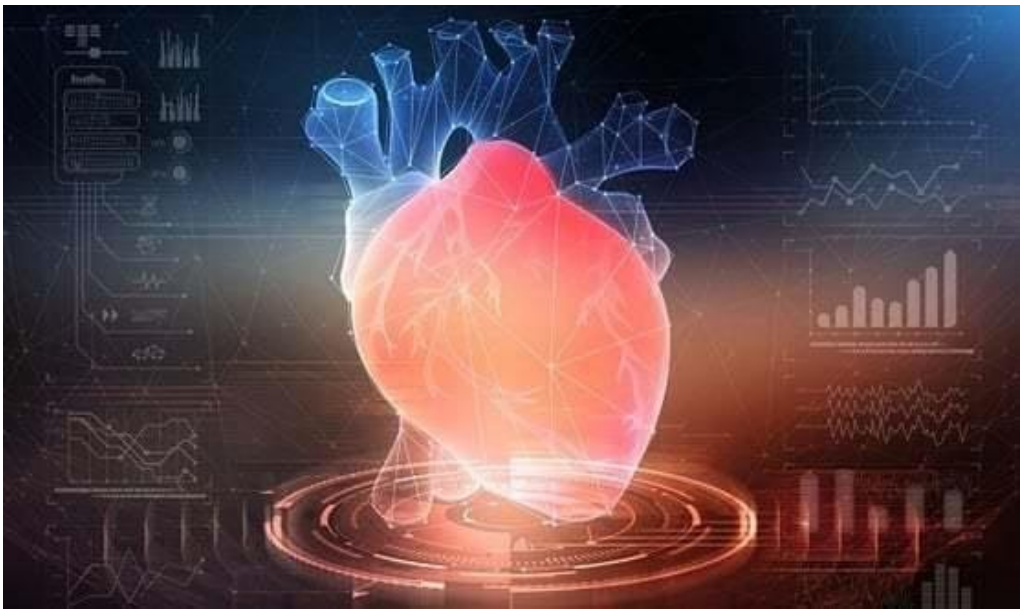
Proyecto Final de Data Science

“Predicción de una enfermedad coronaria”



Autor: Cristian Alberto Correa

Índice

Contenido

1) Abstract

1.1) Metas

En este estudio, presentamos un dataset que tiene como predecir la presencia de una enfermedad coronaria utilizando técnicas de Data Science. La enfermedad coronaria es una afección cardiovascular grave que afecta a millones de personas en todo el mundo y representa una de las principales causas de morbimortalidad.

El dataset utilizado en este estudio contiene una amplia variedad de características clínicas, recopiladas de una población diversa de pacientes. Estas características incluyen datos como:

- la edad (AGE),
- el género (SEX),
- tipo de dolor en el pecho (CP),
- presión sanguínea (trestbps),
- colesterol (CHOL),
- nivel de azúcar en sangre (FBS),
- resultado de electrocardiograma en reposo (RESTECG),
- frecuencia cardíaca máxima alcanza durante una prueba de esfuerzo (THALACH), paciente con angina (EXANG),
- descenso del segmento ST (OLDPEAK),
- pendiente del segmento ST durante la parte más exigente del ejercicio (SLOPE), resultado del flujo sanguíneo con tinte radioactivo (THAL),
- número de vasos sanguíneos principales coloreados por tinte radioactivo (CA),
- la variable objetivo en el dataset indica la presencia o ausencia de enfermedad coronaria (TARGET).

1.2) Objetivo

El objetivo principal de este proyecto es desarrollar un modelo predictivo preciso y confiable que pueda identificar tempranamente a los individuos en riesgo de desarrollar enfermedad coronaria, lo que permitiría una intervención médica oportuna y potencialmente reducir las complicaciones asociadas.

Para abordar este problema, se aplicarán diversas técnicas donde compararemos varios modelos y veremos cómo se comparta cada uno y cual tiene más probabilidad de éxitos dado lo que buscamos predecir

El análisis de este dataset y el desarrollo del modelo predictivo tienen el potencial de proporcionar una valiosa herramienta para la detección temprana y la prevención de

la enfermedad coronaria. Los resultados obtenidos podrían ayudar a los profesionales de la salud a identificar de manera más precisa a las personas con alto riesgo y a implementar estrategias de intervención adecuadas. Además, este estudio también puede revelar relaciones y patrones ocultos entre las características estudiadas, lo que podría conducir a una mejor comprensión de los factores de riesgo asociados con la enfermedad coronaria

1.3) Contexto comercial

El objetivo comercial de este proyecto es proporcionar una solución basada en datos que permita a las organizaciones comerciales identificar a las personas con mayor riesgo de enfermedad coronaria, con el fin de tomar medidas preventivas y ofrecer programas de salud personalizados para mejorar la salud de sus empleados y reducir los costos asociados, consiguiendo los siguientes beneficios:

- **Identificación temprana** de personas con alto riesgo de enfermedad coronaria, lo que permite la implementación de medidas preventivas y tratamientos personalizados.
- **Reducción de costos** para las compañías de seguros y proveedores de atención médica al detectar y tratar la enfermedad en etapas tempranas, evitando complicaciones más graves y costosas.
- **Mejora de la calidad de vida** de los pacientes al recibir un diagnóstico temprano y un tratamiento adecuado.

1.4) Contexto analítico

En nuestro caso buscamos responder algunas de las siguientes preguntas que por lo general suelen darse con respecto a este tipo de afección para entender como la misma afecta a la población en general;

1) ¿Cuál es el porcentaje de personas que tienen algún tipo de afección coronaria?
2) ¿Los hombres son más propensos a tener afecciones cardiacas?
3) ¿Qué tipo de variables nos pueden ayudar a hacer una prevención temprana?

El proyecto consiste en analizar el conjunto de datos de pacientes con cardiopatías y procesarlos adecuadamente. A través de distintos modelos que buscamos entrenar para redecir la presencia de una enfermedad cardíaca en un paciente, es decir, hablamos de un problema de clasificación, con características

de entrada como una variedad de parámetros, y la variable objetivo como una variable binaria, prediciendo si la enfermedad cardíaca está presente o no.

2) Adquisición de datos

El conjunto de datos ha sido tomado de UCI Machine Learning Repository https://archive.ics.uci.edu/dataset/45/heart+disease, esta base de datos contiene 76 atributos, pero todos los experimentos publicados en dicho repositorio hacen referencia al uso de un subconjunto de 14 de ellos, que es el que utilizaremos nosotros para abordar nuestro proyecto.
Tenemos alrededor de 300 pacientes y las variables recogidas están descritas a continuación.

