frecuentes en individuos de mayor edad. Este patrón sugiere una posible correlación entre la edad y la aparición de enfermedades cardiovasculares, destacando la importancia de la prevención y el cuidado de la salud cardiovascular en la población de mayor edad.

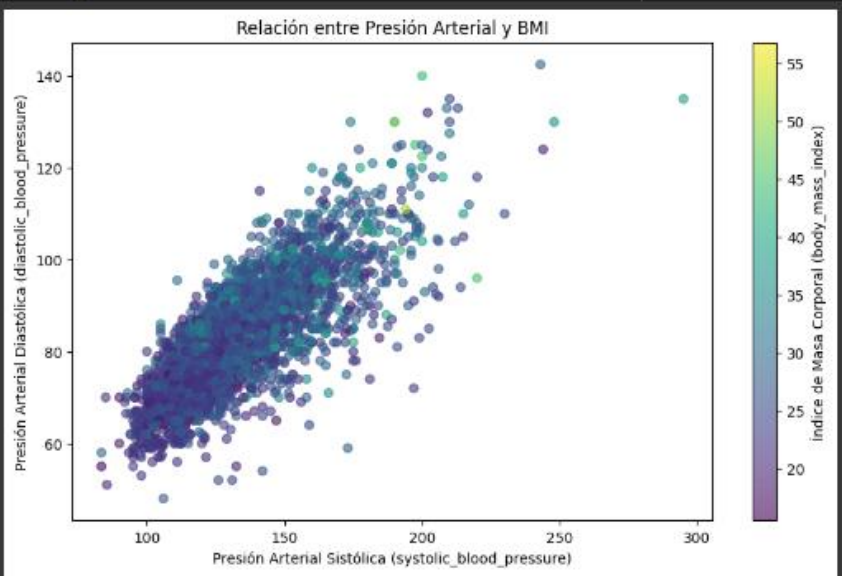


3) Relación de los tipos de presiones arteriales y el Índice de Masa

Explorar la relación entre los diferentes tipos de presión arterial, tanto sistólica como diastólica, y el índice de masa corporal (BMI) nos permite comprender cómo la distribución del peso corporal está asociada con la salud cardiovascular. El BMI es una medida comúnmente utilizada para evaluar el peso relativo en relación con la altura de una persona, mientras que la presión arterial es un indicador crítico de la salud del sistema circulatorio. Al examinar esta relación, podemos identificar si existe alguna correlación entre el peso corporal y los niveles de presión arterial

3) Relación de los tipos de presiones arteriales y el Índice de Masa

```
# Eliminar filas con valores "na" en la columna 'BMI'
df_cleaned = df_heart_disease_cleaned[df_heart_disease_cleaned['body_mass_index'] != 'na']

# Convertir la columna 'BMI' a números (float)
df_cleaned['body_mass_index'] = df_cleaned['body_mass_index'].astype(float)

# Crear el gráfico de dispersión
plt.figure(figsize=(10, 6))
plt.scatter(df_cleaned['systolic_blood_pressure'], df_cleaned['diastolic_blood_pressure'], c=df_cleaned['body_mass_index'], cmap='viridis', alpha=0.6)

# Añadir etiquetas y título
plt.xlabel('Presión Arterial Sistólica (systolic_blood_pressure)')
plt.ylabel('Presión Arterial Diastólica (diastolic_blood_pressure)')
plt.title('Relación entre Presión Arterial y BMI')

# Añadir barra de colores para el índice de masa corporal (BMI)
plt.colorbar(label='Índice de Masa Corporal (body_mass_index)')

# Mostrar el gráfico
plt.show()
```

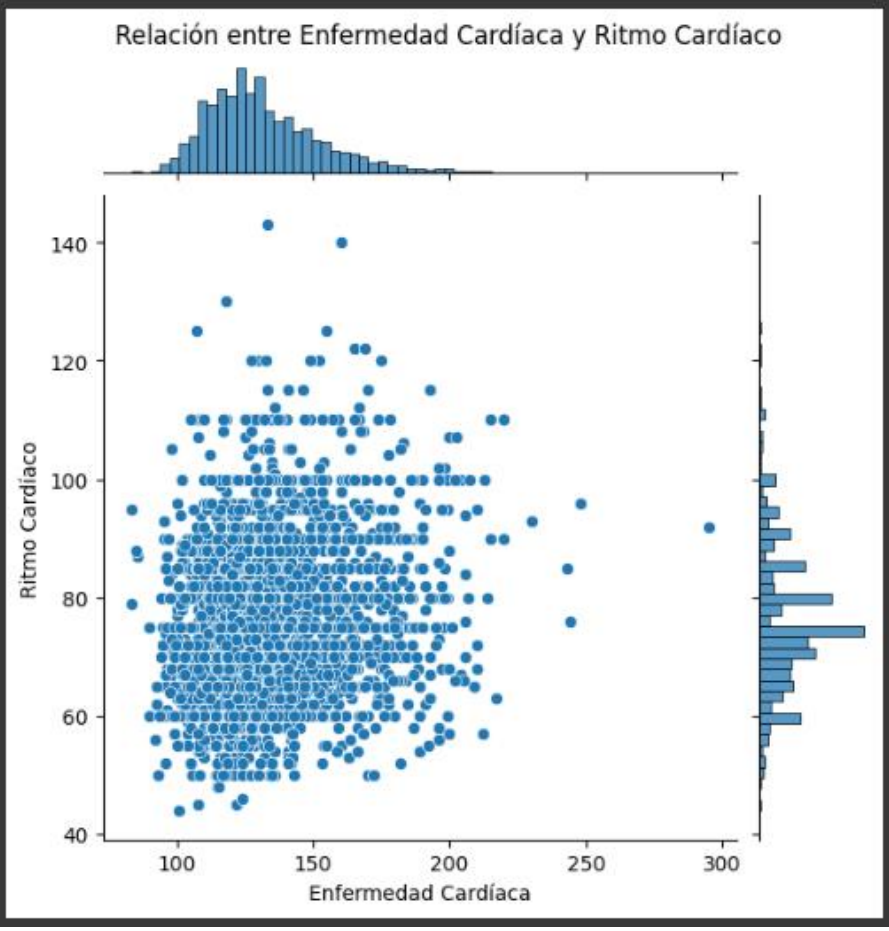


4) Relación entre Enfermedad Cardíaca y Ritmo Cardíaco

Se observa una tendencia general de aumento de la presión arterial con el aumento del IMC. Esto significa que las personas con un IMC más alto tienden a tener una presión arterial más alta. La tendencia es más pronunciada en la presión arterial sistólica (la presión cuando el corazón late) que en la presión arterial diastólica (la presión cuando el corazón está en reposo).

4) Relación entre Enfermedad Cardíaca y Ritmo Cardíaco

```
[ ]  # Crear un gráfico de dispersión con histogramas utilizando sns.jointplot
     sns.jointplot(data=df_heart_disease_cleaned, x='systolic_blood_pressure', y='heartRate', kind='scatter', height=6)
     plt.xlabel('Enfermedad Cardíaca')
     plt.ylabel('Ritmo Cardíaco')
     plt.suptitle('Relación entre Enfermedad Cardíaca y Ritmo Cardíaco', y=1.02)
     plt.show()
```



CAPÍTULO 4: RESULTADOS MODELIZACIÓN Y OPTIMIZACIÓN

4.1. MODELIZACIÓN

Primero, debemos comparar dos modelos para ver quién es el que obtiene los mejores puntajes en las métricas que se obtienen, de ahí, vamos a elegir al modelo que tiene mayores puntajes que el otro modelo, lo optimizamos y presentamos su resultados finales de ese modelo ya optimizado.

Vamos a empezar con la “Modelización Random Forest con todas las variables”.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

#Automcompletar rápido
%config IPCompleter.greedy=True
#Desactivar la notación científica
pd.options.display.float_format = '{:.3f}'.format
```

```
df_heart_modelizacion = pd.read_pickle('/content/drive/My Drive/BPA/TP/df_transformationdata_transformacionheart.pickle')
df_heart_modelizacion
```

	heart_disease_Yes	age	blood_pressure_medications	body_mass_index	cigarettes_per_day	diabetes	diastolic_blood_pressure	glucose	heartRate	hypertension	is_current_smoker	systolic_blood_pressure	total_cholesterol
0	0	0.184	0.000	0.277	0.000	0.000	0.233	0.105	0.364	0.000	0.000	0.106	0.158
1	0	0.421	0.000	0.238	0.286	0.000	0.339	0.085	0.313	0.000	1.000	0.208	0.262
2	1	0.763	0.000	0.316	0.429	0.000	0.497	0.178	0.212	1.000	1.000	0.314	0.220
3	0	0.368	0.000	0.183	0.329	0.000	0.381	0.127	0.414	0.000	1.000	0.220	0.345
4	0	0.289	0.000	0.358	0.000	0.000	0.656	0.167	0.333	1.000	0.000	0.456	0.227
...	...	...	...	...	...	...	...	...	...	...	...	...	...
3456	0	0.684	0.000	0.228	0.000	0.000	0.349	0.116	0.364	1.000	0.000	0.272	0.141
3457	1	0.947	0.000	0.184	0.000	0.000	0.519	0.110	0.162	1.000	0.000	0.400	0.119
3458	1	0.474	0.000	0.253	0.014	0.000	0.466	0.130	0.222	1.000	1.000	0.452	0.403
3459	0	0.500	0.000	0.101	0.614	0.000	0.339	0.079	0.212	0.000	1.000	0.203	0.183
3460	0	0.526	0.000	0.144	0.000	0.000	0.370	0.189	0.364	0.000	0.000	0.236	0.312

3461 rows × 13 columns

Se define las variables independientes y la variable dependiente (target) llamado heart_disease.

