

UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS
TRABAJO FINAL
BUSINESS PREDICTIVE ANALYTICS
GRUPO 2



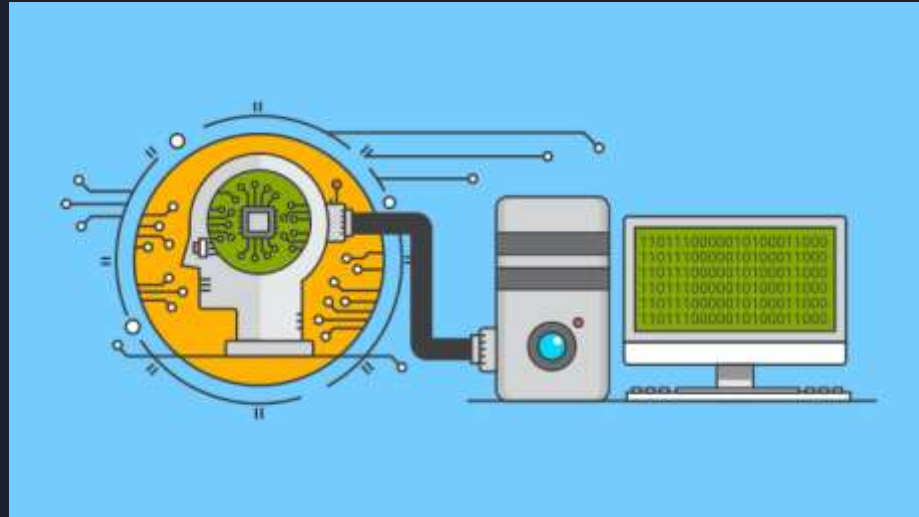
INTEGRANTES:

- ANDRÉ NICOLAS VÁSQUEZ CASTRO (U20211B559).
- VALERIA MILAGROS CAQUI PIZARRO (U20211C241).
- LUIS ÁNGEL BENDEZÚ JIMENEZ (U202022364).
- NEIL EDUARDO TRUJILLO NEYRA (U202020118).
- GIANELLA LUCÍA SILVINA GONZALES (U202019652).

SECCIÓN: SS92

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.



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.



1.2. EVALUAR Y ANALIZAR ESCENARIOS

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.



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



1.3. DEFINIR LOS OBJETIVOS DE ML

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.



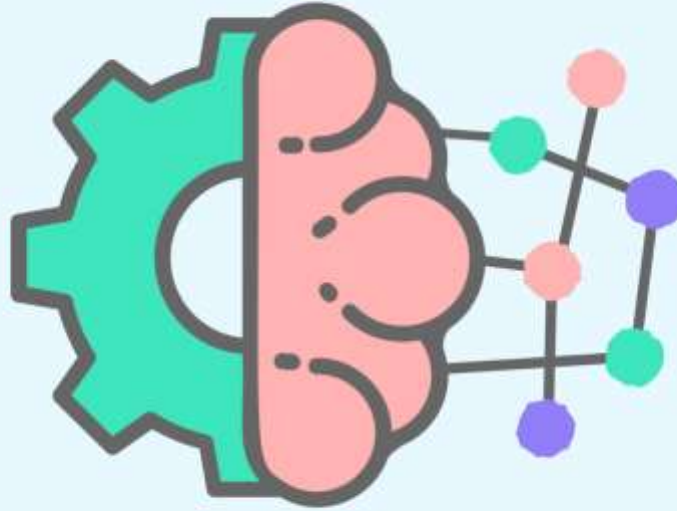
1.3. DEFINIR LOS OBJETIVOS DE ML

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.



1.3. DEFINIR LOS OBJETIVOS DE ML

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

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 Bendezu/Neil Trujillo
11	2.4	Transformación de los datos	4 horas	Viernes 03/05/2024	Jueves 27/06/2024	Luis Bendezu/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 Bendezu

1.4. PLAN DE PROYECTO

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 Bendezu 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 Bendezu Jimenez
23	10	Bibliografía	2 horas	25/06/2024	Jueves 27/06/2024	Gianella Silvina



PRE-PROCESAMIENTO DE LOS DATOS (DATA QUALITY & CLEANING)

Carga de datos

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

%matplotlib inline

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

from IPython.display import display
```

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

Mounted at /content/drive

```
[14] # Cargar el dataset desde el archivo CSV
df_heart_disease_dirty5 = pd.read_csv('/content/drive/my-drive/ana/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 dataset ordenado
df_heart_disease_dirty5.head(5)
```

	smz	smmeds	Gender	Heart_stroke	age	cigsperDay	currentSmoker	diastP	diabetes	education	glucose	heartRate	prevalentbyp	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	243
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

Corrección de cabeceras o headers

```
[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

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)
```

Verificación de inconsistencias textuales

```
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: []
```


Correcciones de 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)
```

Correcciones de 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 duplicadas

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

Eliminar 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 - categóricas

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

Verificación de valores nulos - categóricas

```
0. #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
```

```
0. [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
```

Eliminar valores nulos - categóricas

```
[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
```


Verificar valores nulos - numéricas

✓
0 s



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

✓
0 s



```
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
```

Verificar valores nulos - 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
```

Grabar fin de fase - Data Quality & Cleaning

GRABAR FIN DE FASE

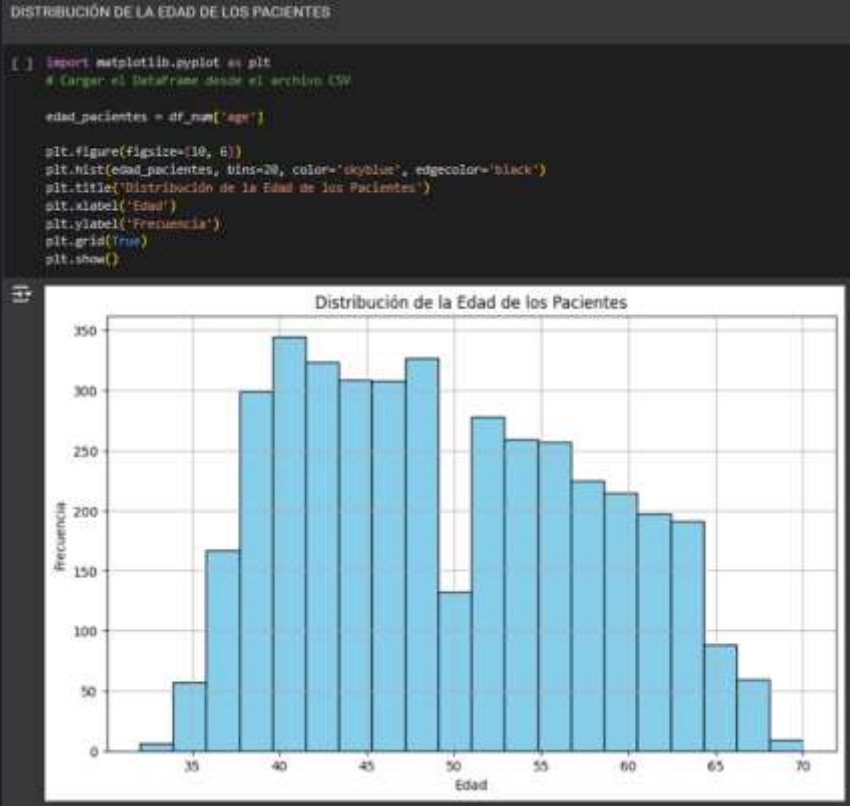
```
[ ] df_cat.to_pickle('/content/drive/My Drive/BPA/TP/df_heart_disease_cat_fin_quality_cleaning.pickle')  
    df_num.to_pickle('/content/drive/My Drive/BPA/TP/df_heart_disease_num_fin_quality_cleaning.pickle')  
    df_cat_num.to_pickle('/content/drive/My Drive/BPA/TP/df_heart_disease_cat_num_fin_quality_cleaning.pickle')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