- **[age] edad** (#)
- **[sex] sexo** : 1= Masculino, 0= Femenino (Binario)
- **[chest_pain] (cp)** tipo de dolor torácico (4 valores -Ordinal):
  - Valor 1: angina típica,
  - Valor 2: angina atípica,
  - Valor 3: dolor no anginoso,
  - Valor 4: asintomático
- **[resting_blood_pressure] (trestbps)** presión arterial en reposo (#)
- **[cholesterol] (chol)** colesterol sérico en mg/dl (#)
- **[fasting_blodd_sugar] (fbs)** glucemia en ayunas > 120 mg/dl(Binario)(1 = verdadero; 0 = falso)
- **[rest_ecg] (restecg)** resultados electrocardiográficos en reposo(valores 0,1,2)
- **[thalach] (thalach)** frecuencia cardiaca máxima alcanzada(#)
- **[exercise_induced_angina] (exang)** angina inducida por el ejercicio (binario) (1 = sí; 0 = no)
- **[st_depression] (oldpeak)** = depresión del ST inducida por el ejercicio en relación con el reposo (#)
- **[st_slope] (pendiente)** del segmento ST pico del ejercicio (ordinal) (Valor 1: pendiente ascendente , Valor 2: plano , Valor 3: pendiente descendente)
- **[number_of_vessels] (ca)** número de vasos principales (0-3, ordinal) coloreados por fluoroscopia
- **[maximum_heart_rate] (thal)** frecuencia cardiaca máxima alcanzada - (Ordinal): 3 = normal; 6 = defecto fijo; 7 = defecto reversible

3) Data Wrangling & EDA (Exploratory Data Analysis)

3.1) Datos iniciales

Primero importamos todas las librerías a utilizar:

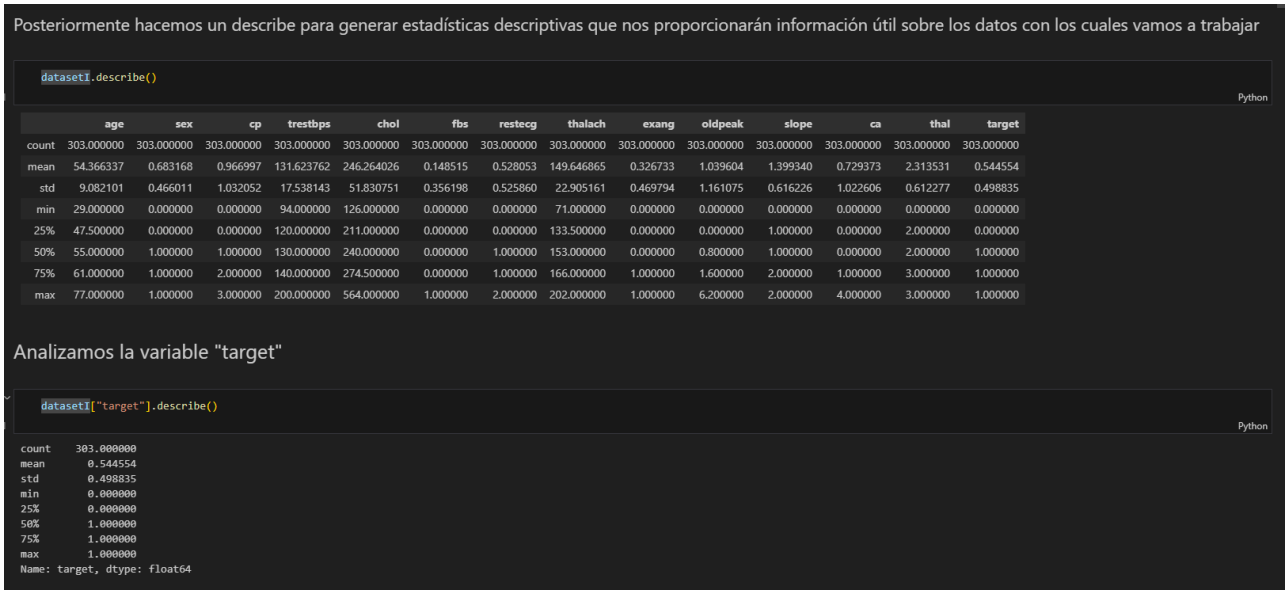
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import rcParams
from matplotlib.cm import rainbow
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

import seaborn as sns
from scipy import stats
```

Vamos a utilizar función **info()**, nos va a dar una descripción general de nuestro conjunto de datos, lo que nos va a ayudar a entender mejor su estructura y a tomar decisiones sobre cómo manipular o limpiar los datos en función de la información que obtenemos

```
datasetI.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

Posteriormente hacemos un **describe()**, la cual es una función que se utiliza para obtener un resumen estadístico de las columnas numéricas en el conjunto de datos.



	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	2.313531	0.544554
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	0.612277	0.498835
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	2.000000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	2.000000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000	1.000000

Los resultados obtenidos en primeria instancia nos indican lo siguiente:

- Count (Recuento): Nos muestra la cantidad de valores no nulos en cada columna. En este caso ya no tenemos la presencia de nulos y tenemos 303 registros en nuestro set de datos.

- Mean (Media): Representa la media aritmética de cada columna. Por ejemplo, si toma la columna “edad”, la edad promedio es aproximadamente 54.37, el promedio de la presión arterial en reposo (trestbps) es aproximadamente 131.62, etc.

- Std (Desviación estándar): Es la medida de dispersión de los valores. Es decir, nos indica cuánto varían los valores de la media. Por ejemplo, la desviación estándar de la edad es aproximadamente 9.08, lo que indica que la edad tiende a variar alrededor de 9 años respecto a la media.

- Min (Mínimo): Muestra el valor mínimo en cada columna. Por ejemplo, la edad mínima es 29 años.

- 25%, 50%, 75% (Percentiles): Estos valores muestran los percentiles correspondientes. Por ejemplo, el percentil 50% (o mediana) de la edad es 55, lo que significa que el 50% de los pacientes tienen una edad igual o inferior a 55 años.

- Max (Máximo): Muestra el valor máximo en cada columna. Por ejemplo, la presión arterial máxima en reposo es 200.

3.1.1) Detección y remoción de Outliers

Un valor atípico (outlier) es un valor extremadamente grande o pequeño en relación con el resto del conjunto de datos. Puede tratarse de un error cuando se realizó la introducción de

datos o de datos auténticos, que realmente son atípicos y pueden llegar a generarnos ruido en nuestra generalización.