```
df_heart_modelizacion.shape
```

```
(3461, 13)
```

```
X = df_heart_modelizacion.drop(['heart_disease_Yes'], axis = 1)
y = df_heart_modelizacion['heart_disease_Yes']
```

```
X
```

	age	blood_pressure_medications	body_mass_index	cigarettes_per_day	diabetes	diastolic_blood_pressure	glucose	heartRate	hypertension	is_current_smoker	systolic_blood_pressure	total_cholesterol
0	0.184	0.000	0.277	0.000	0.000	0.233	0.105	0.364	0.000	0.000	0.106	0.158
1	0.421	0.000	0.238	0.286	0.000	0.339	0.085	0.313	0.000	1.000	0.208	0.262
2	0.763	0.000	0.316	0.429	0.000	0.497	0.178	0.212	1.000	1.000	0.314	0.220
3	0.368	0.000	0.183	0.329	0.000	0.381	0.127	0.414	0.000	1.000	0.220	0.345
4	0.289	0.000	0.358	0.000	0.000	0.656	0.167	0.333	1.000	0.000	0.456	0.227
...	...	...	...	...	...	...	...	...	...	...	...	...
3456	0.684	0.000	0.228	0.000	0.000	0.349	0.116	0.364	1.000	0.000	0.272	0.141
3457	0.947	0.000	0.184	0.000	0.000	0.519	0.110	0.162	1.000	0.000	0.400	0.119
3458	0.474	0.000	0.253	0.014	0.000	0.466	0.130	0.222	1.000	1.000	0.452	0.403
3459	0.500	0.000	0.101	0.614	0.000	0.339	0.079	0.212	0.000	1.000	0.203	0.183
3460	0.526	0.000	0.144	0.000	0.000	0.370	0.189	0.364	0.000	0.000	0.236	0.312

3461 rows × 12 columns

```
[ ] y

0       0
1       0
2       1
3       0
4       0
       ..
3456    0
3457    1
3458    1
3459    0
3460    0
Name: heart_disease_Yes, Length: 3461, dtype: int64
```

Ahora vamos a importar la librería “train_test_split” para poder hacer las variables de entrenamiento y testeo.

```
from sklearn.model_selection import train_test_split

[ ] X_train,X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

[ ] print(X_train.shape)
    print(X_test.shape)
    print(y_train.shape)
    print(y_test.shape)

(2768, 12)
(693, 12)
(2768,)
(693,)
```

Importamos el modelo “RandomForestClassifier” para poder instanciarlo y entrenarlo.

```
#Importamos el modelo
from sklearn.ensemble import RandomForestClassifier
```

```
#Instanciamos el modelo
modelo_RandomForest = RandomForestClassifier()
```

```
#Entrenamos el modelo
modelo_RandomForest.fit(X_train,y_train)
```

```
RandomForestClassifier
RandomForestClassifier()
```

Ahora, debemos predecir, importar las métricas, ejecutar la matriz de confusión y ver los resultados de las métricas.

```
#Predecimos usando los datos de test
y_pred_RandomForest       = modelo_RandomForest.predict(X_test)
y_pred_RandomForest_proba = modelo_RandomForest.predict_proba(X_test)
#y_pred_RandomForest_proba[:,1][:10]
```

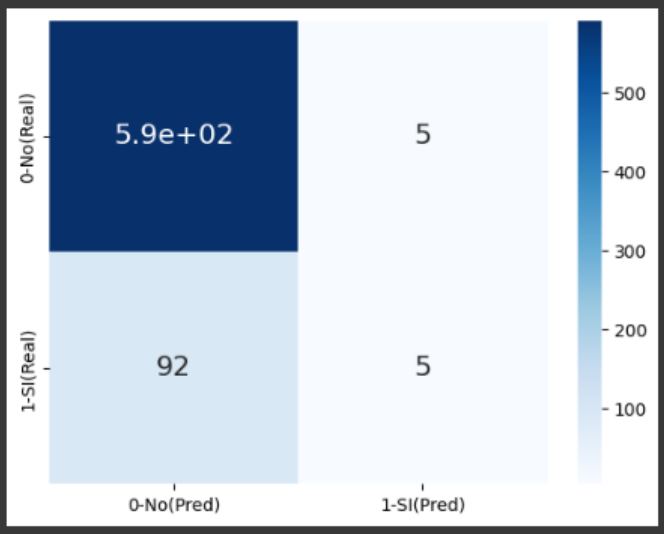
```
#Importamos las métricas
from sklearn.metrics import roc_auc_score,confusion_matrix,f1_score,classification_report,\
                            accuracy_score,precision_score,recall_score
```

```
confusion_matrix_RandomForest = confusion_matrix(y_test,y_pred_RandomForest)
```

```
#Mostrar el confusion matrix como dataframe
pd.DataFrame(confusion_matrix_RandomForest,columns = ['Prediccion NO','Prediccion SI'], index = ['Real NO','Real SI'])
```

	Prediccion NO	Prediccion SI
Real NO	591	5
Real SI	92	5

```
[ ] #Mostrar el confusion matrix como grafico
    sns.heatmap(confusion_matrix_RandomForest, annot=True,xticklabels=['0-No(Pred)','1-SI(Pred)'],
                yticklabels=['0-No(Real)','1-SI(Real)'], cmap = 'Blues',annot_kws={"size": 16});
```




```
[ ] acc_RandomForest  = accuracy_score(y_test,y_pred_RandomForest)
    f1_RandomForest   = f1_score(y_test,y_pred_RandomForest)
    prec_RandomForest = precision_score(y_test, y_pred_RandomForest)
    rec_RandomForest  = recall_score(y_test, y_pred_RandomForest)
    auc_RandomForest  = roc_auc_score(y_test,y_pred_RandomForest_proba[:,1])
```

```
[ ] results = pd.DataFrame([['Random Forest', acc_RandomForest,f1_RandomForest,prec_RandomForest,rec_RandomForest,auc_RandomForest]],
                           columns = ['Model','Accuracy','F1','Precision','Recall','AUC'])
```

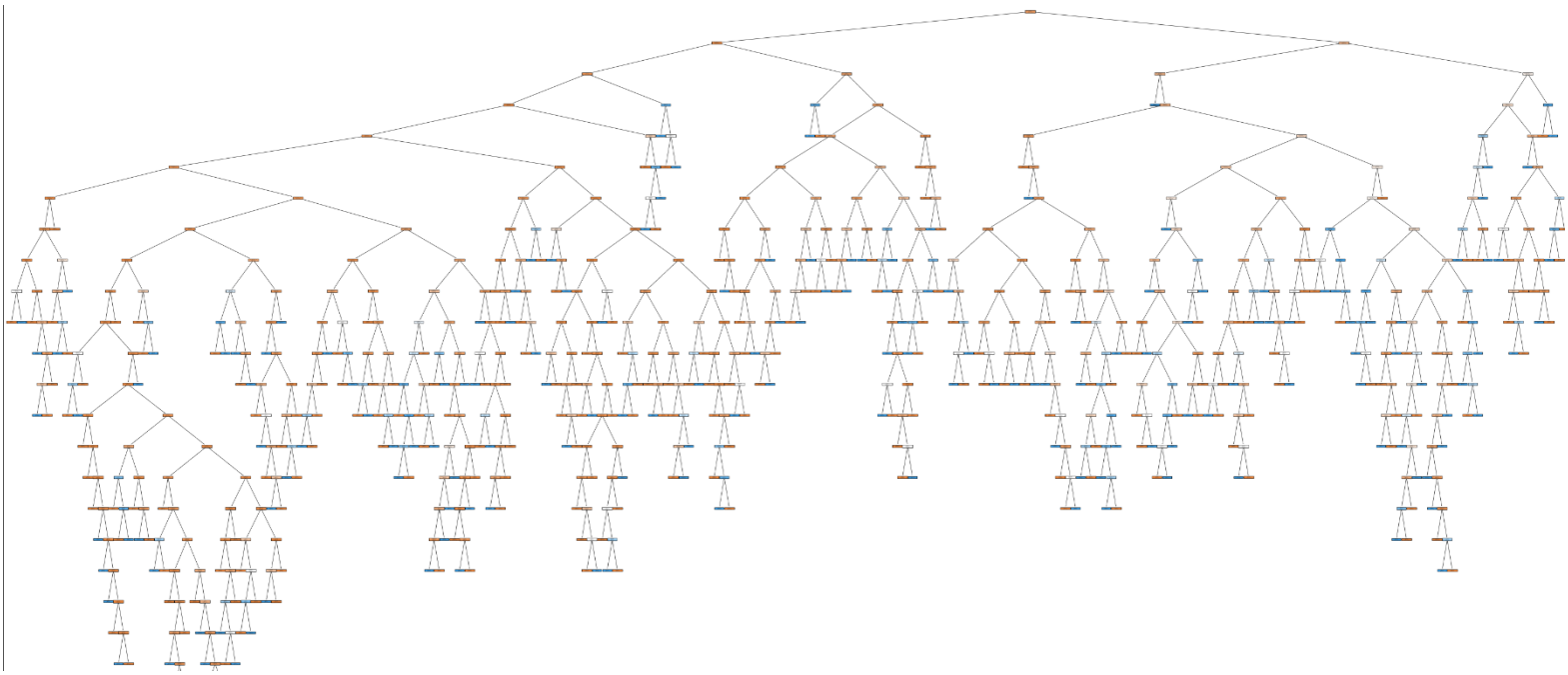
Aquí podemos ver los puntajes de las métricas obtenidas tras ejecutar el “Modelo de Random Forest con todas las variables”.