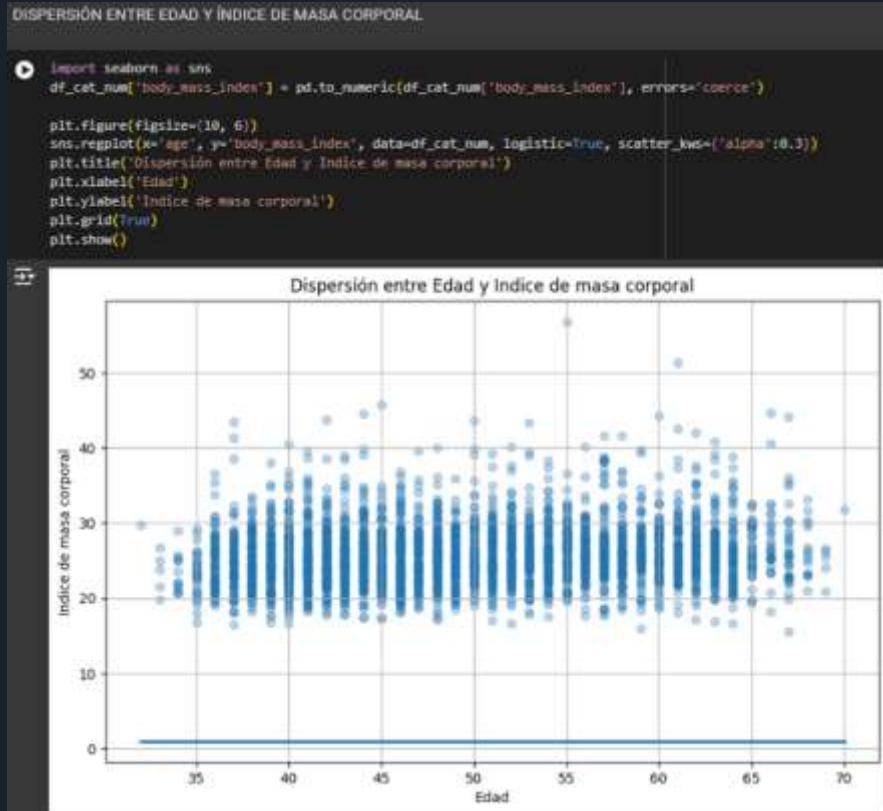
EDA

(Análisis exploratorio de datos)

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



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



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

```
import matplotlib.pyplot as plt

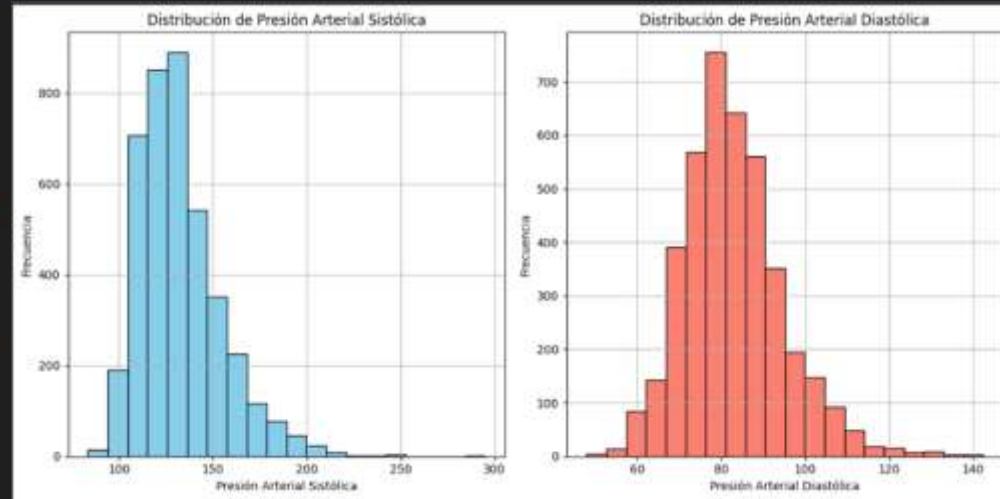
presion_sistolica = df_num['systolic_blood_pressure']
presion_diastolica = df_num['diastolic_blood_pressure']

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?

RELACIÓN ENTRE PRESIÓN ARTERIAL SISTÓLICA Y DIASTÓLICA EN HOMBRES Y MUJERES

```
import matplotlib.pyplot as plt

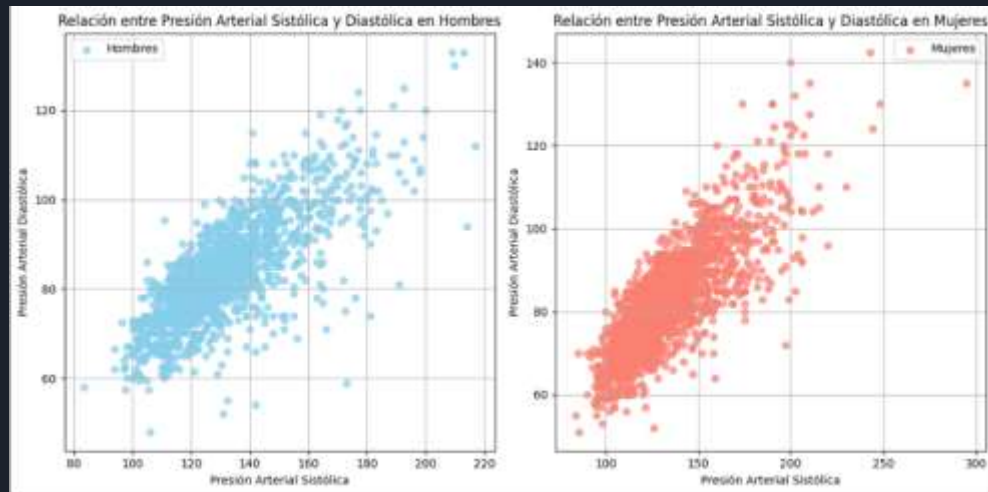
# Filtrar datos por género
df_male = df_cat_num[df_cat_num['gender'] == 'Male']
df_female = df_cat_num[df_cat_num['gender'] == 'Female']

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

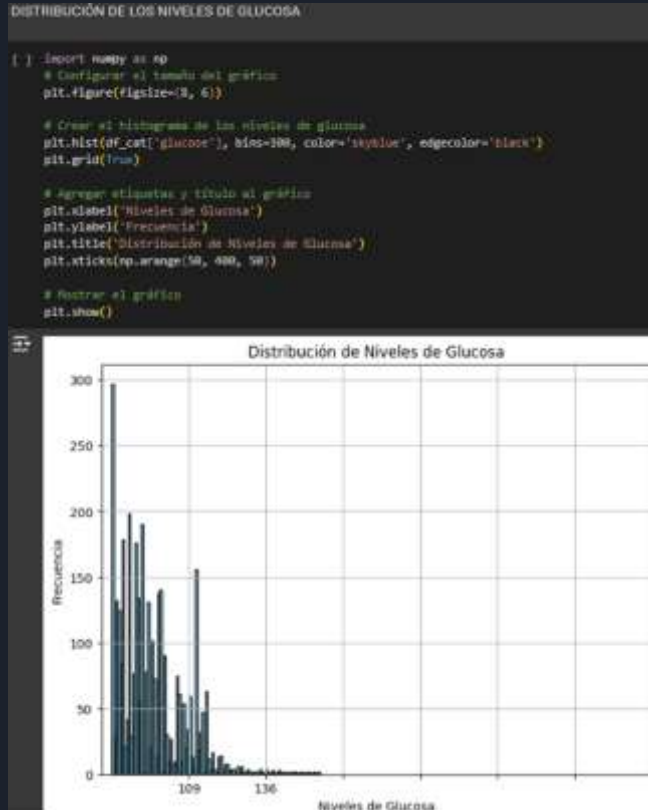
# Gráfica de dispersión para la relación entre presión arterial sistólica y diastólica en hombres
plt.subplot(1, 2, 1)
plt.scatter(df_male['systolic_blood_pressure'], df_male['diastolic_blood_pressure'], color='skyblue', label='Hombres', alpha=0.7)
plt.title('Relación entre Presión Arterial Sistólica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

