# Tarea 4 Tema 2: Random Forest

Juan Manuel García Moyano
IABD
Informática y comunicaciones

# Índice

# 1. Importación de librerías necesarias

He importado las librerías necesarias con el fin de entrenar un modelo con arboles de decisión. Además, todas las funcionalidades para realizar todos los apartados que no están directamente relacionado con el entrenamiento.

```
from google.colab import drive
from sklearn import preprocessing
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, ConfusionMatrixDisplay, roc_curve,
roc_auc_score
from sklearn.model_selection import TimeSeriesSplit, train_test_split,
cross_val_score, KFold

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

# 2. Preproceso

## 2.1 Importación de los datos del dataset

La importación de los datos lo he realizado desde el drive. Para ello subí el dataset, luego monté mi drive en colab y leí el csv.

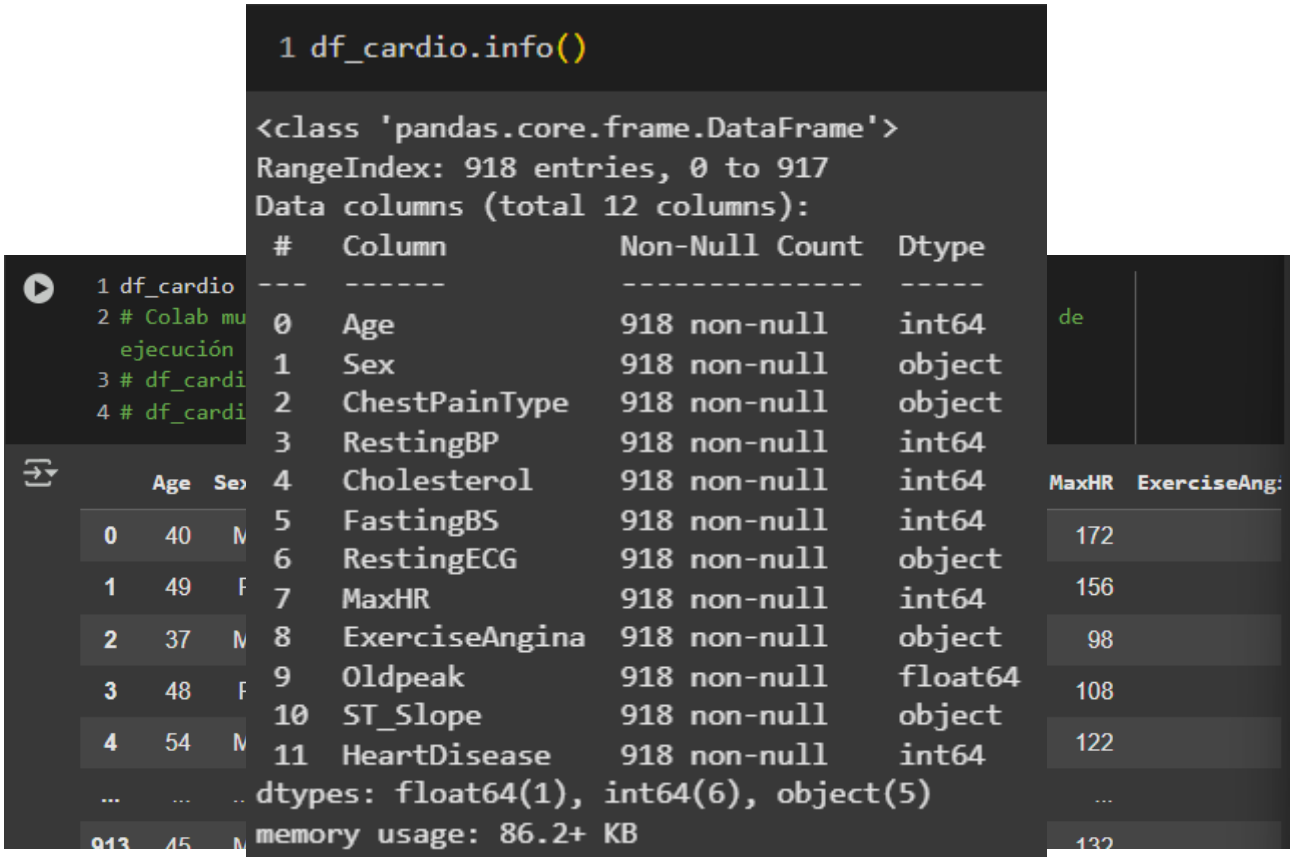
```
drive.mount("/content/drive")