```
results
```

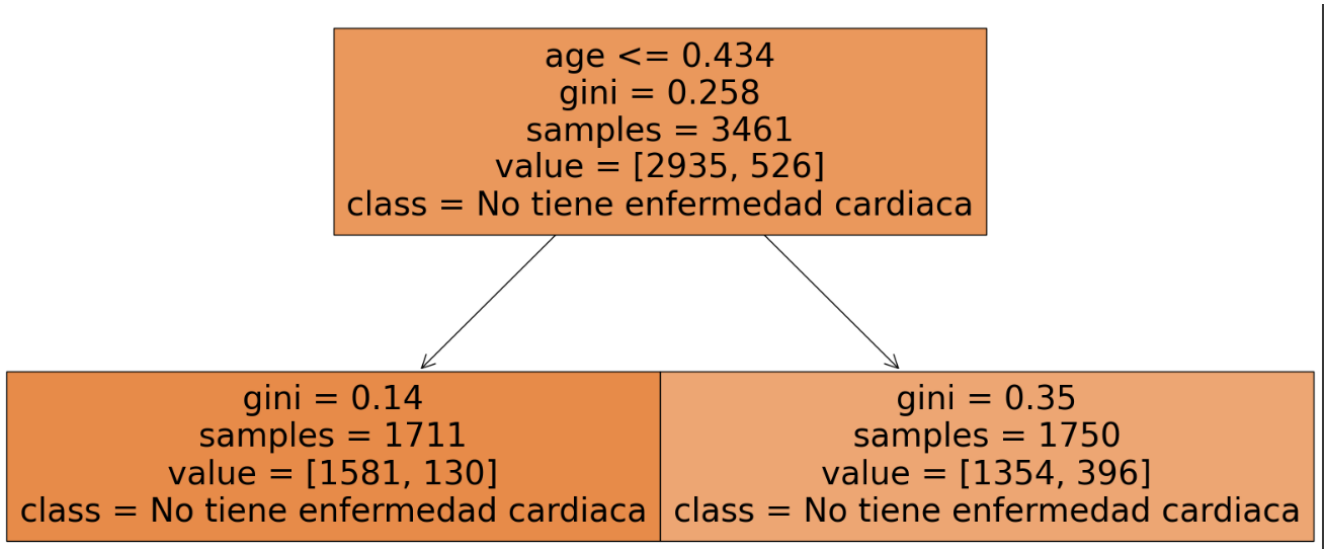
	Model	Accuracy	F1	Precision	Recall	AUC
0	Random Forest	0.860	0.093	0.500	0.052	0.719

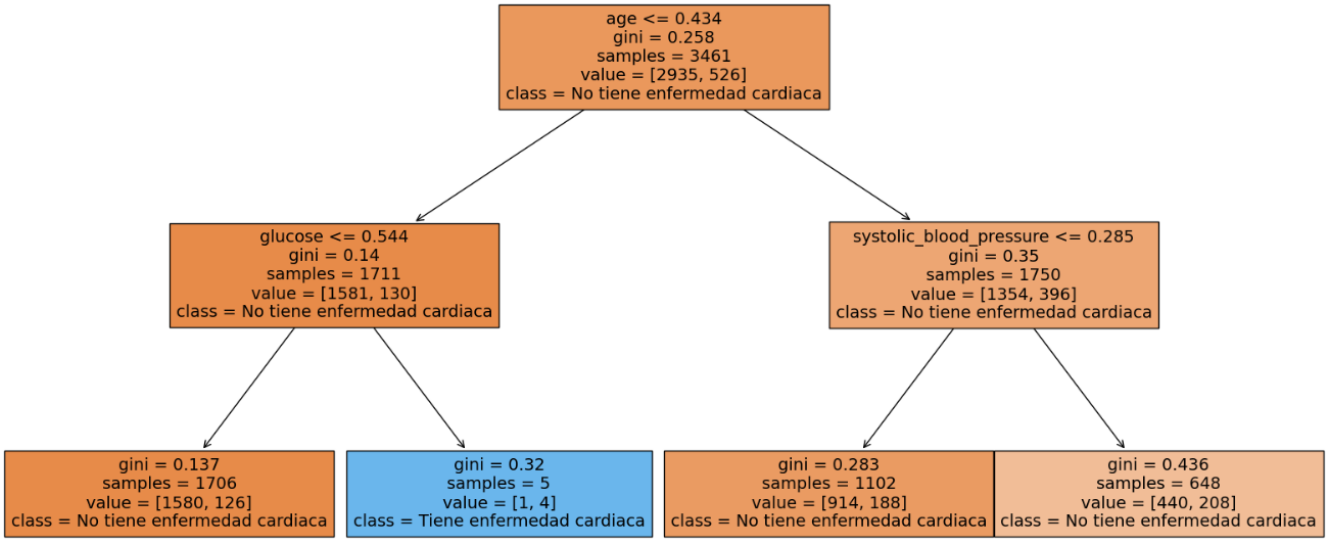
Ahora, vamos a generar los árboles de decisión usando “Random Forest”, nos va a decir que número de árboles de decisión se va a generar, las variables usadas, definimos el índice del árbol, importamos la librería “tree” y vamos a visualizar todos los árboles de decisión generados (La primera imagen de los árboles de decisión no se podrá visualizar bien por la cantidad generada, pero mostraremos más de cerca en las siguientes imágenes después de la primera).

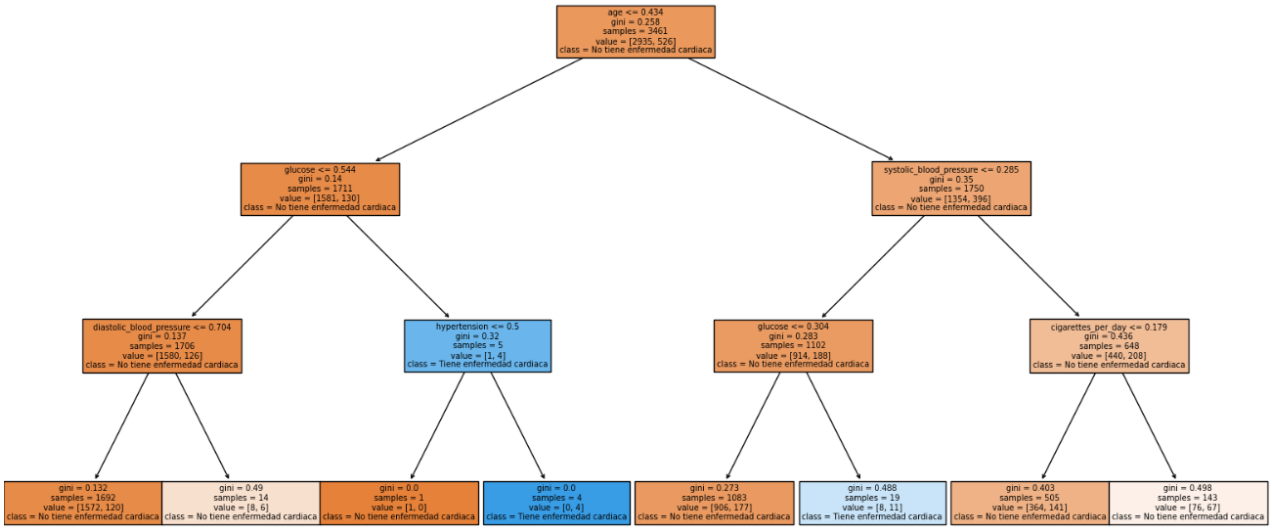
```
[ ]  # Limitar la profundidad del árbol para facilitar la visualización
     estimator.set_params(max_depth=3)  # Ajusta la profundidad según sea necesario

     fig = plt.figure(figsize=(80,40))  # Tamaño grande para alta resolución
     _ = tree.plot_tree(estimator,
                        feature_names=X.columns,
                        class_names=['No tiene enfermedad cardiaca', 'Tiene enfermedad cardiaca'],
                        node_ids=True,
                        filled=True)
     fig.savefig("decision_tree_high_res.png", dpi=300)  # Guardar con alta resolución
     plt.show()
```



```python
from sklearn import tree
import matplotlib.pyplot as plt

# Función para graficar un subárbol
def plot_subtree(estimator, max_depth, feature_names, class_names, filename):
    sub_estimator = tree.DecisionTreeClassifier(max_depth=max_depth)
    sub_estimator.fit(X, y)
    fig = plt.figure(figsize=(20,10))
    _ = tree.plot_tree(sub_estimator,
                       feature_names=feature_names,
                       class_names=class_names,
                       filled=True)
    fig.savefig(filename)
    plt.show()