# Gráfica 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?



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

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')

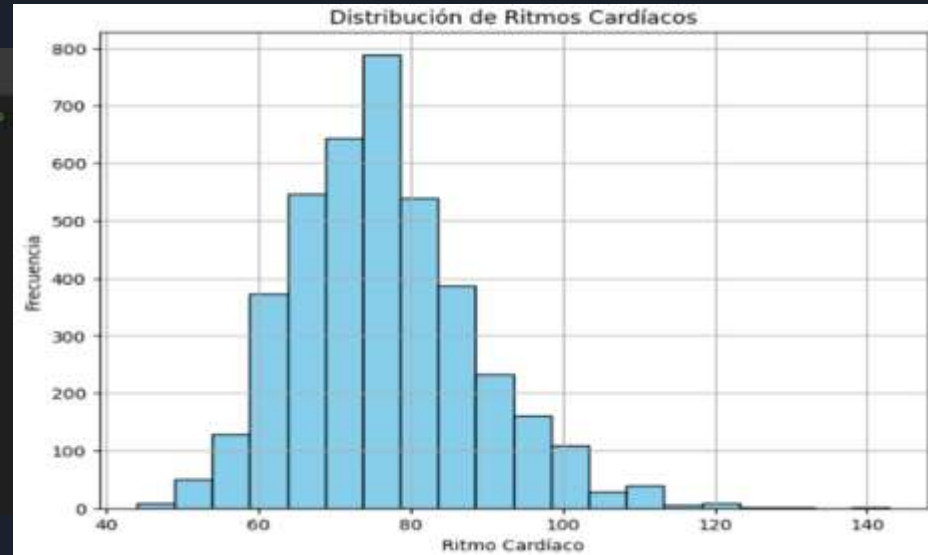
# Configurar el tamaño del gráfico
plt.figure(figsize=(8, 6))

# 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

RITMO CARDÍACO SEGÚN INTERVALOS DE CIGARRILLOS POR DÍA

```
[ ] import numpy as np

# Eliminar filas con valores faltantes en las columnas de interés
df_cleaned = df_cat.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) # Intervalo 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]

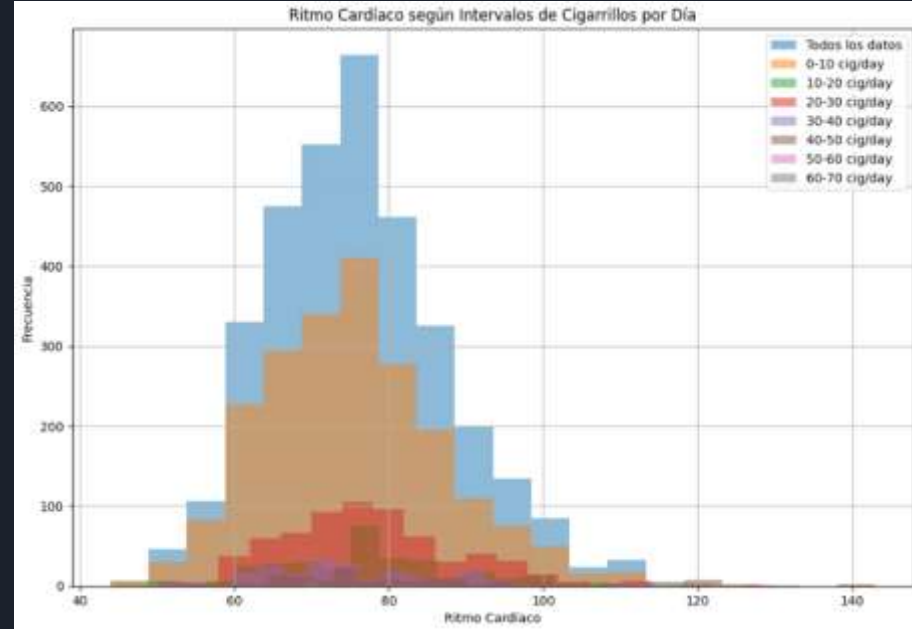
    # Filtrar datos dentro del intervalo de cigarrillos por día
    subset_data = df_cleaned[(df_cleaned['cigarettes_per_day'] >= lower_bound) & (df_cleaned['cigarettes_per_day'] < upper_bound)]

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

# Agregar cuadrícula 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





TRANSFORMACIÓN DE LOS DATOS

Transformación de los datos

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)
```


Transformación de los datos a oneHotEncoder - categóricas

```
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

Transformación de los datos a oneHotEncoder - categóricas gender

```
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 x 1 columns

Transformación de los datos a MinMaxScaler - numéricas

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 x 9 columns

Transformación de los datos a MinMaxScaler - numéricas todas las variables

```
from sklearn.preprocessing import MinMaxScaler
```

```
columns = ['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[columns])  
df_mms = mms.transform(df_heart_disease_cleaned[columns])
```

```
df_mms = pd.DataFrame(data=df_mms, columns=columns)  
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 x 12 columns

Transformación de los datos - dataframe final


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

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 x 13 columns

Transformación de los datos - guardar fin de fase



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



INSIGHTS

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

```
import pandas as pd

# Convertir la columna 'cigarettes_per_day' a numérica, 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)

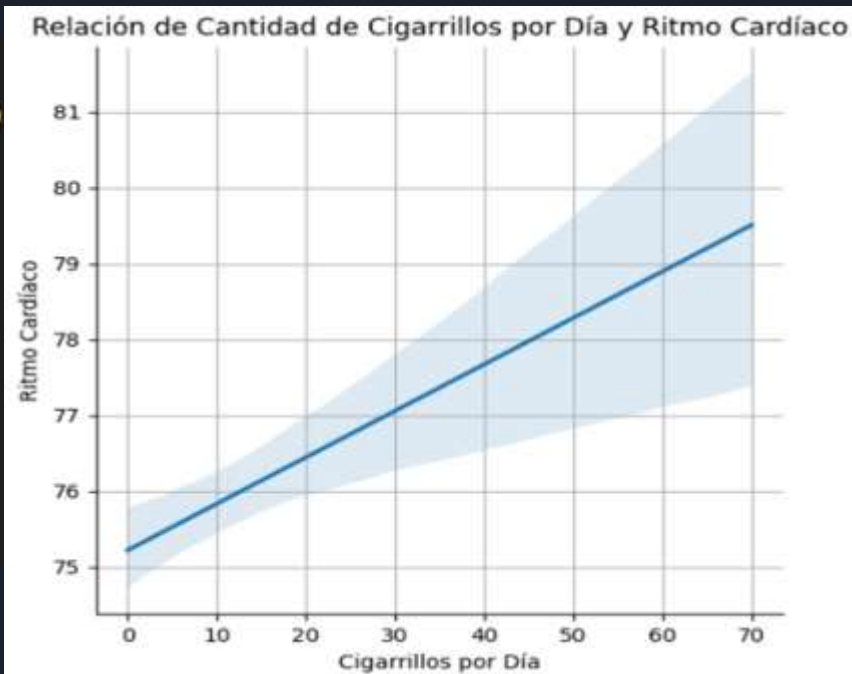
# Ahora puedes intentar crear el gráfico de nuevo, asegurándote que no hay valores NaN que puedan afectar la regresión
import seaborn as sns
import matplotlib.pyplot as plt