df_cardio = pd.read_csv("/content/drive/My Drive/IABD/SAA/datasets/Tema 2/
cardio.csv")
```

## 2.2 Mostrar las primeras y últimas filas del dataframe importado

En colab, si ponemos en la última línea una variable muestra las primeras y últimas filas pero si se hace desde un entorno diferente el cual no sea jupyter, deberíamos utilizar los métodos head y tail.

## 2.3 Estudio de los atributos

Realizo un estudio de los atributos como viendo el tipo de dato y las medidas estadísticas.




## 2.4 Mostrar parámetros estadísticos de los datos (media, desviación típica, cuartiles, etc.)

Un dataframe tiene un método llamado describe que nos muestra los datos estadísticos de todos los atributos. Se debe de tener en cuenta que los datos estén en formato numérico.

```
1 df_cardio_copia.describe()
```

| | Age | Sex | ChestPainType | RestingBP | Cholesterol | FastingBS | RestingECG | |
|---|---|---|---|---|---|---|---|---|
| count | 918.000000 | 918.000000 | 918.000000 | 918.000000 | 918.000000 | 918.000000 | 918.000000 | 91 |
| mean | 53.510893 | 0.789760 | 0.781046 | 132.396514 | 198.799564 | 0.233115 | 0.989107 | 13 |
| std | 9.432617 | 0.407701 | 0.956519 | 18.514154 | 109.384145 | 0.423046 | 0.631671 | 2 |
| min | 28.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 6 |
| 25% | 47.000000 | 1.000000 | 0.000000 | 120.000000 | 173.250000 | 0.000000 | 1.000000 | 12 |
| 50% | 54.000000 | 1.000000 | 0.000000 | 130.000000 | 223.000000 | 0.000000 | 1.000000 | 13 |
| 75% | 60.000000 | 1.000000 | 2.000000 | 140.000000 | 267.000000 | 0.000000 | 1.000000 | 15 |
| max | 77.000000 | 1.000000 | 3.000000 | 200.000000 | 603.000000 | 1.000000 | 2.000000 | 20 |

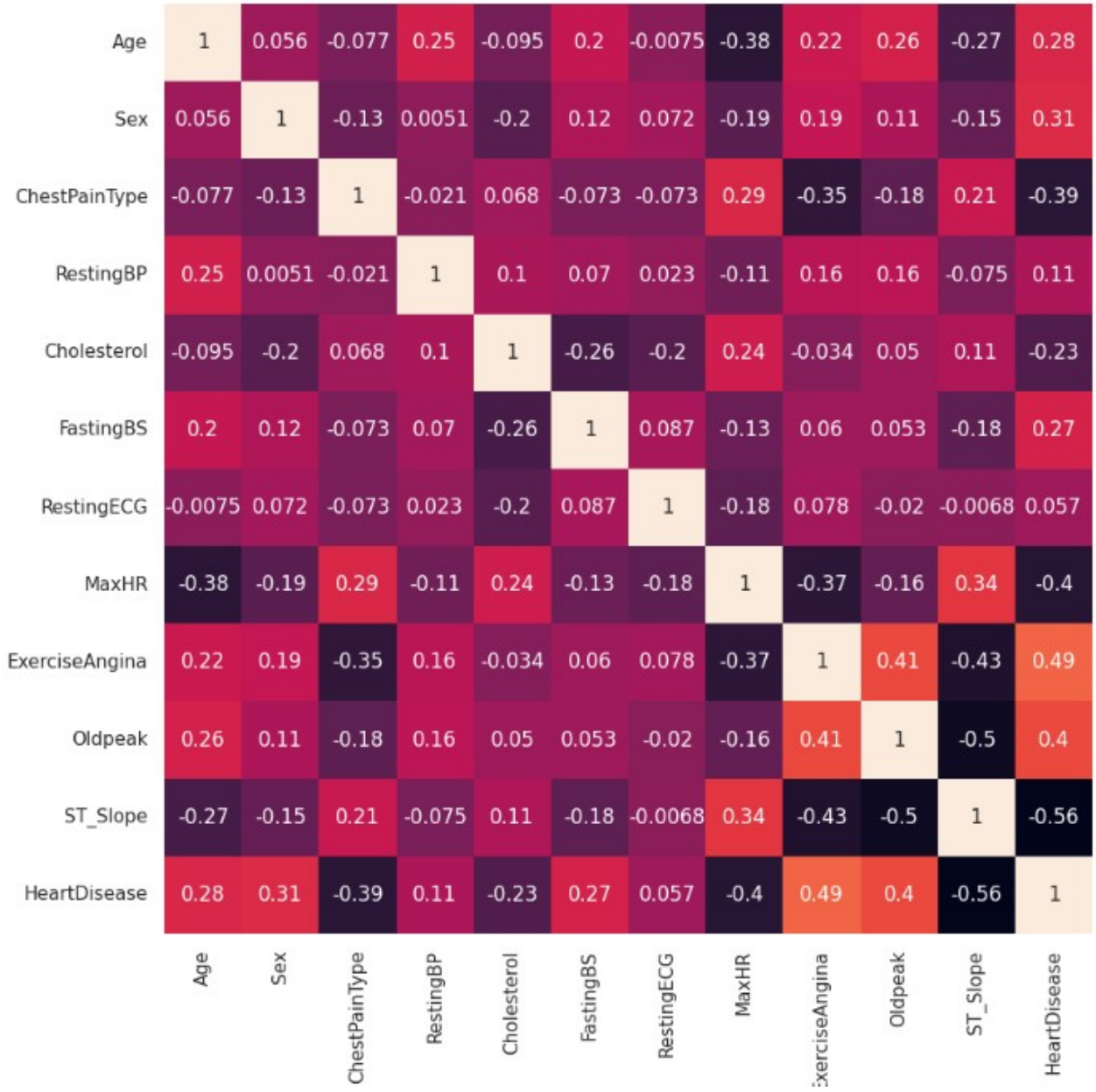
## 2.5 Mostrar un mapa de calor que indique la correlación entre variables

Para realizar el mapa de calor he utilizado la librería seaborn. Para ello tuve de indicar la dimensión del mapa porque en un principio no se veía muy bien, ya que, los nombres de los atributos se juntaban. Por ello, primero lo redimensiono y después lo muestro con el método headmap(dataframe.corr(), square = True, annot =True). Vamos a desglosar los parámetros:

- dataframe.corr(): calcula la matriz de correlación entre las columnas numéricas del dataframe. La correlación mide la relación lineal entre dos variables, con valores que van desde:
  - 1: correlación positiva perfecta (ambas variables aumentas juntas).
  - -1: correlación negativa perfecta (una variable aumenta mientras la otra disminuye).
  - 0: no hay relación. Estas nos indican que deben ser eliminadas para el entrenamiento.
- square = True: fuerza que cada celda mapa de calor tenga forma cuadrada.
- Annot = True: muestra los valores numéricos de la correlación dentro de cada celda.