# Graficar árboles de diferentes profundidades
for depth in range(1, 11):  # Ajusta el rango según sea necesario
    plot_subtree(estimator, depth, X.columns, ['No tiene enfermedad cardiaca', 'Tiene enfermedad cardiaca'], f"decision_tree_depth_{depth}.png")
```







Después de hacer todo el procedimiento necesario para este modelo, hemos considerado lo más importante para comenzar con la comparación, este cuadro de métricas con matriz de confusión.

Random Forest con todas las variables		
	Prediccion NO	Prediccion SI
Real NO	591	5
Real SI	92	5

Accuracy	0.860
F1-score	0.093
Precision	0.500
Recall	0.052
AUC	0.719

Ahora, vamos con el segundo modelo llamado “Random Forest con Feature Selection – Técnica de Random Forest”.

Antes de empezar con el modelo, vamos a hacer el procedimiento de “Feature Selection – Técnica de Random Forest”. Vamos a importar las librerías necesarias, montar el drive y leer el pickle generado de transformación de datos.

FEATURE SELECTION - TÉCNICA DE RANDOM FOREST

```
[ ] import pandas as pd
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.feature_selection import SelectFromModel
```

```
[ ] from google.colab import drive
    drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[ ] # Ruta de la fuente de datos
    ruta = '/content/drive/My Drive/BPA/TP/df_transformationdata_transformacionheart.pickle'

    df_tabla_heart = pd.read_pickle(ruta)
    df_tabla_heart
```

	heart_disease_Yes	age	blood_pressure_medications	body_mass_index	cigarettes_per_day	diabetes	diastolic_blood_pressure	glucose	heartRate	hypertension	is_current_smoker	systolic_blood_pressure	total_cholesterol
0	0	0.184	0.000	0.277	0.000	0.000	0.233	0.105	0.364	0.000	0.000	0.106	0.158
1	0	0.421	0.000	0.238	0.286	0.000	0.339	0.085	0.313	0.000	1.000	0.208	0.262
2	1	0.763	0.000	0.316	0.429	0.000	0.497	0.178	0.212	1.000	1.000	0.314	0.220
3	0	0.368	0.000	0.183	0.329	0.000	0.381	0.127	0.414	0.000	1.000	0.220	0.345
4	0	0.289	0.000	0.358	0.000	0.000	0.656	0.167	0.333	1.000	0.000	0.456	0.227
...	...	...	...	...	...	...	...	...	...	...	...	...	...
3456	0	0.684	0.000	0.228	0.000	0.000	0.349	0.116	0.364	1.000	0.000	0.272	0.141
3457	1	0.947	0.000	0.184	0.000	0.000	0.519	0.110	0.162	1.000	0.000	0.400	0.119
3458	1	0.474	0.000	0.253	0.014	0.000	0.466	0.130	0.222	1.000	1.000	0.452	0.403
3459	0	0.500	0.000	0.101	0.614	0.000	0.339	0.079	0.212	0.000	1.000	0.203	0.183
3460	0	0.526	0.000	0.144	0.000	0.000	0.370	0.189	0.364	0.000	0.000	0.236	0.312

3461 rows × 13 columns

Hacemos lo mismo como en el modelo anterior, separamos las variables independientes con la variable dependiente que es heart_disease, separamos variables de entrenamiento, testeo y definimos “RandomForest Classifier”.

```
[ ] df_tabla_heart.shape
```

```
(3461, 13)
```

```
[ ] #Separamos las predictoras vs la target

    X = df_tabla_heart.drop(['heart_disease_Yes'], axis = 1)
    y = df_tabla_heart['heart_disease_Yes']
```

```
[ ] #Importamos el train_test_split
    from sklearn.model_selection import train_test_split
```

```
[ ] #Separamos Train y test
    X_train,X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
[ ] sel = SelectFromModel(RandomForestClassifier(n_estimators = 100)) #, threshold=0.03)
    sel.fit(X_train, y_train)
```

```
SelectFromModel
estimator: RandomForestClassifier
RandomForestClassifier
```

Y obtenemos las variables que nos interesa, las más resaltantes, las demás que no son mencionadas se eliminan, todo eso presentado en la segunda imagen.

```
[ ] ## Obtener la máscara de booleanos que indica qué características han sido seleccionadas
    sel.get_support()

array([ True, False,  True, False, False,  True,  True,  True, False,
       False,  True,  True])

[ ] #columnas seleccionadas por el modelo
    selected_feat= X_train.columns[(sel.get_support())]
    selected_feat

Index(['age', 'body_mass_index', 'diastolic_blood_pressure', 'glucose',
       'heartRate', 'systolic_blood_pressure', 'total_cholesterol'],
      dtype='object')

[ ] print(len(selected_feat))

7

[ ] # Obtener la importancia de las características del estimador
    sel.estimator_.feature_importances_

array([0.1341079 , 0.00761535, 0.13870092, 0.05384373, 0.00401657,
       0.12408011, 0.12746353, 0.10291791, 0.01887345, 0.0119778 ,
       0.14450614, 0.13189658])

[ ] # Calculamos el promedio de los scores
    # proporciona una visión general del nivel promedio de importancia de las características en el conjunto de datos según el clasificador subyacente
    sel.estimator_.feature_importances_.mean()

0.08333333333333336
```

En esta imagen, se ve las variables que queremos y las más resaltantes, todo que no está aquí, lo eliminamos con un drop.

```
#columnas seleccionadas por el modelo
selected_feat= X_train.columns[(sel.get_support())]
selected_feat

Index(['age', 'body_mass_index', 'diastolic_blood_pressure', 'glucose',
       'heartRate', 'systolic_blood_pressure', 'total_cholesterol'],
      dtype='object')
```

Una vez que ya tenemos las variables que queremos, ahora sí, podemos continuar con el siguiente modelo llamado “Random Forest con Feature Selection – Técnica de Random Forest”.

Importamos las librerías necesarias, cargamos el pickle de la transformación de datos.

MODELIZACIÓN RANDOM FOREST CON FEATURE SELECTION - TÉCNICA DE RANDOM FOREST

```
[ ] import pandas as pd
    import numpy as np
    import seaborn as sns
    import matplotlib.pyplot as plt
    %matplotlib inline

    #Automcompletar rápido
    %config IPCompleter.greedy=True
    #Desactivar la notación científica
    pd.options.display.float_format = '{:.3f}'.format
```

```
[ ] df_heart_modelizacion = pd.read_pickle('/content/drive/My Drive/BPA/TP/df_transformationdata_transformacionheart.pickle')
    df_heart_modelizacion
```

	heart_disease_Yes	age	blood_pressure_medications	body_mass_index	cigarettes_per_day	diabetes	diastolic_blood_pressure	glucose	heartRate	hypertension	is_current_smoker	systolic_blood_pressure	total_cholesterol
0	0	0.184	0.000	0.277	0.000	0.000	0.233	0.105	0.364	0.000	0.000	0.106	0.158
1	0	0.421	0.000	0.238	0.286	0.000	0.339	0.085	0.313	0.000	1.000	0.208	0.262
2	1	0.763	0.000	0.316	0.429	0.000	0.497	0.178	0.212	1.000	1.000	0.314	0.220
3	0	0.368	0.000	0.183	0.329	0.000	0.381	0.127	0.414	0.000	1.000	0.220	0.345
4	0	0.289	0.000	0.358	0.000	0.000	0.656	0.167	0.333	1.000	0.000	0.456	0.227
...	...	...	...	...	...	...	...	...	...	...	...	...	...
3456	0	0.684	0.000	0.228	0.000	0.000	0.349	0.116	0.364	1.000	0.000	0.272	0.141
3457	1	0.947	0.000	0.184	0.000	0.000	0.519	0.110	0.162	1.000	0.000	0.400	0.119
3458	1	0.474	0.000	0.253	0.014	0.000	0.466	0.130	0.222	1.000	1.000	0.452	0.403
3459	0	0.500	0.000	0.101	0.614	0.000	0.339	0.079	0.212	0.000	1.000	0.203	0.183
3460	0	0.526	0.000	0.144	0.000	0.000	0.370	0.189	0.364	0.000	0.000	0.236	0.312

3461 rows × 13 columns

Ahora, borramos las variables que no nos va a servir o las que no salieron en el Feature Selection que hicimos anteriormente, después separamos variables independientes con la variable dependiente heart_disease.

```
[ ] df_heart_modelizacion_nuevo.shape
```

```
(3461, 8)
```

```
[ ] X = df_heart_modelizacion_nuevo.drop(['heart_disease_Yes'], axis = 1)
    y = df_heart_modelizacion_nuevo['heart_disease_Yes']
```

```
[ ] X
```

	age	body_mass_index	diastolic_blood_pressure	glucose	heartRate	systolic_blood_pressure	total_cholesterol
0	0.184	0.277	0.233	0.105	0.364	0.106	0.158
1	0.421	0.238	0.339	0.085	0.313	0.208	0.262
2	0.763	0.316	0.497	0.178	0.212	0.314	0.220
3	0.368	0.183	0.381	0.127	0.414	0.220	0.345
4	0.289	0.358	0.656	0.167	0.333	0.456	0.227
...	...	...	...	...	...	...	...
3456	0.684	0.228	0.349	0.116	0.364	0.272	0.141
3457	0.947	0.184	0.519	0.110	0.162	0.400	0.119
3458	0.474	0.253	0.466	0.130	0.222	0.452	0.403
3459	0.500	0.101	0.339	0.079	0.212	0.203	0.183
3460	0.526	0.144	0.370	0.189	0.364	0.236	0.312

3461 rows × 7 columns

```
y
```

```
0       0
1       0
2       1
3       0
4       0
       ..
3456    0
3457    1
3458    1
3459    0
3460    0
Name: heart_disease_Yes, Length: 3461, dtype: int64
```

Definimos las variables de entrenamiento, de testeo, importamos el modelo, lo instanciamos, lo entrenamos.