# 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)

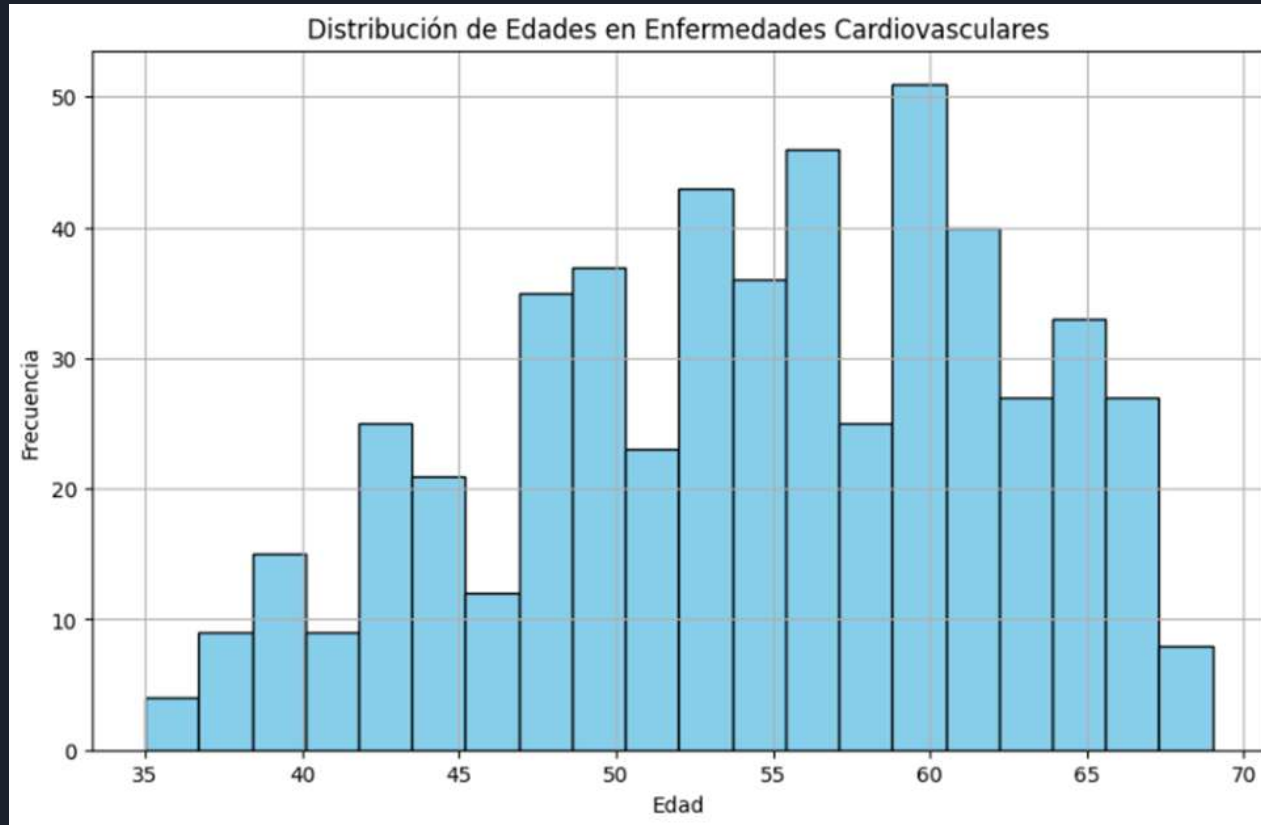
# Añadir título y etiquetas de ejes
plt.title('Relación de Cantidad de Cigarrillos por Día y Ritmo Cardíaco')
plt.xlabel('Cigarrillos por Día')
plt.ylabel('Ritmo Cardíaco')

# Agregar cuadrículas al gráfico
plt.grid(True)

# Mostrar el gráfico
plt.show()
```



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



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

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"]

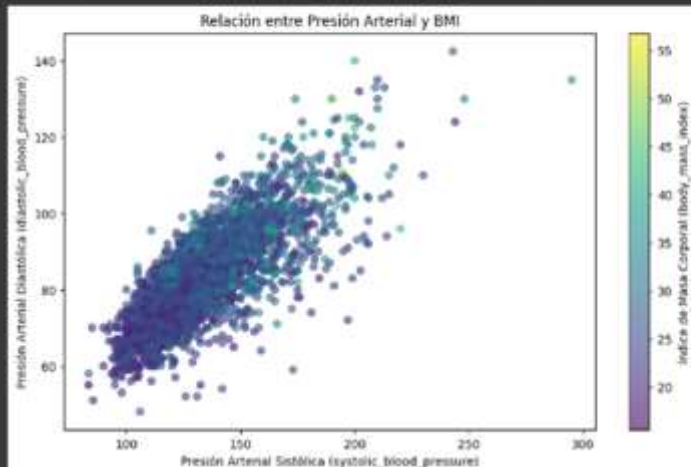
# Convertir la columna "BMI" a Numera (float)
df_cleaned["body_mass_index"] = df_cleaned["body_mass_index"].astype(float)

# Crear el gráfico de dispersión
plt.figure(figsize=(10, 8))
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 Diastólica (diastolic_blood_pressure)")
plt.ylabel("Presión Arterial Sistólica (systolic_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)")

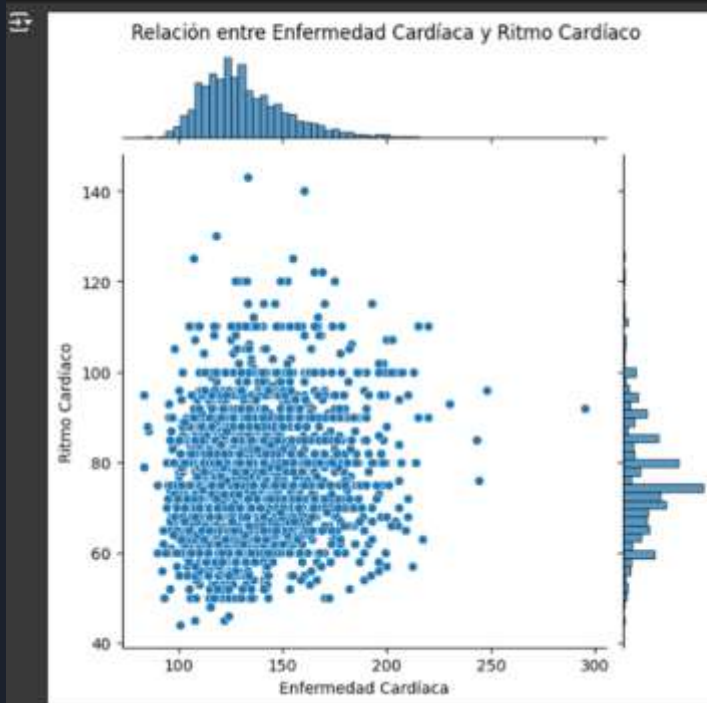
# Mostrar el gráfico
plt.show()
```



4. Relación entre Enfermedad Cardíaca y Ritmo Cardíaco

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()
```





MODELIZACIÓN

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

#Autocompletar 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 x 13 columns

Modelización Random Forest con todas las variables - matriz de confusión