```
1 sns.set(rc={"figure.figsize": (20, 10)})
2 sns.heatmap(df_cardio_copia.corr(), square = True, annot = True)
```

```
<Axes: >
```




## 2.6 Transformación de los atributos

En este apartado se va a modificar los atributos de tal forma que buscaremos que el modelo pueda realizar un aprendizaje más realista y eliminar todo el sesgo posible. Primero se transformarán todas las características con la valores categóricas.

```
1 data_norm = pd.get_dummies(df_cardio)
2 data_norm
```

| rtDisease | Sex_F | Sex_M | ChestPainType_ASY | ... | ChestPainType_NAP | ChestPainType_TA | R |
|---|---|---|---|---|---|---|---|
| 0 | False | True | False | ... | False | False | |
| 1 | True | False | False | ... | True | False | |
| 0 | False | True | False | ... | False | False | |
| 1 | True | False | True | ... | False | False | |
| 0 | False | True | False | ... | True | False | |
| ... | ... | ... | ... | ... | ... | ... | |
| 1 | False | True | False | ... | False | True | |
| 1 | False | True | True | ... | False | False | |
| 1 | False | True | True | ... | False | False | |
| 1 | True | False | False | ... | False | False | |

Ahora los outliers se modificarán poniéndole la media del atributo.