```
[ ] from sklearn.model_selection import train_test_split

[ ] X_train,X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

[ ] print(X_train.shape)
    print(X_test.shape)
    print(y_train.shape)
    print(y_test.shape)

    (2768, 7)
    (693, 7)
    (2768,)
    (693,)

[ ] #Importamos el modelo
    from sklearn.ensemble import RandomForestClassifier

[ ] #Instanciamos el modelo
    modelo_RandomForest = RandomForestClassifier()

[ ] #Entrenamos el modelo
    modelo_RandomForest.fit(X_train,y_train)

    ▾ RandomForestClassifier
    RandomForestClassifier()
```

Ahora, vamos a predecir usando los datos de testeo, importamos las métricas, la matriz de confusión, y vamos a ver los resultados.

```
[ ] #Predecimos usando los datos de test
    y_pred_RandomForest       = modelo_RandomForest.predict(X_test)
    y_pred_RandomForest_proba = modelo_RandomForest.predict_proba(X_test)
    #y_pred_RandomForest_proba[:,1][:10]
```

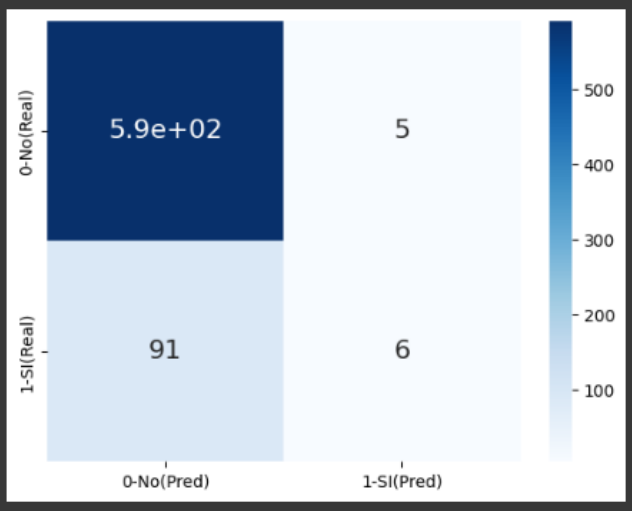
```
[ ] #Importamos las métricas
    from sklearn.metrics import roc_auc_score,confusion_matrix,f1_score,classification_report,\
                                accuracy_score,precision_score,recall_score
```

```
[ ] confusion_matrix_RandomForest = confusion_matrix(y_test,y_pred_RandomForest)
```

```
[ ] #Mostrar el confusion matrix como dataframe
    pd.DataFrame(confusion_matrix_RandomForest,columns = ['Prediccion NO','Prediccion SI'], index = ['Real NO','Real SI'])
```

	Prediccion NO	Prediccion SI
Real NO	591	5
Real SI	91	6

```
[ ] #Mostrar el confusion matrix como grafico
    sns.heatmap(confusion_matrix_RandomForest, annot=True,xticklabels=['0-No(Pred)','1-SI(Pred)'],
                yticklabels=['0-No(Real)','1-SI(Real)'], cmap = 'Blues',annot_kws={"size": 16});
```




```
[ ] acc_RandomForest  = accuracy_score(y_test,y_pred_RandomForest)
    f1_RandomForest   = f1_score(y_test,y_pred_RandomForest)
    prec_RandomForest = precision_score(y_test, y_pred_RandomForest)
    rec_RandomForest  = recall_score(y_test, y_pred_RandomForest)
    auc_RandomForest  = roc_auc_score(y_test,y_pred_RandomForest_proba[:,1])
```

```
[ ] results = pd.DataFrame([['Random Forest', acc_RandomForest,f1_RandomForest,prec_RandomForest,rec_RandomForest,auc_RandomForest]],
                           columns = ['Model','Accuracy','F1','Precision','Recall','AUC'])
```

Ahora, vamos a ver los resultados de las métricas, se puede observar que se obtuvo un puntaje mayor al anterior modelo que ejecutamos.

```
[ ]  results
```

	Model	Accuracy	F1	Precision	Recall	AUC
0	Random Forest	0.861	0.111	0.545	0.062	0.707

Random Forest con Feature Selection - Técnica de Random Forest

	Prediccion NO	Prediccion SI
Real NO	591	5
Real SI	91	6

Accuracy	0.861
F1-score	0.111
Precision	0.545
Recall	0.062
AUC	0.707

Observamos los árboles de decisión generados, número de variables usadas (son siete, las más resaltantes), definimos el árbol en índice 5.