En nuestro caso, vamos a filtrar características como age , resttecg en reposo, chol y thal, características numéricas que pueden llegar a tener valores atípicos.

```
data_fil = datasetI[['age','restecg','chol','thal']]
data_fil.head()
```

	age	restecg	chol	thal
0	63	0	233	1
1	37	1	250	2
2	41	0	204	2
3	56	1	236	2
4	57	1	354	2

Es difícil decir qué puntos son atípicos, por lo que vamos a tener que definir un umbral

```
threshold = 3
print(np.where(z > 3))
```

```
(array([ 28,  48,  85, 220, 246, 281], dtype=int64), array([2, 3, 2, 2, 2, 3], dtype=int64))
```

La primera matriz contiene la lista de números de fila y la segunda matriz los respectivos números de columna, lo que significa que z[28][2] tiene una puntuación Z superior a 3. Hay un total de 6 puntos de datos que son valores atípicos.

Y finalmente nos quedamos con nuestro DataSet final con el cual vamos a seguir trabajando;

```
datasetII.head()
```

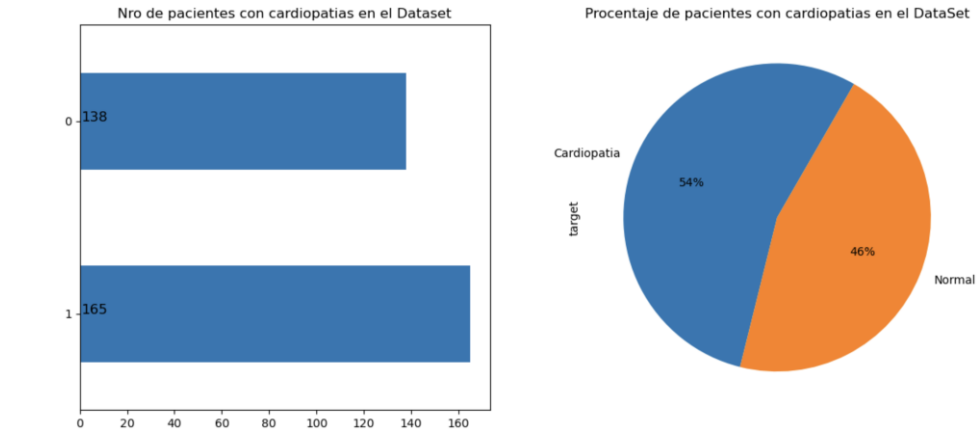
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

3.2) Análisis univariado

Ahora verificamos nuestra variable "target" en la población de nuestro dataset e intentaremos responder una de las preguntas que nos planteamos al principio:

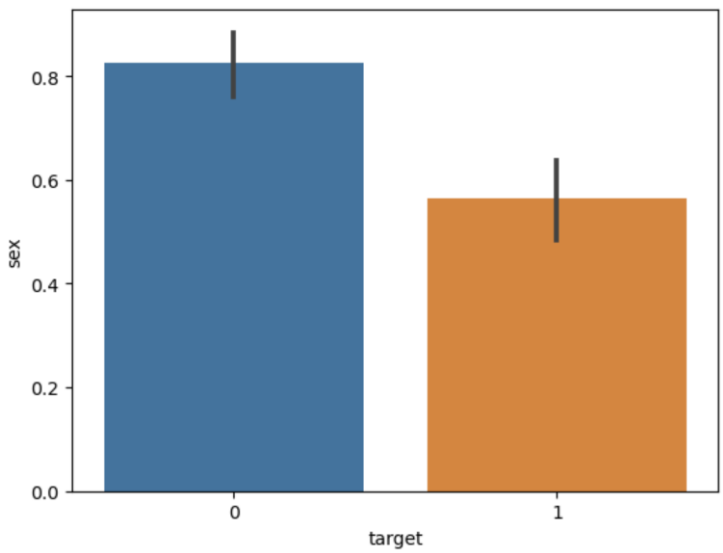
1) Cuál es el porcentaje de personas que tienen algún tipo de afección coronaria?

```
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, sharey=False, figsize=(14,6))

ax1 = datasetI["target"].value_counts().plot(kind="barh" ,ax =ax1)
for i,j in enumerate(datasetI["target"].value_counts().values):
    ax1.text(.5,i,j,fontsize=12)
ax1.set(title = 'Nro de pacientes con cardiopatias en el Dataset')

ax2 = datasetI['target'].value_counts().plot.pie( x="Cardiopatias" ,y ='Nro de pacientes',
                    autopct = "%1.0f%%",labels=["Cardiopatia","Normal"], startangle = 60,ax=ax2);
ax2.set(title = 'Procentaje de pacientes con cardiopatias en el DataSet')

plt.show()
```



Posteriormente vamos a analizar la variable "sex" y avanzamos con la segunda respuesta a las consultas iniciales que nos inicimos:

2) Los hombres son más propensos a tener afecciones cardiacas?



0 = mujeres; 1 = hombres

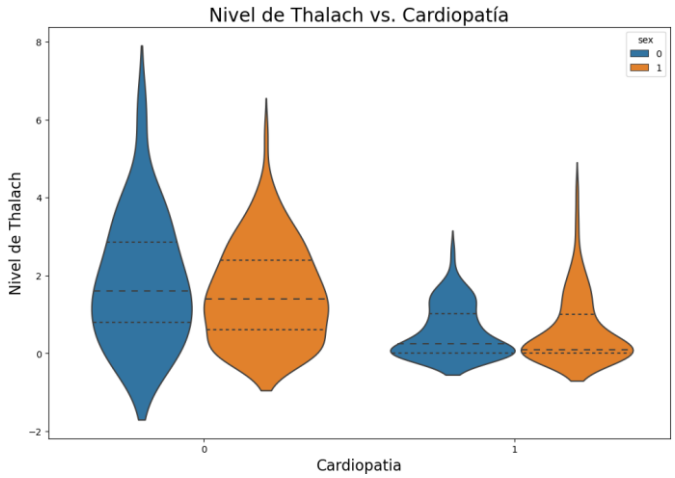
Tal como se puede observar esto no es correcto, ya que vemos claramente que hay un registro mayor de mujeres que de hombres con problemas cardiacos

3.3) Análisis bivariado

El análisis bivariado es el estudio de la relación entre dos variables en el conjunto de datos que estamos analizando. A diferencia del análisis univariado, que se centra en una sola variable a la vez, el análisis bivariado examina cómo dos variables diferentes se relacionan entre sí.

Este análisis es fundamental en la exploración de datos y en la comprensión de cómo las diferentes variables interactúan entre sí. Puede revelar patrones importantes, asociaciones y proporcionar información valiosa para la toma de decisiones.