```python
def detectarLimitesOutlier(df: pd.DataFrame, k: int = 3):
  Q1 = df.quantile(0.25)
  Q3 = df.quantile(0.75)
  IQR = Q3 - Q1
  xL = Q1 - k * IQR # Límite inferior
  xU = Q3 + k * IQR # Límite Superior
  return (xL, xU)

def outliersCambiarMedia(df: pd.DataFrame, k = 3) -> pd.DataFrame:
    df_numeric = df.select_dtypes(include=[np.number])

    xL, xU = detectarLimitesOutlier(df_numeric, k)

    outliers = (df_numeric < xL) | (df_numeric > xU)

    df_sin_outliers = df.copy()

    for columna in df_numeric.columns:
        media = df_numeric[columna].mean()
        if df_sin_outliers[columna].dtype == 'int64':
          df_sin_outliers.loc[outliers[columna], columna] = int(media)
        else:
          df_sin_outliers.loc[outliers[columna], columna] = media

    return df_sin_outliers
```

```python
data_norm = outliersCambiarMedia(data_norm)
data_norm
```

Para finalizar, los datos numéricos se van a escalar.

```
data_norm = data_norm.drop("RestingBP", axis=1)
data_norm_num = data_norm.select_dtypes(include=[np.number])
scaler = preprocessing.MinMaxScaler()

data_norm_num = data_norm_num.drop(columns=["HeartDisease"], axis=1)

data_norm[data_norm_num.columns] = scaler.fit_transform(data_norm_nu

data_norm

```

| | Age | Cholesterol | FastingBS | MaxHR | Oldpeak | HeartDisease |
|---|---|---|---|---|---|---|
| **0** | 0.244898 | 0.546314 | 0.0 | 0.788732 | 0.317073 | 0 |
| **1** | 0.428571 | 0.340265 | 0.0 | 0.676056 | 0.439024 | 1 |
| **2** | 0.183673 | 0.534972 | 0.0 | 0.267606 | 0.317073 | 0 |
| **3** | 0.408163 | 0.404537 | 0.0 | 0.338028 | 0.500000 | 1 |
| **4** | 0.530612 | 0.368620 | 0.0 | 0.436620 | 0.317073 | 0 |
| **...** | ... | ... | ... | ... | ... | ... |
| **913** | 0.346939 | 0.499055 | 0.0 | 0.507042 | 0.463415 | 1 |
| **914** | 0.816327 | 0.364839 | 0.0 | 0.570423 | 0.731707 | 1 |
| **915** | 0.591837 | 0.247637 | 0.0 | 0.387324 | 0.463415 | 1 |
| **916** | 0.591837 | 0.446125 | 0.0 | 0.802817 | 0.317073 | 1 |
| **917** | 0.204082 | 0.330813 | 0.0 | 0.795775 | 0.317073 | 0 |

## 2.7 Separar datos entre datos de entrada y etiquetas (resultados)

El estudio de se va realizar con todos los atributos excepto RestingBP y HeartDisease. Para ello divido los datos en una variable (x) que va a tener los datos con los atributos anteriores y otra variable (y) con los datos del atributo objetivo.

```
x_df_cardio = data_norm.drop(["HeartDisease"], axis=1)

y_df_cardio = data_norm['HeartDisease']
```

### 2.8 Separar datos entre entrenamiento y prueba (usando un 80% para entrenamiento y 20% para test)

Para separar los datos utilizo la funcionalidad train_test_split. A este le paso dos variables una con los datos de la variable x y la otra variable es y. X tiene los datos sin los atributos anteriormente mencionados y la otra variable que es la que tiene todos los datos del atributo objetivo. Además, a esta función le paso los parámetros random_state (1) y test_size (0.20). Esto lo que hace que la división sea reproducible y representa la proporción del dataframe total asignada al conjunto de prueba, respectivamente.

Mediante el desempaquetado múltiple, lo que devuelve la función se lo asigno a 4 variables. Estas son:

- x_entrenamiento: tiene el 80% de los datos que van a servir para entrenar el modelo.
- x_test: tiene el otro 20% de los datos que van a servir hacer la predicción.
- y_entrenamiento: tiene el 80% de los datos que van a servir para entrenar el modelo.
- y_test: tiene el otro 20% de los datos que van a servir para calcular la probabilidad de acierto del modelo.

```
1 x_entrenamiento, x_test, y_entrenamiento, y_test = train_test_split
  (x_df_cardio, y_df_cardio, random_state=1, test_size=0.2)
```

# 3. Entrenamiento y predicción

## 3.1 Elegir, instanciar y entrenar el modelo

Primero instancio el modelo deseado (clasificación). Después, lo entreno con las variables anteriores de entrenamiento.