```
[ ] #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



Modelización Random Forest con todas las variables - métricas



results

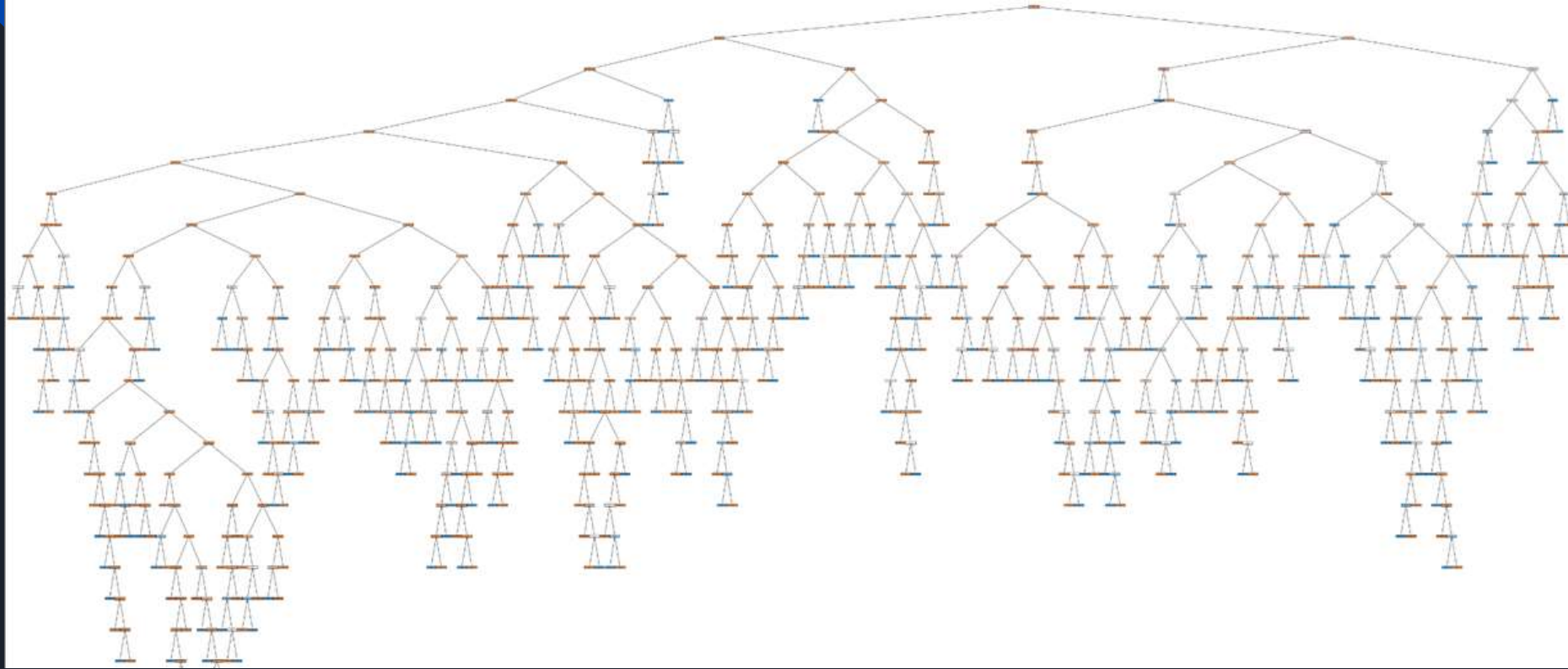


	Model	Accuracy	F1	Precision	Recall	AUC
--	-------	----------	----	-----------	--------	-----

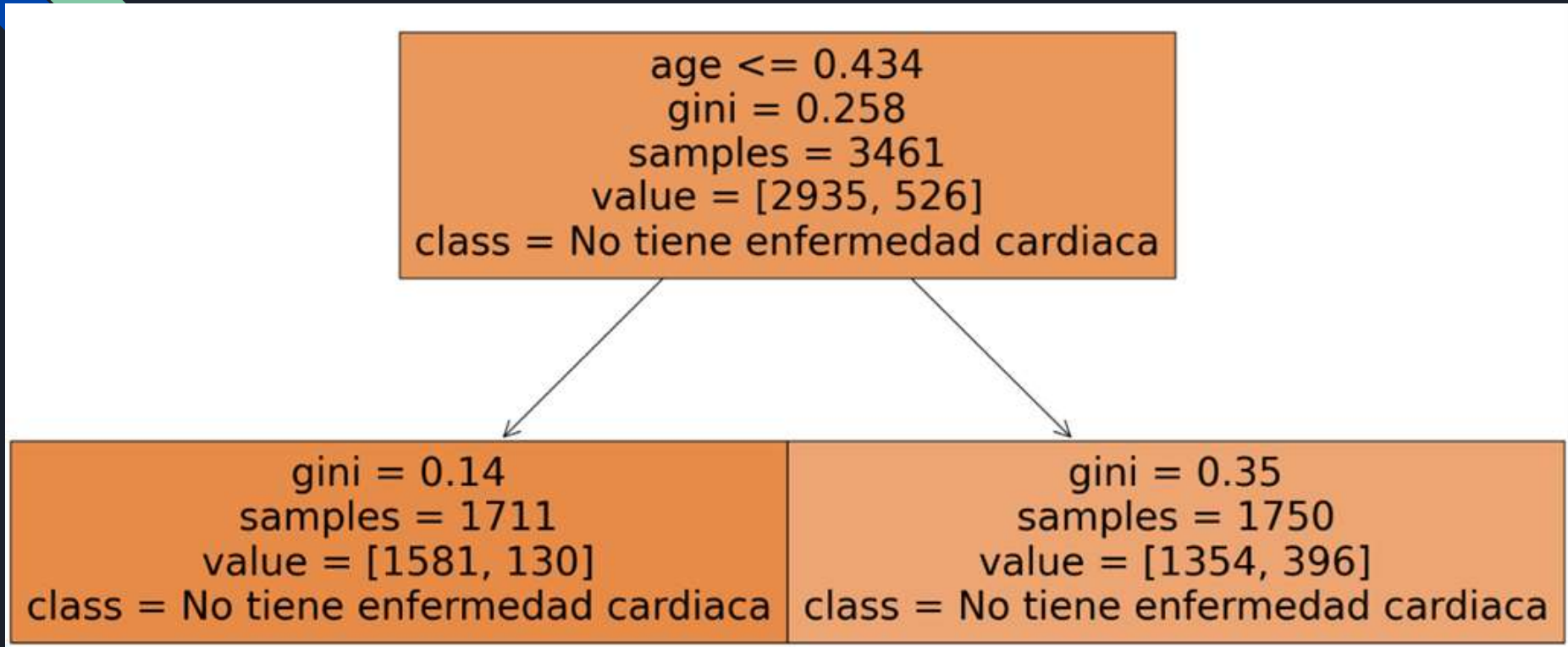
0	Random Forest	0.860	0.093	0.500	0.052	0.719
---	---------------	-------	-------	-------	-------	-------