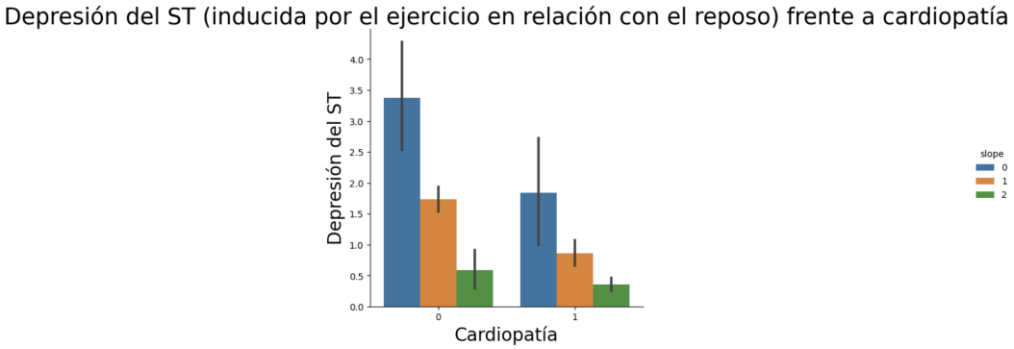
En nuestro caso hicimos una comparativa entre el nivel de Thalach y las cardiopatias resultantes mediante un Violinplot



La forma general y la distribución de los pacientes negativos y positivos difieren bastante. Los pacientes positivos muestran una mediana más baja para el nivel de depresión ST y, por lo tanto, una gran distribución de sus datos está entre 0 y 2, mientras que los pacientes negativos están entre 1 y 3. Y no hay muchas diferencias entre los resultados de hombres y mujeres y femeninos

Y si hacemos un nuevo análisis, pero en este compramos la depresión del ST [oldpeak] (stress inducido por el ejercicio en relación con el reposo) frete a las cardiopatías resultantes.

```
sns.catplot(x="target", y="oldpeak", hue="slope", kind="bar", data=datasetI);

plt.title('Depresión del ST (inducida por el ejercicio en relación con el reposo) frente a cardiopatía',size=25)
plt.xlabel('Cardiopatía',size=20)
plt.ylabel('Depresión del ST',size=20)
```



La depresión del segmento ST se produce cuando el ventrículo está en reposo y, por tanto, repolarizado. Si el trazo en el segmento ST es anormalmente bajo por debajo de la línea de base, esto puede conducir a esta Enfermedad Cardíaca. Esto apoya el gráfico realizado porque una baja Depresión del ST hace que las personas tengan un mayor riesgo de padecer una enfermedad cardiaca. Mientras que una depresión alta del ST se considera normal y saludable.

El matiz "pendiente", se refiere al segmento ST pico de ejercicio, con valores:
0: pendiente ascendente,
1: plano,
2: pendiente descendente

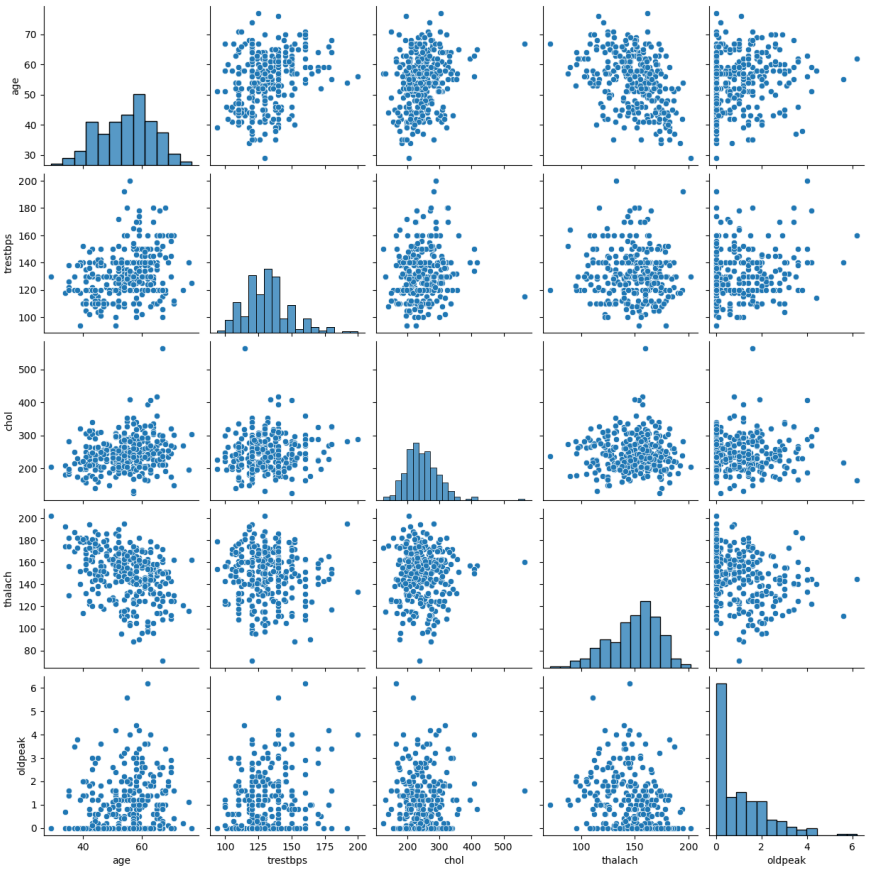
Tanto los pacientes con cardiopatía positiva como negativa muestran distribuciones iguales de las 3 categorías de pendiente.

3.4) Análisis multivariado

El análisis multivariado en implica el estudio simultáneo de tres o más variables para comprender las relaciones complejas entre ellas. A diferencia del análisis bivariado, que se centra en la relación entre dos variables, el análisis multivariado aborda la interacción entre múltiples variables en un conjunto de datos. Este enfoque es especialmente útil cuando se trabaja con conjuntos de datos que contienen múltiples características o variables.