```
#Número de arboles generados
modelo_RandomForest.n_estimators
```

```
100
```

```
[ ]  #Numero de variables usadas
modelo_RandomForest.n_features_in_
```

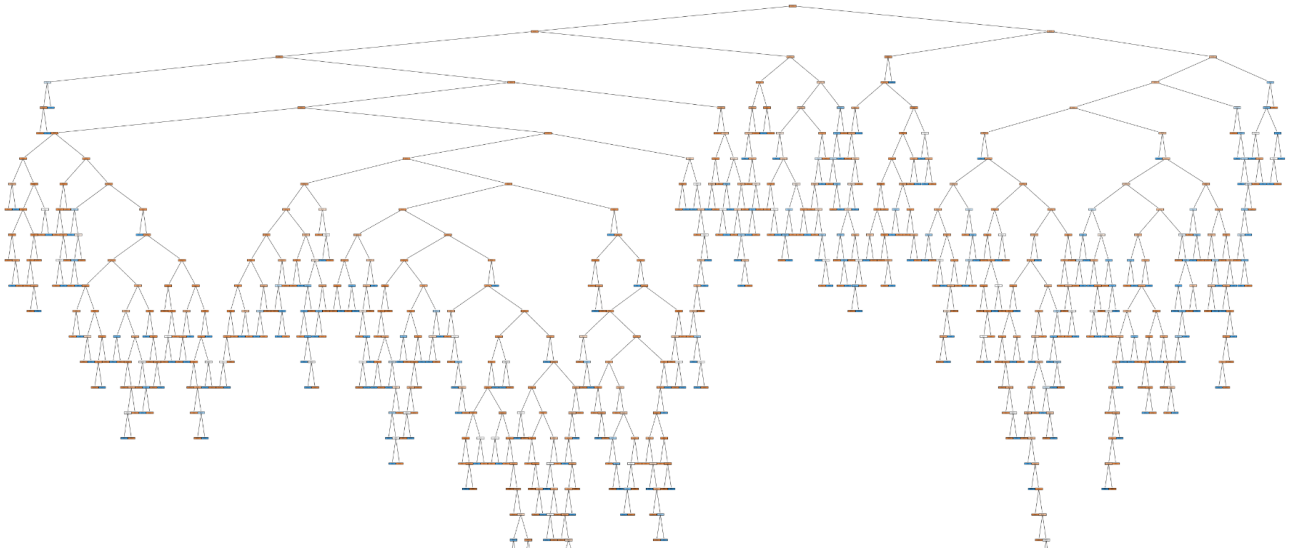
```
7
```

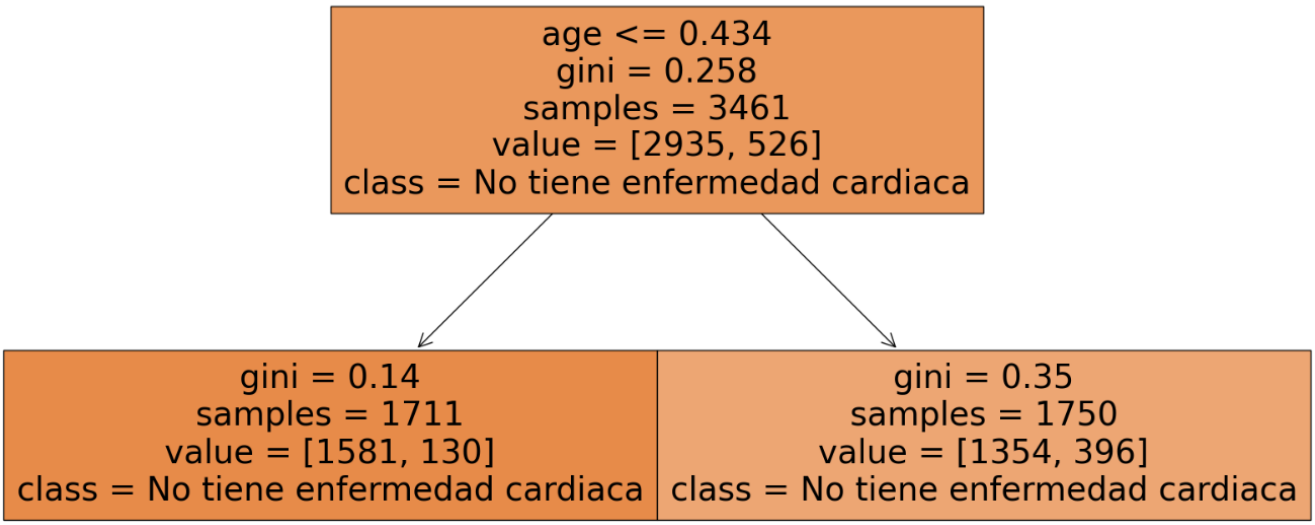
```
[ ]  #Arbol en indice 5
estimator = modelo_RandomForest.estimators_[5]
```

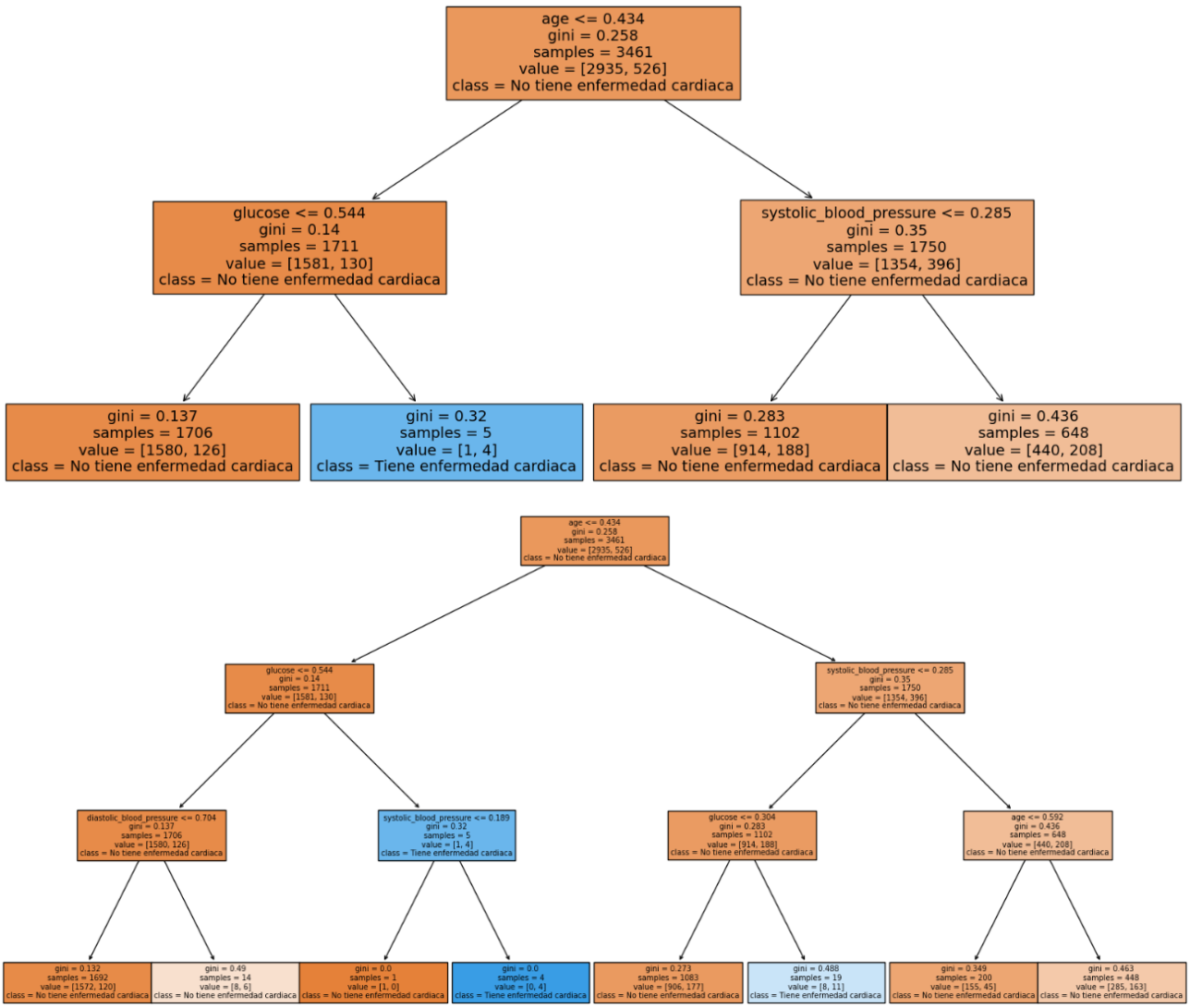
Ahora, vamos a ver todos los árboles generados.

```
[ ] from sklearn import tree
```

```
# Limitar la profundidad del árbol para facilitar la visualización
estimator.set_params(max_depth=3)  # Ajusta la profundidad según sea necesario

fig = plt.figure(figsize=(80,40))  # Tamaño grande para alta resolución
_ = tree.plot_tree(estimator,
                   feature_names=X.columns,
                   class_names=['No tiene enfermedad cardiaca', 'Tiene enfermedad cardiaca'],
                   node_ids=True,
                   filled=True)
fig.savefig("decision_tree_high_res.png", dpi=300)  # Guardar con alta resolución
plt.show()
```







Comparando los modelos, con los puntajes obtenidos y las métricas más importantes que son el F1-score y AUC, se concluye en la comparación que el modelo “Random Forest con Feature Selection – Técnica de Random Forest” es el ganador y el que utilizaremos para los siguientes procedimientos que es la optimización y los nuevos resultados que saldrá después de realizar eso.

Random Forest con todas las variables		
	Prediccion NO	Prediccion SI
Real NO	591	5
Real SI	92	5
Accuracy		0.860
F1-score		0.093
Precision		0.500
Recall		0.052
AUC		0.719

MODELO DESCARTADO

Random Forest con Feature Selection - Técnica de Random Forest		
	Prediccion NO	Prediccion SI
Real NO	591	5
Real SI	91	6
Accuracy		0.861
F1-score		0.111
Precision		0.545
Recall		0.062
AUC		0.707

MEJOR MODELO

Por lo que guardaremos la fase del modelo de “Random Forest con Feature Selection – Técnica de Random Forest”.

```
[ ]  df_heart_modelizacion_nuevo.to_pickle('/content/drive/My Drive/BPA/TP/df_heart_modelizacion_nuevo_variables_actualizadas.pickle')
```

4.2. OPTIMIZACIÓN

Ahora, vamos a optimizar el modelo ganador que es el “Random Forest con Feature Selection – Técnica de Random Forest”, de esa forma obtendremos mejores resultados de las métricas y de la matriz de confusión, importamos librerías, montamos drive y cargamos el pickle del modelo de “Random Forest con Feature Selection – Técnica de Random Forest”.

OPTIMIZACIÓN

```
[ ]  import pandas as pd
     import numpy as np
     import seaborn as sns
     import matplotlib.pyplot as plt
     %matplotlib inline

     #Automcompletar rápido
     %config IPCompleter.greedy=True
     #Desactivar la notación científica
     pd.options.display.float_format = '{:.3f}'.format
```

```
[ ]  from google.colab import drive
     drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[ ]  df_heart_optimizacion = pd.read_pickle('/content/drive/My Drive/BPA/TP/df_heart_modelizacion_nuevo_variables_actualizadas.pickle')
     df_heart_optimizacion
```

	heart_disease_Yes	age	body_mass_index	diastolic_blood_pressure	glucose	heartRate	systolic_blood_pressure	total_cholesterol
0	0	0.184	0.277	0.233	0.105	0.364	0.106	0.158
1	0	0.421	0.238	0.339	0.085	0.313	0.208	0.262
2	1	0.763	0.316	0.497	0.178	0.212	0.314	0.220
3	0	0.368	0.183	0.381	0.127	0.414	0.220	0.345
4	0	0.289	0.358	0.656	0.167	0.333	0.456	0.227
...	...	...	...	...	...	...	...	...
3456	0	0.684	0.228	0.349	0.116	0.364	0.272	0.141
3457	1	0.947	0.184	0.519	0.110	0.162	0.400	0.119
3458	1	0.474	0.253	0.466	0.130	0.222	0.452	0.403
3459	0	0.500	0.101	0.339	0.079	0.212	0.203	0.183
3460	0	0.526	0.144	0.370	0.189	0.364	0.236	0.312

Separamos variables independientes con la variable dependiente heart disease, importamos “train_test_split”, definimos las variables de entrenamiento y testeo, importamos “RandomForestClassifier” y le damos a una variable algoritmo_rf.

```
X = df_heart_optimizacion.drop(['heart_disease_Yes'], axis = 1)
y = df_heart_optimizacion['heart_disease_Yes']

from sklearn.model_selection import train_test_split

X_train,X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

from sklearn.ensemble import RandomForestClassifier

algoritmo_rf = RandomForestClassifier()
```

Ahora, vamos a usar la técnica de Grid Search para poder buscar sistemáticamente a través de un conjunto predefinido de hiperparámetros para encontrar la combinación óptima que maximiza el rendimiento del modelo ganador que hemos dicho.