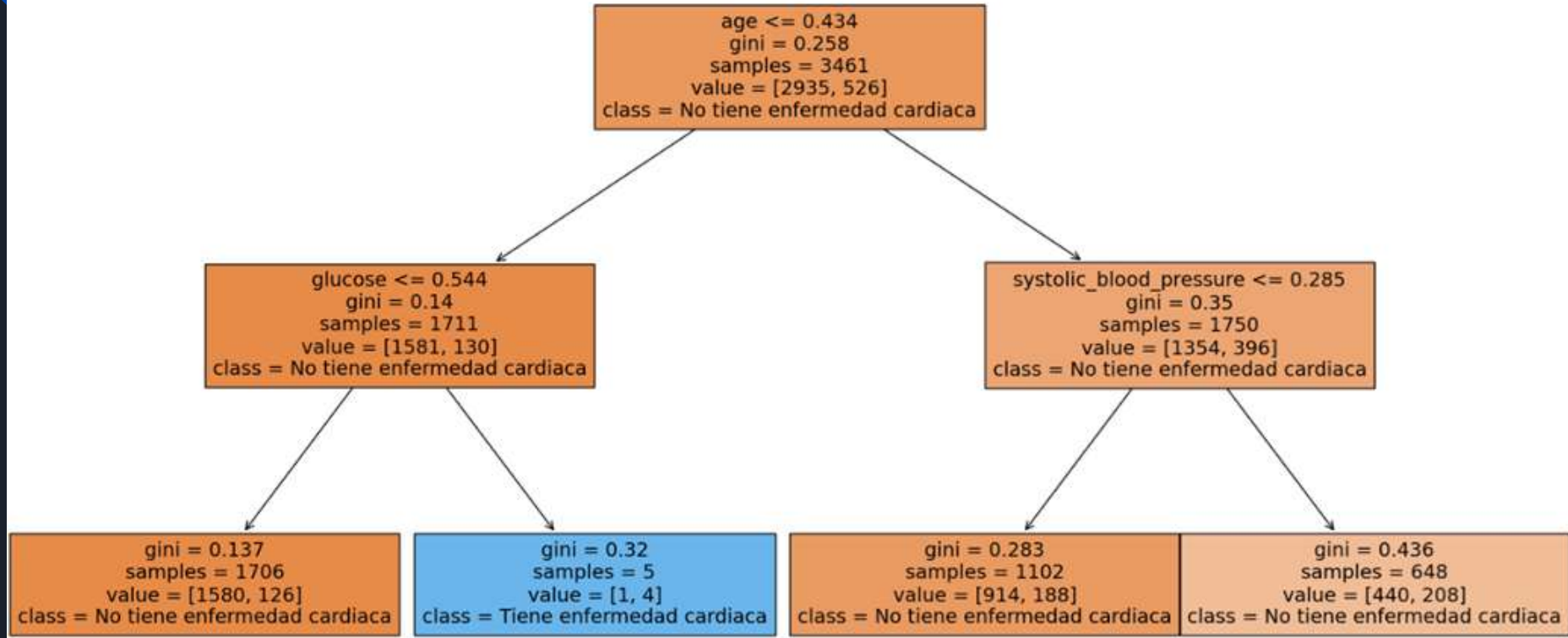
Modelización Random Forest con todas las variables - árboles de decisión



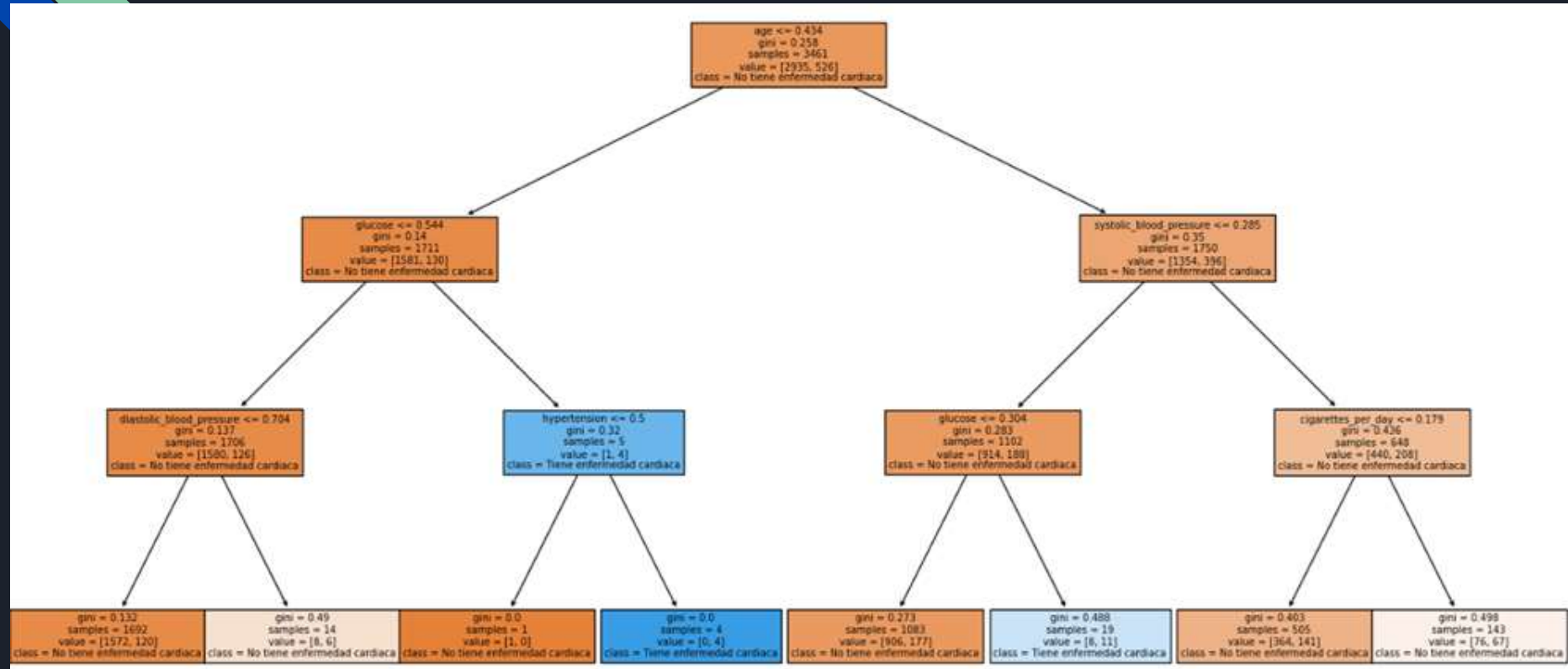
Modelización Random Forest con todas las variables - árboles de decisión



Modelización Random Forest con todas las variables - árboles de decisión



Modelización Random Forest con todas las variables - árboles de decisión



Modelización Random Forest con todas las variables - final

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	

Modelización Random Forest con Feature Selection – Técnica de Random Forest

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 x 13 columns

Modelización Random Forest con Feature Selection – Técnica de Random Forest

```
[ ] ## 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
```

Modelización Random Forest con Feature Selection – Técnica de Random Forest



```
#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')
```


Modelización Random Forest con Feature Selection – Técnica de Random Forest

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

#Autocompletar 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	0.694	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 x 13 columns