Vamos a realizar un gráfico con variables continuas para profundizar en las relaciones. Es una buena forma de ver si existe una correlación positiva o negativa.

```
subData = datasetI[['age','trestbps','chol','thalach','oldpeak']]
sns.pairplot(subData)
```



4) Feature Engineer / Feature Selection

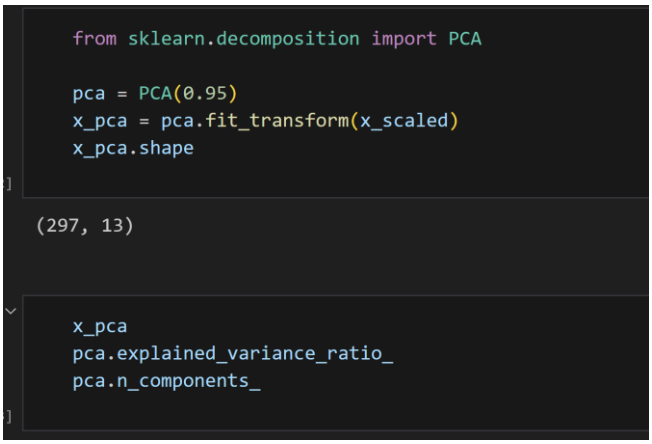
En nuestro caso decidimos utilizar StandarScaler para realizar el escalamiento estándar, dado que muchas veces las características de los datos pueden estar en diferentes escalas o rangos, lo que puede afectar el rendimiento de nuestros algoritmos. El escalamiento busca llevar todas las características a una escala común, generalmente con una media de 0 y una desviación estándar de 1 (lo que se conoce como estandarización).

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
x_scaled = scaler.fit_transform(datasetII)
x_scaled
```

```
array([[ 0.96065206,  0.66464094,  1.9726347 , ..., -0.71160071,
        -2.26944049,  0.91287093],
       [-1.89908395,  0.66464094,  1.00264711, ..., -0.71160071,
        -0.55436714,  0.91287093],
       [-1.45912456, -1.50457179,  0.03265951, ..., -0.71160071,
        -0.55436714,  0.91287093],
       ...,
       [ 1.51060129,  0.66464094, -0.93732808, ...,  1.25440311,
         1.16070621, -1.09544512],
       [ 0.30071298,  0.66464094, -0.93732808, ...,  0.2714012 ,
         1.16070621, -1.09544512],
       [ 0.30071298, -1.50457179,  0.03265951, ...,  0.2714012 ,
        -0.55436714, -1.09544512]])
```

Posteriormente realizamos la implementación de PCA, el cual es una técnica en el campo del análisis de datos y el aprendizaje automático. Su principal objetivo es reducir la dimensionalidad de los datos al tiempo que conserva la mayor cantidad de información posible

En nuestro primer análisis utilizamos un “PCA(0,95)”, lo que significa retener el 95% de la varianza total de los datos originales al realizar la reducción de dimensionalidad. Cuando aplicamos PCA con "PCA(0.95)", el algoritmo seleccionará automáticamente el número de componentes principales necesarios para retener al menos el 95% de la varianza de tus datos originales. El valor "0.95" representa el porcentaje mínimo de varianza que deseas conservar, y PCA calculará cuántos componentes principales son necesarios para cumplir con ese requisito



Posteriormente, cuando vayamos probando los modelos vamos a ver como impacta este proceso de Feature Engineer cuando evaluemos cada modelo que vamos a utilizar para seleccionar el que mejor se adapte al requerimiento que tenemos.

Feature Selection

Seguidamente nos centramos en hacer un feature selection para ver si es posible reducir aún más nuestro dataset pero a través de una técnica distinta.

Feature selection (selección de características) es un proceso en el cual elegimos un subconjunto relevante de características (variables) de nuestro dataset con el objetivo de reducir la dimensionalidad. La dimensionalidad se refiere al número de características en tus datos, y reducirla puede ser beneficioso debido a que:

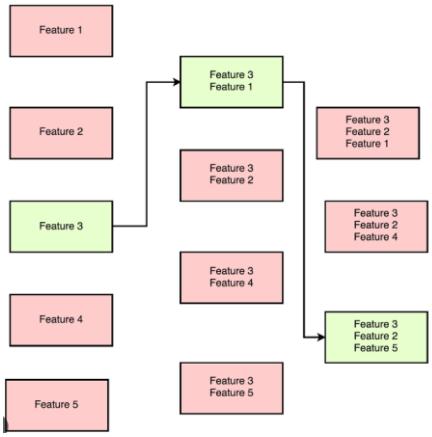
- **Mejora de la eficiencia computacional**: Reducir el número de características puede hacer que los algoritmos de machine learning sean más eficientes computacionalmente, ya que hay menos datos con los que trabajar.

- **Evitar la maldición de la dimensionalidad**: En conjuntos de datos de alta dimensionalidad, la cantidad de datos requeridos para estimar de manera fiable un modelo puede aumentar significativamente. La reducción de la dimensionalidad puede ayudar a evitar problemas asociados con la maldición de la dimensionalidad.

- **Mejora de la interpretabilidad:** Al tener menos características, es más fácil visualizar y entender las relaciones en los datos. La interpretación de un modelo también puede volverse más sencilla.

- **Prevención del sobreajuste (overfitting):** Al reducir la dimensionalidad, podemos disminuir la probabilidad de sobreajustar el modelo a características específicas que pueden no ser generalizables a nuevos datos.

En nuestro caso decimos avanzar con el método “Forward Selection” es un enfoque en el proceso de selección de características que se utiliza para construir un modelo al ir agregando características una a una, comenzando con la más informativa o relevante.



El proceso de "forward selection" generalmente sigue estos pasos:

1. Elija un nivel de significancia (por ejemplo, SL = 0.05 con un 95% de confianza).

2. Ajuste todos los modelos de regresión simple posibles considerando una característica a la vez. Los modelos totales 'n' son posibles. Seleccione la característica con el valor p más bajo.

3. Ajuste todos los modelos posibles con una característica adicional agregada a las características seleccionadas anteriormente.

4. Nuevamente, seleccione la función con un valor p mínimo. si p_v < alpha, vamos al Paso 3; de lo contrario, finalizamos el proceso.

```python
import statsmodels.api as sm
def forward_selection(data, target, significance_level=0.01):
    initial_features = data.columns.tolist()
    best_features = []
    while (len(initial_features)>0):
        remaining_features = list(set(initial_features)-set(best_features))
        new_pval = pd.Series(index=remaining_features)
        for new_column in remaining_features:
            model = sm.OLS(target, sm.add_constant(data[best_features+[new_column]])).fit()
            new_pval[new_column] = model.pvalues[new_column]
        min_p_value = new_pval.min()
        if(min_p_value<significance_level):
            best_features.append(new_pval.idxmin())
        else:
            break
    return best_features
```

Esta función anterior acepta datos, variable objetivo y nivel de significancia como argumentos y devuelve la lista final de características significativas basadas en valores p a través de la selección hacia adelante.