```
modelo = RandomForestClassifier(n_estimators = 100, random_state=1)
modelo.fit(x_entrenamiento, y_entrenamiento)
```

### 3.2 Realizar una predicción con los datos de prueba

Al modelo ya entrenado hago que haga una predicción con los datos de x_test.

```
y_prediccion = modelo.predict(x_test)
```

# 4. Evaluación

Este ejercicio es un problema de clasificación, por eso realizo estos apartados.

### 4.1 Mostrar el porcentaje de elementos correctamente clasificados

Con la ayuda de la función accuracy_score, que me sirve para comprobar la probabilidad de acertar que tiene el modelo. A esta función le paso los datos y_test (datos del atributo calidad) y la predicción realizada en el anterior apartado. Muestro la probabilidad de 0 a 1.

```
print("La probabilidad de acertar es: ", accuracy_score(y_test, y_prediccion))
```

```
La probabilidad de acertar es:  0.8804347826086957
```

## 4.2 Mostrar la predicción realizada (imprimir la variable con la predicción)

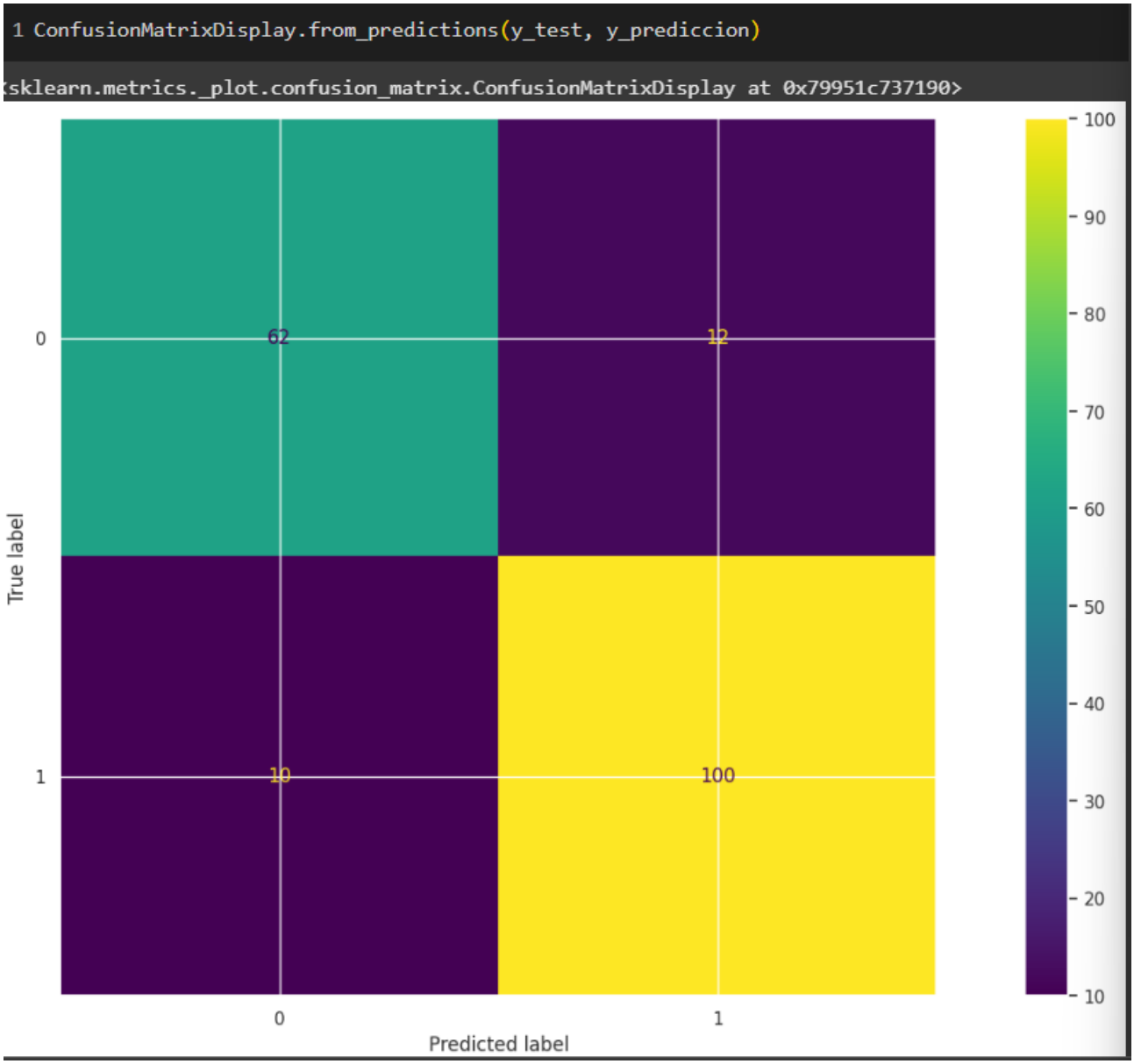
Muestro los datos que he realizado con la predicción.