Modelización Random Forest con Feature Selection – Técnica de Random Forest - matriz de confusión

```
[ ] #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

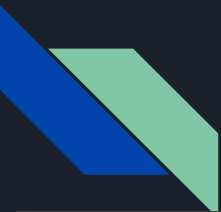
[ ] confusion_matrix_RandomForest = confusion_matrix(y_test, y_pred_RandomForest)

[ ] #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





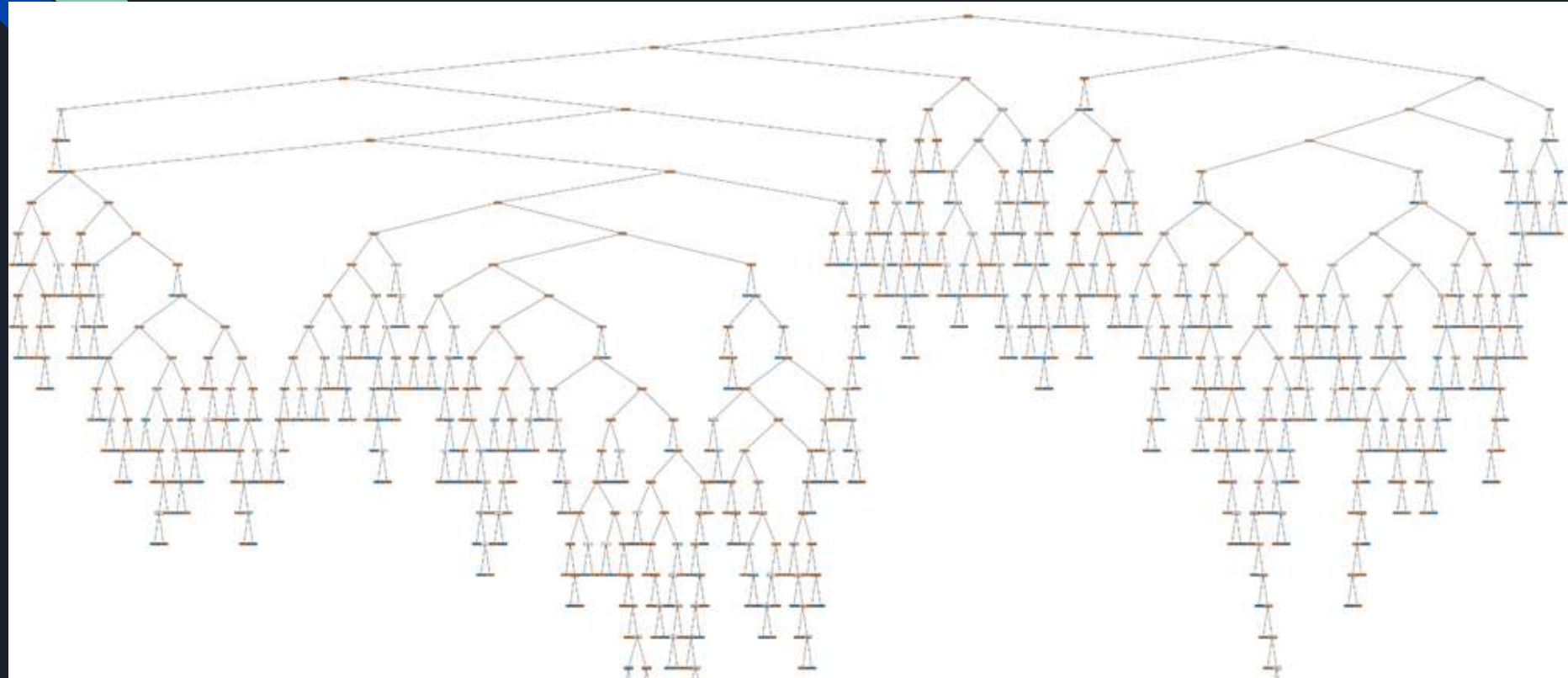
```
[ ] results
```

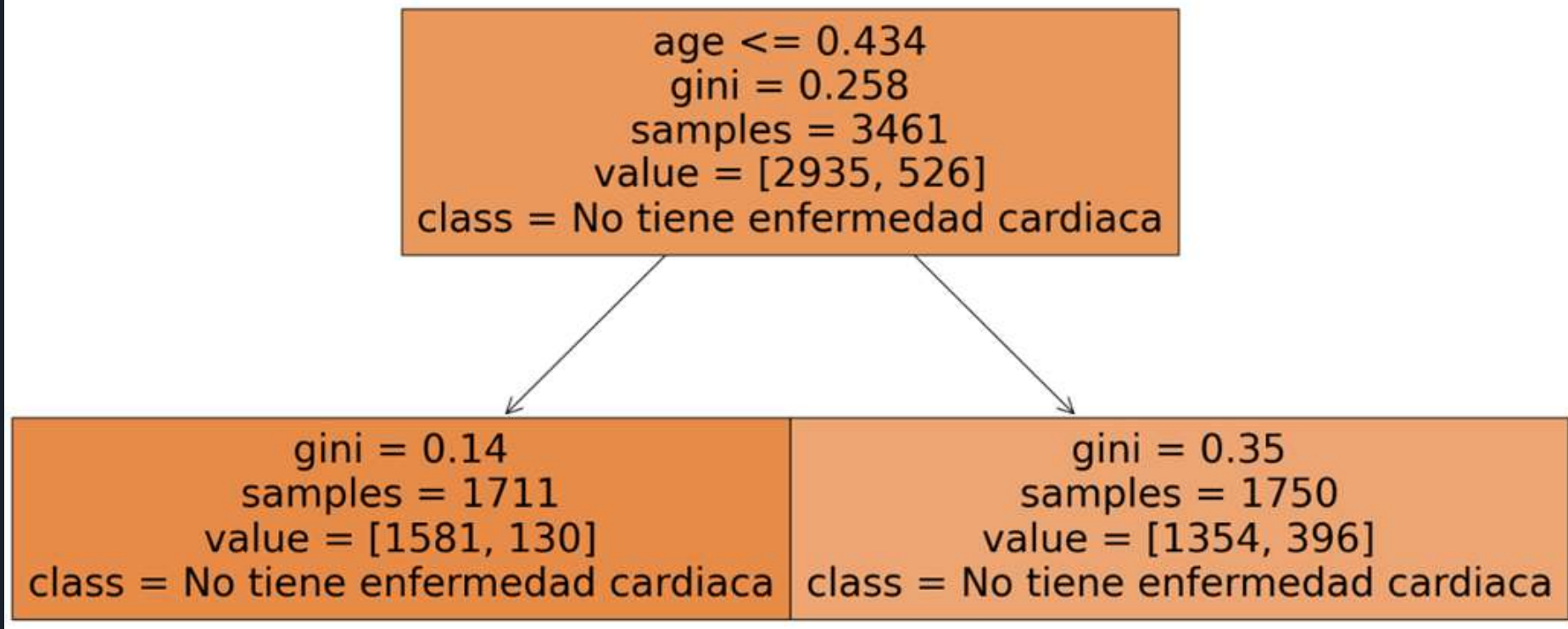


	Model	Accuracy	F1	Precision	Recall	AUC
--	-------	----------	----	-----------	--------	-----

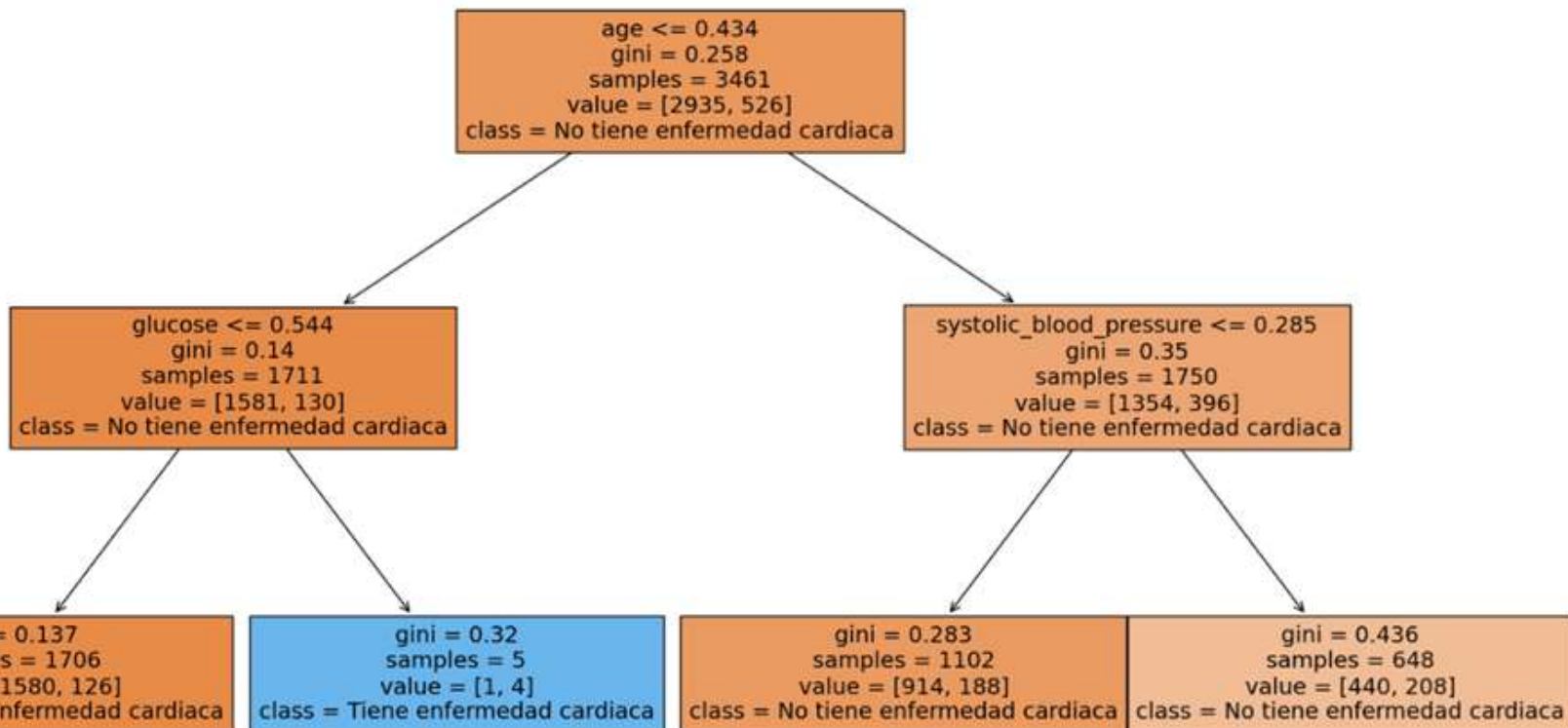
0	Random Forest	0.861	0.111	0.545	0.062	0.707
---	---------------	-------	-------	-------	-------	-------

Modelización Random Forest con Feature Selection – Técnica de Random Forest - árboles de decisión

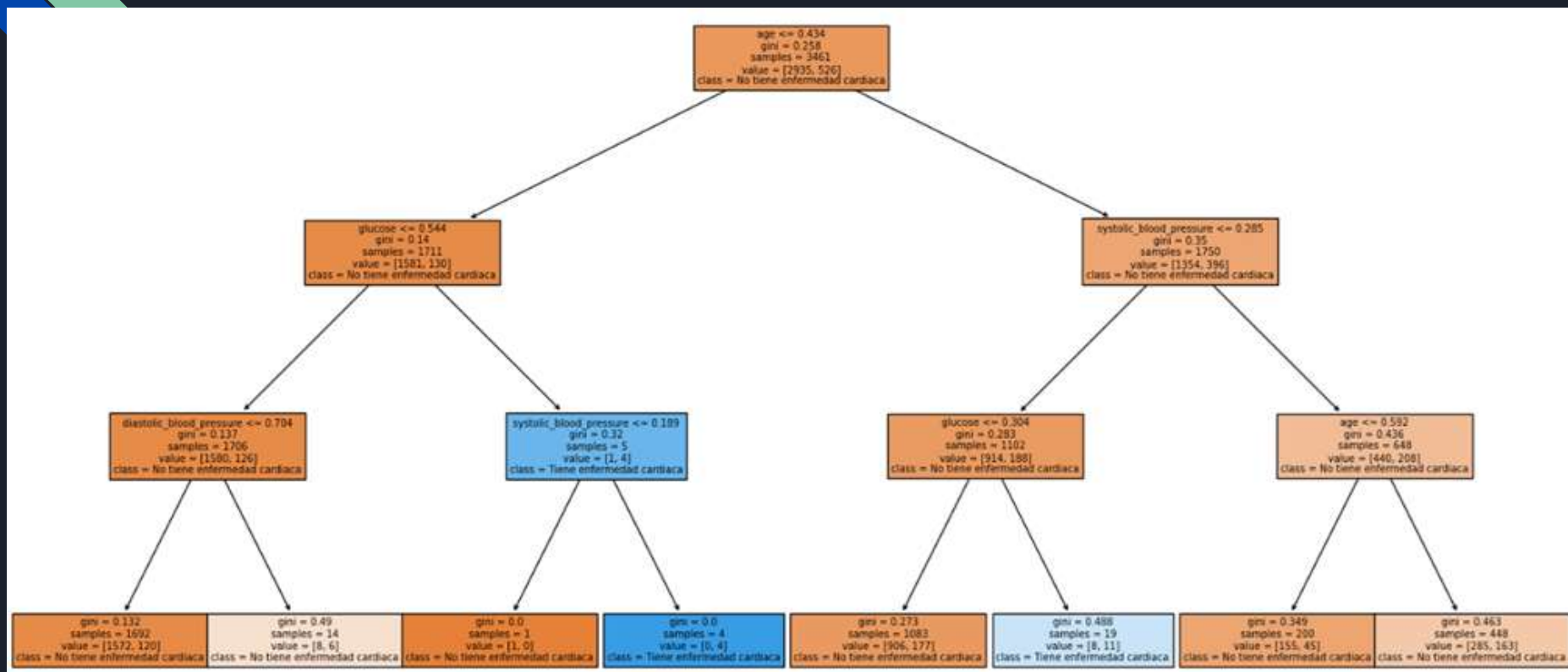




Modelización Random Forest con Feature Selection – Técnica de Random Forest - árboles de decisión



Modelización Random Forest con Feature Selection – Técnica de Random Forest - árboles de decisión



Modelización Random Forest con Feature Selection – Técnica de Random Forest - final

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

Comparación entre los dos modelos vistos anteriormente

Random Forest con todas las variables

	Predicción NO	Predicción 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

	Predicción NO	Predicción 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

Grabar Fin de Fase del modelo ganador



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



OPTIMIZACIÓN DEL MODELO GANADOR

Optimización del modelo ganador - 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