```python
forward_selection(X, y)
```

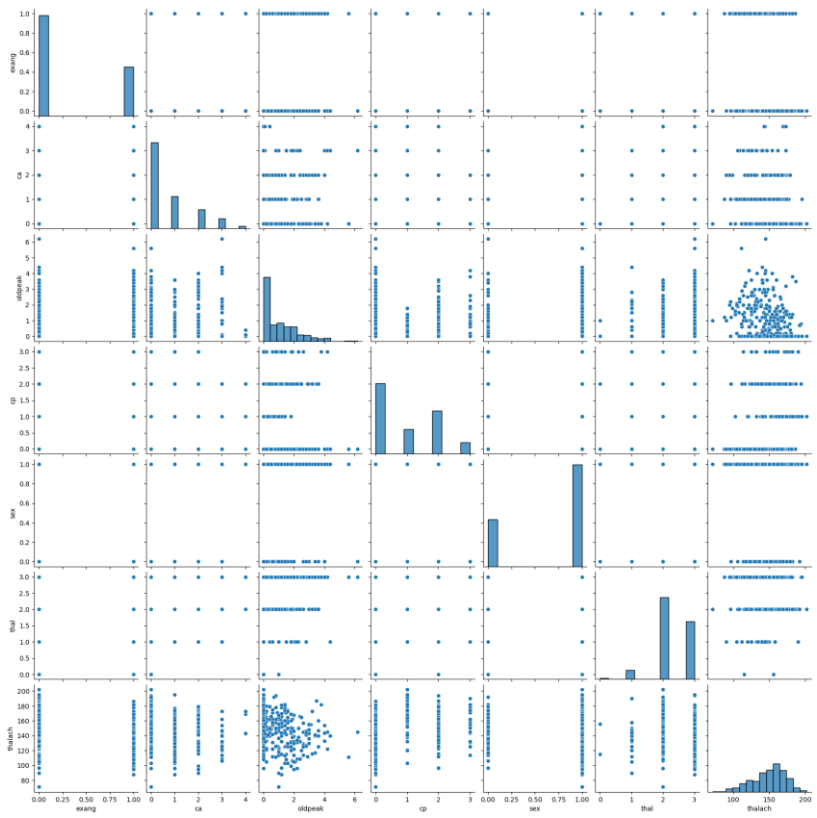
Python

```
['exang', 'ca', 'oldpeak', 'cp', 'sex', 'thal', 'thalach']
```

Finalmente, nuestro redujo nuestro dataset en el siguiente subjunto de variables:

['exang', 'ca', 'oldpeak', 'cp', 'sex', 'thal', 'thalach']

Hacemos nuevamente un pairplot para analizar las relaciones de este subconjunto:



5)Entrenamiento de modelos

Primero vamos a importar la función 'train_test_split del módulo 'model_selection' de la biblioteca scikit-learn (sklearn). Esta función la vamos a utilizar en el proceso de entrenamiento y evaluación de modelos de aprendizaje automático.

La idea es utilizar el conjunto de entrenamiento para ajustar el modelo a los datos y luego evaluar su rendimiento en el conjunto de prueba para obtener una estimación de su capacidad para generalizar.

```
from sklearn.model_selection import train_test_split

predictors = datasetI.drop("target",axis=1)
target = datasetI["target"]

X_train,X_test,Y_train,Y_test = train_test_split(predictors,target,test_size=0.20,random_state=0)
```

La función 'train_test_split' toma varios argumentos, siendo el más importante el conjunto de datos que se va a dividir:

- **'test_size' o 'train_size'**, especifican la proporción de datos que se asignará al conjunto de prueba o entrenamiento, respectivamente

```
X_train.shape
```

```
(242, 13)
```

```
X_test.shape
```

```
(61, 13)
```

```
Y_train.shape
```

```
(242,)
```

```
Y_test.shape
```

5.1) Generación de cálculo de métricas para validar el modelo que seleccionamos

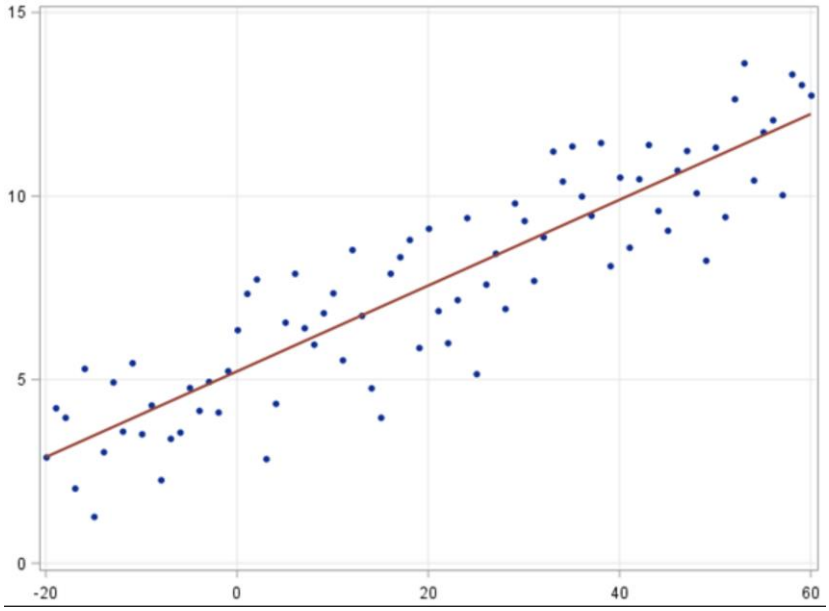
Para esto vamos a utilizar la función '**accuracy_score**' del módulo '**metrics**' para evaluar la precisión o el rendimiento del modelo en primera instancia.

```
from sklearn.metrics import accuracy_score
```

5.1.1) Regresión logística

La regresión logística es un modelo de regresión que se utiliza comúnmente para problemas de clasificación binaria, donde la variable de respuesta es categórica y tiene dos categorías distintas (por ejemplo, positivo/negativo, sí/no, 1/0).

Es ampliamente utilizada en problemas de clasificación binaria, como la predicción de la probabilidad de que un correo electrónico sea spam o no spam, la probabilidad de que un paciente tenga una enfermedad o no, etc. También puede extenderse a problemas de clasificación multiclase utilizando técnicas como la regresión logística multinomial.