```
GRID SEARCH

grid_parameters = { 'criterion'        : ['giny','entropy'], # Gini y entropy sirven medir la impureza de un nodo aleatoriamente
                    'max_depth'        : [10, 14, 20], #profundidad del arbol
                    'max_features'     : [10,20], #cantidad máxima de características que se consideran para dividir un nodo en un árbol de decisión
                    'min_samples_leaf' : [2, 4],  #número mínimo de muestras requeridas  para formar una hoja
                    'min_samples_split': [5, 10], #número mínimo de muestras requeridas para realizar una división en un nodo interno
                    'n_estimators'     : [100, 500], #Crear un clasificador de bosque aleatorio con N# estimadores
                    'n_jobs'           : [-1]} #(n_jobs) -1 para usar todos los núcleos disponibles

from sklearn.model_selection import GridSearchCV

grid_search_rf = GridSearchCV(
                                estimator           = algoritmo_rf,
                                param_grid          = grid_parameters,
                                scoring             = 'roc_auc',
                                n_jobs              = -1,
                                cv                  = 3,
                                verbose             = 0
                               )

mejor_modelo_rf = grid_search_rf.fit(X_train,y_train)
```

Una vez hecho el entrenamiento, nos va a dar los mejores parámetros que tenemos que definir a nuestro modelo ganador.

```
print('Configuración de los mejores parámetros:')
mejor_modelo_rf.best_params_

Configuración de los mejores parámetros:
{'criterion': 'entropy',
 'max_depth': 10,
 'max_features': 20,
 'min_samples_leaf': 2,
 'min_samples_split': 10,
 'n_estimators': 500,
 'n_jobs': -1}
```

Y este es el mejor resultado de la métrica AUC.

```
[ ] print(f'Resultado de la métrica {mejor_modelo_rf.scoring} de la mejor configuración de parámetros:')

    mejor_modelo_rf.best_score_
```

```
Resultado de la métrica roc_auc de la mejor configuración de parámetros:
0.6651980761193755
```

CAPÍTULO 5: RESULTADOS DE MODELIZACIÓN

5.1. PRESENTACIÓN DE RESULTADOS FINAL

Ahora, vamos a entrenar nuestro modelo ganador con los mejores parámetros que nos ha dado la técnica Grid Search.

```
#Modelo Final: Entrenar con los mejores parámetros a todo TRAIN
algoritmo_rf = RandomForestClassifier(criterion='entropy', max_depth=10, max_features=20,
                       min_samples_leaf=2, min_samples_split=10,
                       n_estimators=500, n_jobs=-1)
algoritmo_rf.fit(X_train, y_train)
```

```
RandomForestClassifier(criterion='entropy', max_depth=10, max_features=20,
                       min_samples_leaf=2, min_samples_split=10,
                       n_estimators=500, n_jobs=-1)
```

```
[ ] y_pred_rf_hy       = algoritmo_rf.predict(X_test)
    y_pred_rf_proba_hy = algoritmo_rf.predict_proba(X_test)
    y_pred_rf_proba_hy[:,1][:10]
```

```
array([0.14432537, 0.15713656, 0.2240986 , 0.06649744, 0.13366875,
       0.06578384, 0.47359307, 0.24350122, 0.2232925 , 0.02463556])
```

Veremos los nuevos resultados de las métricas con la matriz de confusión.

```
[ ] from sklearn.metrics import roc_auc_score,confusion_matrix,f1_score,classification_report,\
                                accuracy_score,precision_score,recall_score
```

```
[ ] acc_rf_hy  = accuracy_score(y_test,y_pred_rf_hy)
    f1_rf_hy   = f1_score(y_test,y_pred_rf_hy)
    prec_rf_hy = precision_score(y_test, y_pred_rf_hy)
    rec_rf_hy  = recall_score(y_test, y_pred_rf_hy)
    auc_rf_hy  = roc_auc_score(y_test,y_pred_rf_proba_hy[:,1])
```

Aquí están los resultados finales optimizados.