```
1 print(y_prediccion)
```

```
[1 1 1 1 1 0 0 0 0 1 0 1 1 1 1 0 1 0 1 0 1 0 1 0 1 1 0 1 1 0 0 0 0 0 1 1 1
 1 0 0 1 1 1 0 0 0 1 0 0 1 0 1 1 0 0 0 1 0 1 0 1 1 1 0 0 1 1 1 0 1 1 0 1 1
 0 1 0 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 0 0 1 1 0 0
 1 0 0 0 0 1 0 0 1 1 1 1 0 1 0 1 0 1 1 0 1 1 1 1 1 1 1 0 1 1 0 0 1 1 0 1 1
 0 1 1 0 0 0 0 1 0 1 1 1 1 1 1 0 1 1 0 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 0 0 1]
```

## 4.3 Representar gráficamente la clasificación obtenida (matriz de confusión)

Aquí utilizo el método de ConfusionMatrixDisplay que es from_predictions. Se va a encargar de mostrarnos el mapa de confusión comparando los datos de test (y_test) y los datos de la predicción (y_prediccion).




## 4.4 Curva ROC y el AUC.

La curva de ROC es un gráfico que muestra el rendimiento de un modelo de clasificación al variar el umbral de decisión. En lugar de elegir un umbral fijo (por ejemplo, 0.5), evalúa cómo cambian las tasas de verdaderos positivos y falsos positivos con diferentes umbrales.

**El AUC**

Es el Área Bajo la Curva ROC. Es un valor numérico que mide el desempeño del modelo.

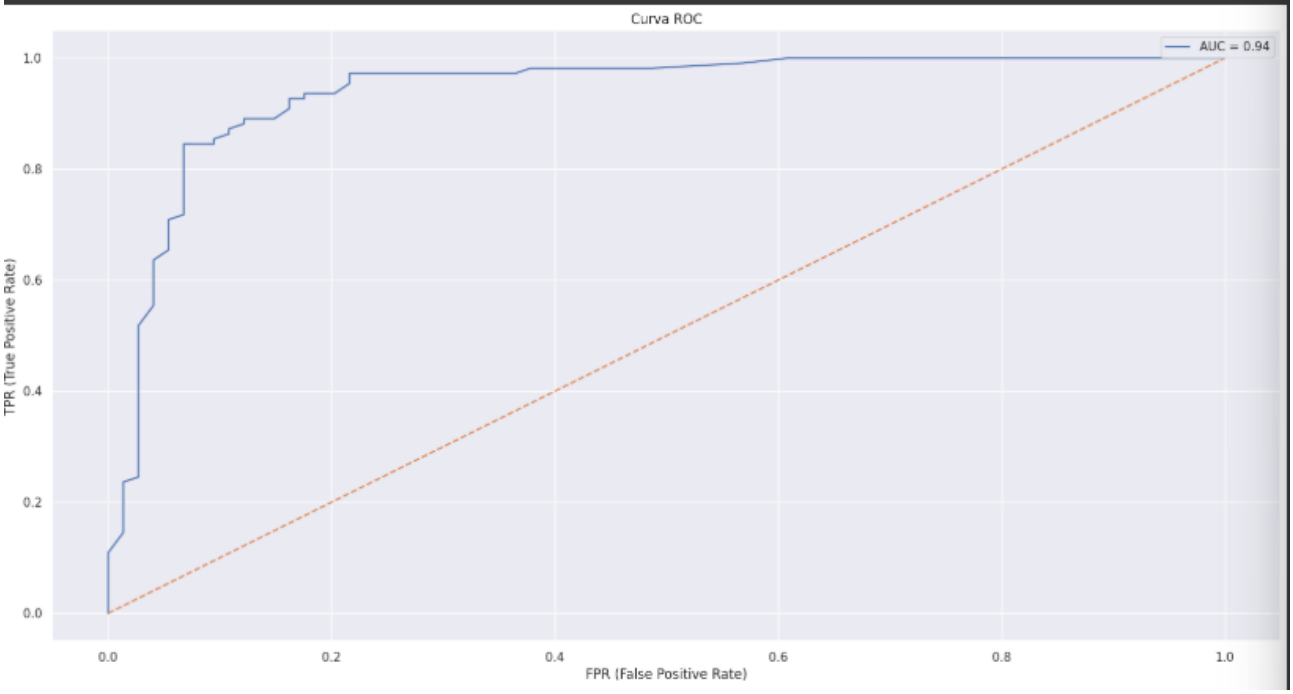
•Interpretación:

•AUC = 1.0: Clasicicación perfecta.

•AUC = 0.5: Clasificador aleatorio (sin capacidad predictiva).

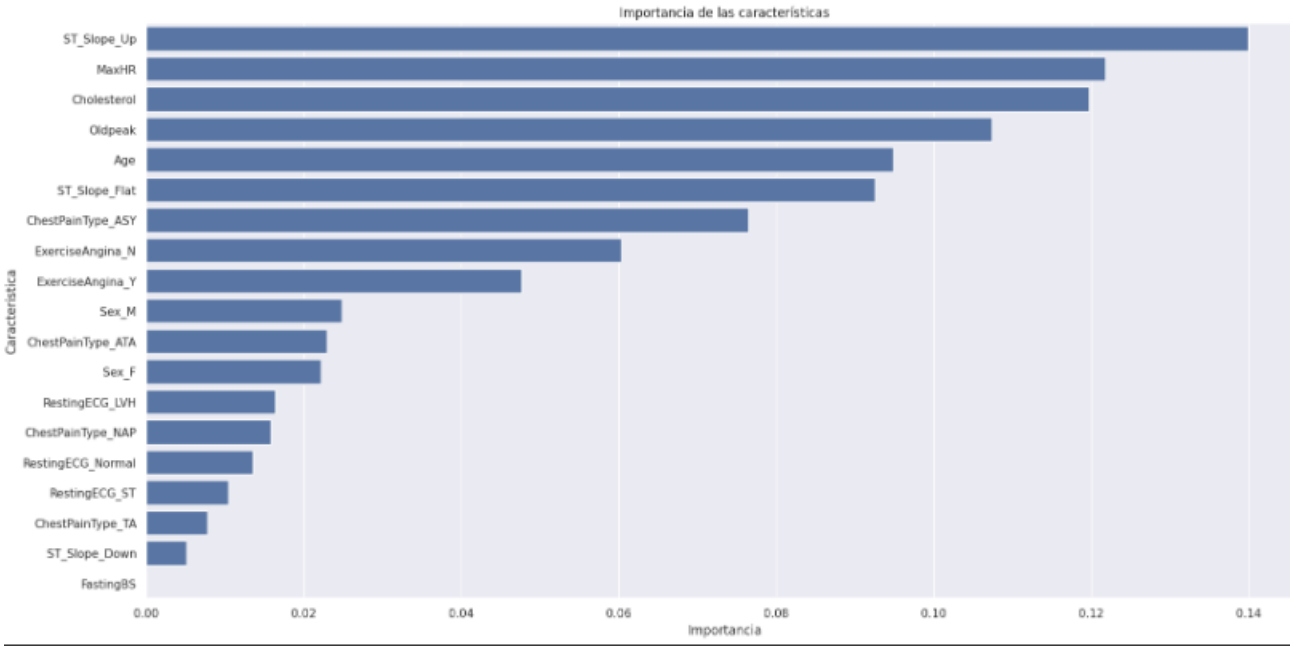
•AUC < 0.5:Rendimiento peor que un clasificador aleatorio (esto indica que el modelo está prediciendo al revés, lo cuál generalmente significa que hay un problema en el entrenamiento o los datos).