```
from sklearn.linear_model import LogisticRegression

lregression = LogisticRegression()
lregression.fit(X_train,Y_train)
Y_pred_lregression = lregression.predict(X_test)
Y_pred_lregression.shape
```

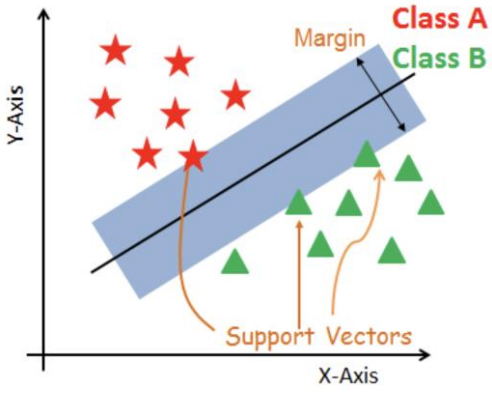
```
(61,)
```

```
score_lregression = round(accuracy_score(Y_pred_lregression,Y_test)*100,2)

print("La puntuación obtenida con la regresión logística es: "+str(score_lregression)+" %")
```

```
La puntuación obtenida con la regresión logística es: 85.25 %
```

5.1.2) SVM

Es un algoritmo de aprendizaje supervisado utilizado para problemas de clasificación y regresión. Fue desarrollado principalmente para problemas de clasificación, pero también puede ser aplicado a problemas de regresión. SVM es particularmente eficaz en espacios de alta dimensión y es ampliamente utilizado en tareas como reconocimiento de imágenes, procesamiento de texto y bioinformática




```
from sklearn import svm

sv = svm.SVC(kernel='linear')
sv.fit(X_train, Y_train)
Y_pred_svm = sv.predict(X_test)
Y_pred_svm.shape
```

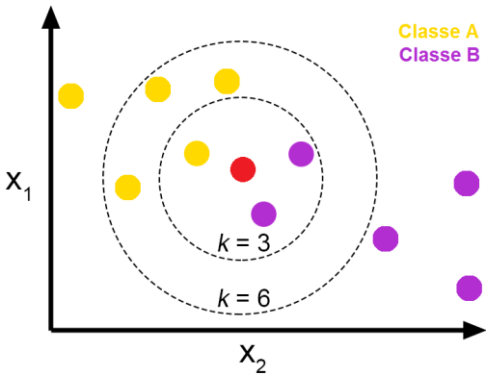
```
(61,)
```

```
score_svm = round(accuracy_score(Y_pred_svm,Y_test)*100,2)

print("La puntuación obtenida con SVM es: "+str(score_svm)+" %")
```

```
La puntuación obtenida con SVM es: 81.97 %
```

5.1.3) KNN

KNN (K-Nearest Neighbors o K-Vecinos más Cercanos) es un algoritmo de aprendizaje supervisado utilizado para problemas de clasificación y regresión. Es un tipo de algoritmo de "aprendizaje basado en instancias" o "aprendizaje perezoso" porque no construye un modelo explícito durante la fase de entrenamiento. En lugar de eso, almacena todos los datos de entrenamiento y clasifica las nuevas observaciones basándose en la similitud con las instancias existentes




```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train,Y_train)
Y_pred_knn=knn.predict(X_test)
Y_pred_knn.shape
```

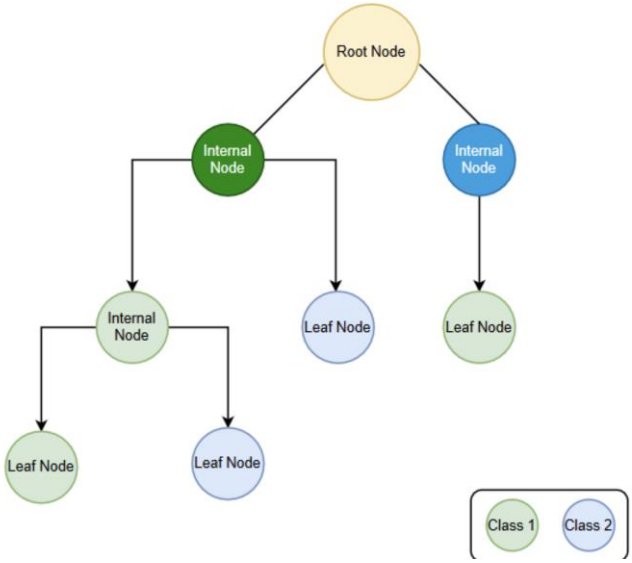
```
(61,)
```

```
score_knn = round(accuracy_score(Y_pred_knn,Y_test)*100,2)

print("La puntuación obtenida con KNN es: "+str(score_knn)+" %")
```

```
La puntuación obtenida con KNN es: 67.21 %
```

5.1.4) Decision Tree

Es un algoritmo de aprendizaje supervisado utilizado en problemas de clasificación y regresión. Se utiliza para tomar decisiones basadas en múltiples condiciones. Podemos pensar en un árbol de decisión como una estructura jerárquica en forma de árbol, donde cada nodo representa una pregunta o prueba sobre una característica, cada rama representa una posible respuesta a esa pregunta, y cada hoja del árbol representa una decisión o un valor de salida.



```
from sklearn.tree import DecisionTreeClassifier

max_accuracy = 0

for x in range(200):
    dtree = DecisionTreeClassifier(random_state=x)
    dtree.fit(X_train,Y_train)
    Y_pred_dtree = dtree.predict(X_test)
    current_accuracy = round(accuracy_score(Y_pred_dtree,Y_test)*100,2)
    if(current_accuracy>max_accuracy):
        max_accuracy = current_accuracy
        best_x = x

dtree = DecisionTreeClassifier(random_state=best_x)
dtree.fit(X_train,Y_train)
Y_pred_dtree = dtree.predict(X_test)
Y_pred_dtree.shape
```

```
(61,)
```

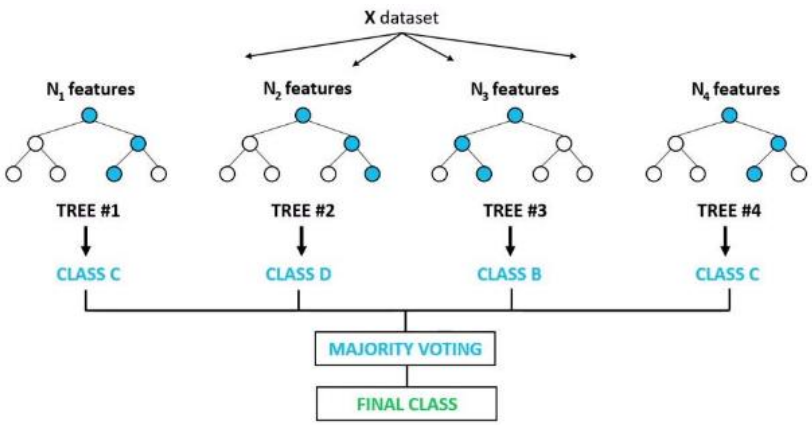
```
score_dtree = round(accuracy_score(Y_pred_dtree,Y_test)*100,2)

print("La puntuación obtenida con Decision Tree es: "+str(score_dtree)+" %")
```

```
La puntuación obtenida con Decision Tree es: 81.97 %
```

5.1.5) Random Forest

Random Forest (Bosque Aleatorio) es un algoritmo de aprendizaje supervisado que se utiliza tanto para problemas de clasificación como para regresión. Es una técnica de ensamblado que construye múltiples árboles de decisión durante la fase de entrenamiento y combina sus predicciones para obtener una predicción más robusta y precisa.

La idea central detrás del Random Forest es construir una "muestra" (forest en inglés) de árboles de decisión, donde cada árbol se entrena en una submuestra aleatoria del conjunto de datos de entrenamiento. Además, en cada nodo de decisión de cada árbol, se considera solo un subconjunto aleatorio de características para realizar la división, lo que introduce más variabilidad y diversidad en los árboles.