#Autocompletar 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_optimization = pd.read_pickle('/content/drive/My Drive/BPA/TP/df_heart_modelizacion_nuevo_variabiles_actualizadas.pickle')
df_heart_optimization
```

	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

Optimización del modelo ganador - Random Forest con Feature Selection – Técnica de Random Forest (GRID SEARCH)

GRID SEARCH


```
[ ] grid_parameters = { 'criterion'      : ['gini','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)
```

Optimización del modelo ganador - Random Forest con Feature Selection – Técnica de Random Forest (GRID SEARCH) - Mejores parámetros

```
 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}
```

Optimización del modelo ganador - Random Forest con Feature Selection – Técnica de Random Forest (GRID SEARCH) - Mejores parámetros

```
[ ] 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
```



PRESENTACIÓN DE RESULTADOS FINALES DESPUÉS DE OPTIMIZAR EL MODELO GANADOR

Resultados finales del modelo ganador Random Forest con Feature Selection – Técnica de Random Forest



#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

```
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])
```


Resultados finales del modelo ganador Random Forest con Feature Selection – Técnica de Random Forest - métricas

```
[ ] 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



INTERPRETACIÓN Y CONCLUSIONES DE LOS RESULTADOS FINALES

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

	Prediction 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

CONCLUSIONES



Conclusiones



RECOMENDACIONES



RECOMENDACIONES



EXPLICACIÓN DEL DASHBOARD DE HEART DISEASE POWER BI



¡GRACIAS!