```python
y_prob = modelo.predict_proba(x_test)[:, 1]

fpr, tpr, _ = roc_curve(y_test, y_prob)
auc = roc_auc_score(y_test, y_prob)

plt.plot(fpr, tpr, label=f"AUC = {auc:.2f}")
plt.plot([0, 1], [0, 1], linestyle="--")
plt.xlabel("FPR (False Positive Rate)")
plt.ylabel("TPR (True Positive Rate)")
plt.title("Curva ROC")
plt.legend()
plt.show()
```



### 4.5 Importancia de las características

```
importancias = modelo.feature_importances_
caracteristicas = x_entrenamiento.columns

df_importancias = pd.DataFrame({'Caracteristica': caracteristicas,
 'Importancia': importancias})
df_importancias = df_importancias.sort_values(by='Importancia', ascending=False)

sns.barplot(x='Importancia', y='Caracteristica', data=df_importancias)
plt.title("Importancia de las características")
plt.show()
```




# 5. Optimización de hiperparámetros

Antes de realizar la optimización de hiperparámetros, al mostrar la importancia de las características con respecto del modelo podemos observar el atributo FastingBS tiene una importancia de 0.00. Entonces una de las optimizaciones que se van a realizar va a ser eliminar dicho atributo.

## 5.1 Cada ejecución anterior se deberá hacer usando validación cruzada (por ejemplo n_splits = 5). Con ello obtendremos una medida de bondad del modelo (accuracy_score o mean_absolute_error), como lo ejecutaremos 5 veces, calcularemos la media de esas 5 ejecuciones.

Para continuar con la optimización se va a entrenar dos modelos, uno con gini y otro con entropy. Además, se va a hacer usando validación cruzada (por ejemplo n_splits = 5) y como lo ejecutaremos 5 veces, calcularemos la media de esas 5 ejecuciones.

- Gini

Gini

```
kf = KFold(n_splits=5, shuffle=True, random_state=1)

df_opt = pd.DataFrame(data={}, columns=["max_depth", "media_prob_acertar"])

for i in range(1, 501, 10):
  modelo = RandomForestClassifier(n_estimators = i, random_state=1)
  puntuaciones = cross_val_score(modelo, x_entrenamiento, y_entrenamiento,
  cv=kf, scoring='accuracy')


  df_opt.loc[len(df_opt)] = [i, np.mean(puntuaciones)]

print("Criterio elegido Gini: ")
df_opt
```

```
Criterio elegido Gini:
```

| | max_depth | media_prob_acertar |
|---|---|---|
| 0 | 1.0 | 0.769816 |
| 1 | 11.0 | 0.825673 |
| 2 | 21.0 | 0.820231 |
| 3 | 31.0 | 0.828404 |

- Entropy

```python
kf = KFold(n_splits=5, shuffle=True, random_state=1)

df_opt = pd.DataFrame(data={}, columns=["max_depth", "media_prob_acertar"])

for i in range(1, 501, 10):
  modelo1 = RandomForestClassifier(n_estimators = i,criterion='entropy',
  random_state=1)
  puntuaciones = cross_val_score(modelo1, x_entrenamiento, y_entrenamiento,
  cv=kf, scoring='accuracy')


  df_opt.loc[len(df_opt)] = [i, np.mean(puntuaciones)]

print("Criterio elegido Entropy: ")
df_opt
```

```
Criterio elegido Entropy:
```

| | max_depth | media_prob_acertar |
|---|---|---|
| 0 | 1.0 | 0.771158 |
| 1 | 11.0 | 0.836548 |
| 2 | 21.0 | 0.835216 |
| 3 | 31.0 | 0.835206 |
| 4 | 41.0 | 0.842009 |
| 5 | 51.0 | 0.847461 |

## 5.3 Finalmente los parámetros elegidos serán los que den mejor media de esas medidas anteriormente nombradas.

Si vemos los resultados de ambos modelos, vemos que los mejores son de entropy. Además, en ambos modelos a partir del n_estimators 50 empiezan a estancarse los resultados e incluso puede ser que en número elevados esté empezando el sobre-ajuste. Los mejores hiperparámetros son entropy y n_estimators 51. Aunque la probabilidad se puede asemejar a los valores de gini, hay que tener en cuenta que el número de arboles es 51 con una probabilidad de 0.847 mientras que con gini se necesitan un total de 101 y alcanzaría un 0.842 haciendo que el modelo sea más complejo.

## 5.4 Una vez obtenidos esos parámetros óptimos los aplicaremos al problema en cuestión y mostraremos los resultados.

Realizo un modelo con los mejores hiperparámetros que he mencionado en el anterior apartado.

```
modelo = RandomForestClassifier(n_estimators = 51, random_state=1,
criterion='entropy')
modelo.fit(x_entrenamiento, y_entrenamiento)

y_prediccion = modelo.predict(x_test)

print("La probabilidad de acertar es: ", accuracy_score(y_test, y_prediccion))
```

```
La probabilidad de acertar es:  0.8804347826086957
```