```
[ ] resultsrf_hy = pd.DataFrame([['Random Forest Hyperparameters', acc_rf_hy,f1_rf_hy,prec_rf_hy,rec_rf_hy,auc_rf_hy]],
                                columns = ['Model','Accuracy','F1','Precision','Recall','AUC'])
    resultsrf_hy
```

	Model	Accuracy	F1	Precision	Recall	AUC
0	Random Forest Hyperparameters	0.863	0.128	0.583	0.072	0.721

RESULTADOS FINALES CON OPTIMIZACIÓN: Random Forest con Feature Selection - Técnica de Random Forest

	Prediccion NO	Predicción SI	
Real NO	591	5	
Real SI	90	7	
Accuracy			0.863
F1			0.128
Precision			0.583
Recall			0.072
AUC			0.721

MODELO OPTIMIZADO

En este trabajo, hemos abordado la necesidad crucial de la detección y manejo temprano de enfermedades cardiovasculares mediante el uso de técnicas avanzadas de análisis de datos. Utilizando un conjunto de datos relevantes, como ultimas optimizaciones aplicamos el algoritmo de Random Forest en tres etapas distintas: inicialmente con todas las variables, luego con una selección de características y, finalmente, optimizando los hiperparámetros a través de Grid Search. En la primera fase, el modelo con todas las variables mostró una precisión (accuracy) de 0.860, con un F1-Score de 0.093, precisión de 0.500, recall de 0.052 y un AUC de 0.719. El algoritmo de Random Forest que funciona creando múltiples árboles de decisión durante el entrenamiento y emitiendo la predicción promedio de estos árboles. Esta técnica es robusta frente al sobreajuste y maneja bien tanto las variables categóricas como las numéricas. Sin embargo, incluir todas las variables puede introducir ruido y características irrelevantes, lo que puede afectar el rendimiento del modelo, como se observó en la baja sensibilidad (recall). Al aplicar la técnica de selección de características en la segunda fase, observamos una ligera mejora en las métricas de rendimiento. La selección de características implica identificar y utilizar solo las variables más relevantes para el modelo, eliminando aquellas que no aportan valor o que pueden introducir ruido. Este proceso puede mejorar el rendimiento del modelo al centrarse en las variables que tienen un impacto significativo en la predicción, lo que se reflejó en la mejora de las métricas observadas. Finalmente, la optimización de hiperparámetros

mediante Grid Search llevó el modelo a su mejor desempeño. La precisión alcanzó 0.863, el F1-Score 0.128, la precisión 0.583 y el recall 0.072, con un AUC de 0.721.Optimizar estos parámetros permite encontrar la combinación que maximiza la capacidad predictiva del modelo. Este ajuste fino es crucial para maximizar el rendimiento del modelo, logrando un balance más óptimo entre precisión y recall.

En conclusión se ha demostrado la eficacia en la mejora de la detección temprana de enfermedades cardiovasculares. Cada etapa de mejora ha aportado incrementos significativos en las métricas subrayando la importancia de técnicas avanzadas en el desarrollo de modelos predictivos robustos y precisos. Para futuras investigaciones y aplicaciones prácticas, recomendamos implementar estos modelos en entornos clínicos para validar su utilidad en la detección temprana de enfermedades cardiovasculares, así como la validación de los conjuntos de datos externos para confirmar la generalización y robustez del modelo en diversas poblaciones, garantizando así su eficacia y fiabilidad en diferentes contextos clínicos. La combinación de estas estrategias permitirá desarrollar herramientas predictivas que realmente puedan transformar la atención médica y mejorar los resultados de salud de los pacientes.

VISUALIZACIÓN DEL DASHBOARD HEART DISEASE POWER BI

El dashboard se basa en la búsqueda generalmente de las categorías del género, sea femenino o masculino, en ello veremos lo siguiente:

- **El recuento de cigarros por día:** Nos habla de cuantos cigarros se consume por educación, osea el total de graduados tanto femenino como masculino es de 572, si presionan la categoría de femenino saldrá que la mujeres graduadas que fuman son 378 y hombres de hombres son 194.

- **Top 5: Enfermedades por Edad:** No demuestra el top 5 de las edades con más porcentaje a tener una enfermedad (varia en tanto a mujer o varón).

- **Suma de diabetes:** Es la suma de la cantidad de personas que si tiene diabetes.

- **Fumador actual:** Lo mismo que el anterior, la suma de cuantas personas si son fumadores actuales.

- **Suma de Hipertensión por Heart disease:** Aquí nos explican cuántos si sufren de hipertensión y cuantos no (varia si es mujer o varón).

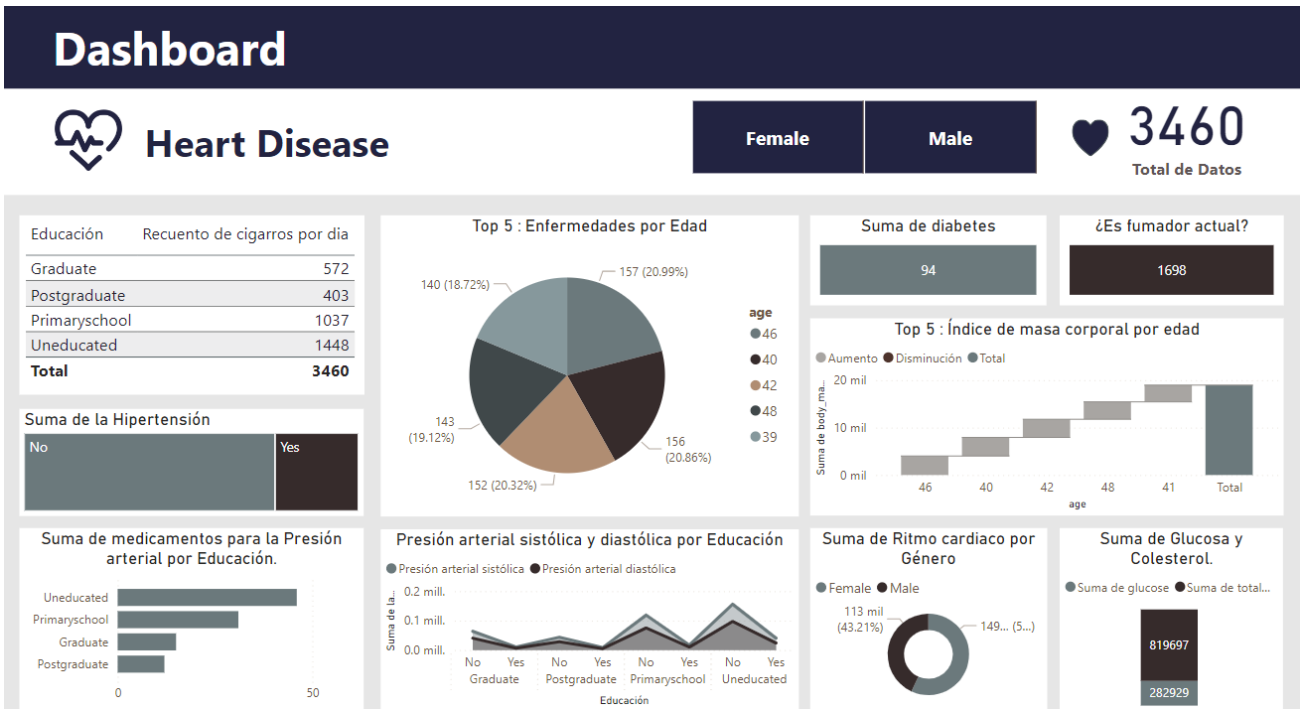
- **Top 5:** Índice de masa corporal por edad: Nos explica la suma de la masa corporal entre un top 5 de edades con más masa corporal.

- **Suma de medicamentos:** Nos explica la cantidad de personas, ya sea mujer o varón, en base a su educación, que toman medicamentos para su presión arterial.

- **Presión sistólica y diastólica:** Nos da un comparativo entre la suma de las presiones, donde las categorías son por la educación y el heart disease.

- **Suma de ritmo cardiaco:** suma de todos los ritmos, mostrados en porcentajes y separados en mujer y varón para saber quién tiene más probabilidad de heart disease.

- **Suma de glucosa y colesterol:** suma de los totales de las cantidades de glucosa y colesterol, separados por genero al dar click en las categorías.



APORTES

- **André Nicolas Vásquez Castro:** Encargado de la recolección y preprocesamiento de datos, así como de la implementación y optimización del modelo Random Forest.

- **Valeria Milagros Caqui Pizarro:** Responsable de la evaluación y análisis de escenarios, así como de la implementación de modelos predictivos.

- **Luis Ángel Bendezú Jiménez:** Contribuyó en el análisis exploratorio de datos (EDA) y en la transformación de variables categóricas y numéricas.

- **Neil Eduardo Trujillo Neyra:** Participó en la definición del problema, los objetivos de Machine Learning y en la generación de insights a partir de los datos.

- **Gianella Lucía Silvina Gonzales:** Colaboró en la presentación y visualización de los resultados finales, y en la elaboración del informe y conclusiones del proyecto.

Logramos colaborar de manera efectiva para realizar un análisis exhaustivo de los datos de la Clínica Mayo, aplicando técnicas avanzadas de Machine Learning para la predicción de enfermedades cardiovasculares. Juntos, desarrollamos un modelo robusto y preciso, optimizado mediante técnicas de selección de características y ajuste de hiperparámetros. Además, nos aseguramos de validar y comunicar claramente los resultados a través de visualizaciones y presentaciones detalladas, garantizando que los hallazgos fueran comprensibles y útiles para la toma de decisiones clínicas en la prevención y manejo de enfermedades cardiovasculares.

CONCLUSIONES

- Es crucial tener una comprensión clara del problema y de los objetivos del negocio al inicio del proyecto. Esta claridad guiará todas las fases del análisis de datos, asegurando que cada paso tomado contribuya efectivamente hacia la solución del problema identificado.

- La preparación y limpieza de datos son fundamentales para garantizar la calidad del análisis. Esto incluye la corrección de errores, la eliminación de datos duplicados o irrelevantes, y la transformación adecuada de los datos para su análisis. Estos pasos incrementan la precisión de los insights generados y la fiabilidad de los modelos predictivos.

- Las visualizaciones son herramientas poderosas para comunicar hallazgos y tendencias en los datos. Utilizar gráficos adecuados puede facilitar la comprensión de complejidades en los datos y resaltar diferencias y similitudes críticas que pueden ser fundamentales para la toma de decisiones.

- Organizar y gestionar adecuadamente los archivos y las rutas de acceso en plataformas como Google Drive no solo mejora la eficiencia sino también la seguridad del acceso a los datos. Esto es vital en entornos colaborativos donde múltiples usuarios necesitan acceder y manipular los datos de manera concurrente.

- Verificar la veracidad y relevancia de los datasets es esencial. Trabajar con datos que son precisos y pertinentes al problema asegura que las conclusiones y predicciones del análisis sean válidas y aplicables en escenarios reales.

RECOMENDACIONES

- Recomendamos realizar una comparación detallada entre el modelo Random Forest y otros modelos predictivos como Gradient Boosting y K-Nearest Neighbors (KNN). Esta comparación debe incluir métricas como precisión, recall, F1-score y área bajo la curva ROC (AUC). Los resultados deben mostrarse en gráficos para facilitar la visualización de las diferencias y la selección del mejor modelo.

- Se sugiere implementar técnicas de optimización de hiperparámetros, como Grid Search o Random Search, para cada modelo probado. Esto mejorará la precisión y robustez de los modelos predictivos, asegurando que se obtenga el mejor rendimiento posible.

- Es recomendable utilizar técnicas de visualización avanzada para presentar los resultados del análisis de datos y los modelos predictivos. Gráficos de dispersión, diagramas de correlación y mapas de calor pueden mostrar las relaciones entre variables y patrones identificados. Estas visualizaciones facilitarán la interpretación de los resultados y ayudarán en la toma de decisiones informadas sobre la prevención y manejo de enfermedades cardiovasculares.

GLOSARIO

- **Conjunto de Datos (Dataset):** Colección de datos organizada, se utiliza para el análisis y modelado en proyectos de machine learning.

- **Aprendizaje Automático (Machine Learning):** Subcampo de la inteligencia artificial que se centra en el desarrollo de algoritmos y modelos que permiten a las computadoras aprender y hacer predicciones o decisiones basadas en datos.

- **Índice de Masa Corporal (IMC):** Medida que relaciona el peso y la altura de una persona, utilizada para evaluar si una persona tiene un peso saludable.

- **Hipertensión:** Condición médica caracterizada por una presión arterial persistentemente alta, que es un factor de riesgo significativo para enfermedades cardiovasculares.

- **Modelo Predictivo:** Algoritmo o conjunto de reglas que se utilizan para predecir un resultado específico basado en datos de entrada. En el contexto de este proyecto, se refiere a modelos que predicen el riesgo de enfermedades cardiovasculares.

- **Random Forest:** Algoritmo de aprendizaje automático basado en la construcción de múltiples árboles de decisión, que utiliza el promedio de las predicciones de estos árboles para mejorar la precisión y evitar el sobreajuste.

- **Grid Search:** Método exhaustivo de búsqueda que evalúa todas las combinaciones posibles de un conjunto predefinido de hiperparámetros.

- **Área Bajo la Curva (AUC):** Métrica que mide la capacidad del modelo para diferenciar entre clases positivas y negativas. Un AUC cercano a 1 indica un modelo con excelente capacidad predictiva.

- **Precisión (Accuracy):** Porcentaje de predicciones correctas realizadas por el modelo.

BIBLIOGRAFÍA

- MedicalNewsToday. (15 de diciembre del 2019). *Todo lo que debes saber acerca de las enfermedades del corazón.* Recuperado de https://www.medicalnewstoday.com/articles/es/327293.

- Cigna Healthcare. (12 de abril del 2021). *Síntomas y causas de enfermedad cardíaca.* Recuperado de https://www.cigna.com/es-us/knowledge-center/heart-health.

- RadiologyInfo.org. (25 de septiembre del 2022). *Detección temprana de enfermedades cardíacas (del corazón).* Recuperado de https://www.radiologyinfo.org/es/info/screening-cardiac.

- Quironsalud. (27 de junio del 2022). *Las pruebas para detectar enfermedades cardíacas, el electrocardiograma y ecocardiograma.* Recuperado de https://www.quironsalud.com/blogs/es/corazon-salud/pruebas-detectar-enfermedades-cardiacas-electrocardiograma.

ANEXOS

- **DATASET:**

  https://www.kaggle.com/datasets/mirzahasnine/heart-disease-dataset

- **LINK DEL GOOGLE COLABORATE DE NUESTRO TRABAJO FINAL:**

  https://drive.google.com/drive/folders/1z3yPVZUWr6DPqd6k6WUS1kXuztowmX9l?usp=sharing

- **DASHBOARD HEART DISEASE POWER BI**

Heart Disease DashBoard Power BI.p