```python
from sklearn.ensemble import RandomForestClassifier

max_accuracy = 0

for x in range(200):   ## Hay que utilizar un rango mayor, lo probe con 2000 pero llevaba 10min y no terminaba
    rforest = RandomForestClassifier(random_state=x)
    rforest.fit(X_train,Y_train)
    Y_pred_rforest = rforest.predict(X_test)
    current_accuracy = round(accuracy_score(Y_pred_rforest,Y_test)*100,2)
    if(current_accuracy>max_accuracy):
        max_accuracy = current_accuracy
        best_x = x

rforest = RandomForestClassifier(random_state=best_x)
rforest.fit(X_train,Y_train)
Y_pred_rforest = rforest.predict(X_test)
Y_pred_rforest.shape
```

```
(61,)
```

```python
score_rforest = round(accuracy_score(Y_pred_rforest,Y_test)*100,2)

print("La puntuación obtenida con Random Forest es: "+str(score_rforest)+" %")
```

```
La puntuación obtenida con Random Forest es: 88.52 %
```

5.2) Evaluación de modelos

En este paso definiremos otras métricas de evaluación que utilizaremos para evaluar nuestros modelos previamente presentados. Las métricas de evaluación más importantes son:

- la sensibilidad,
- la especificidad,
- la precisión,
- la medida F1,
- la media geométrica y el coeficiente de correlación Mathew,
- la curva ROC AUC

5.2.1) Coeficiente de correlación de Mathew (MCC)

El coeficiente de correlación de Matthews (CCM), es un índice estadístico fiable que produce una puntuación alta sólo si la predicción obtuvo buenos resultados en las cuatro categorías de la matriz de confusión (verdaderos positivos, falsos negativos, verdaderos negativos y falsos positivos).

5.2.2) Perdida Logarítmica

La pérdida logarítmica mide el rendimiento de un modelo de clasificación en el que la predicción de entrada es un valor de probabilidad entre 0 y 1. El objetivo de nuestros modelos de aprendizaje automático es minimizar este valor. Un modelo perfecto tendría una pérdida logarítmica de 0. La pérdida logarítmica aumenta a medida que la probabilidad predicha diverge de la etiqueta real buscada. Así, predecir una probabilidad de 0,012 cuando la etiqueta de observación real es 1 sería malo y daría lugar a una pérdida logarítmica alta.

5.2.3) Medida F1

La medida F1 es la media ponderada de Precision y Recall. Por lo tanto, esta puntuación tiene en cuenta tanto los falsos positivos como los falsos negativos. Intuitivamente no es tan fácil de entender como la precisión, pero F1 suele ser más útil que la precisión, sobre todo si se tiene una distribución de clases muy desigual. La precisión funciona mejor si los falsos positivos y los falsos negativos tienen un coste similar. Si el coste de los falsos positivos y los falsos negativos es muy diferente, es mejor tener en cuenta tanto la precisión como la recuperación.

5.3) Comparación con otros modelos

```
data = {        'LREGESSION': Y_pred_lregression,
                'SVM': Y_pred_svm,
                'KNN': Y_pred_knn,
                'DTREE': Y_pred_dtree,
                'RFOREST': Y_pred_rforest,
                }

models = pd.DataFrame(data)

for column in models:
    CM=confusion_matrix(Y_test,models[column])

    TN = CM[0][0]
    FN = CM[1][0]
    TP = CM[1][1]
    FP = CM[0][1]
    specificity = TN/(TN+FP)
    loss_log = log_loss(Y_test, models[column])
    acc= accuracy_score(Y_test, models[column])
    roc=roc_auc_score(Y_test, models[column])
    prec = precision_score(Y_test, models[column])
    rec = recall_score(Y_test, models[column])
    f1 = f1_score(Y_test, models[column])

    mathew = matthews_corrcoef(Y_test, models[column])
    results =pd.DataFrame([[column,acc, prec,rec,specificity, f1,roc, loss_log,mathew]],
                columns = ['Model', 'Accuracy','Precision', 'Sensitivity','Specificity', 'F1 Score','ROC','Log_Loss','mathew_corrcoef'])
    model_results = model_results._append(results, ignore_index = True)

model_results
```

Antes de una comparación con nuestros modelos, vamos a mostrar como hicimos la aplicación de PCA(0,95) en cada modelo:

```
from sklearn.decomposition import PCA

pca = PCA(0.95)
x_pca = pca.fit_transform(x_scaled)
x_pca.shape
```

```
(297, 13)
```

```
x_pca
pca.explained_variance_ratio_
pca.n_components_
```

5.3.1) Regresión Logistica

```
lregression = LogisticRegression()
lregression.fit(X_train,Y_train)
Y_pred_lregression_PCA095 = lregression.predict(X_test)
Y_pred_lregression_PCA095.shape

score_lregression_PCA095 = round(accuracy_score(Y_pred_lregression_PCA095,Y_test)*100,2)

print("La puntuación obtenida con la regresión logística es: "+str(score_lregression_PCA095)+" %")
```

```
La puntuación obtenida con la regresión logística es: 98.33 %
```

5.3.2) SVM

```
sv = svm.SVC(kernel='linear')
sv.fit(X_train, Y_train)
Y_pred_svm_PCA095 = sv.predict(X_test)
Y_pred_svm_PCA095.shape

score_svm_PCA095 = round(accuracy_score(Y_pred_svm_PCA095,Y_test)*100,2)

print("La puntuación obtenida con SVM es: "+str(score_svm_PCA095)+" %")
```

```
La puntuación obtenida con SVM es: 98.33 %
```

5.3.3) KNN

```
8.4) KNN
```

```
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train,Y_train)
Y_pred_knn_PCA095=knn.predict(X_test)
Y_pred_knn_PCA095.shape

score_knn_PCA095 = round(accuracy_score(Y_pred_knn_PCA095,Y_test)*100,2)

print("La puntuación obtenida con KNN es: "+str(score_knn_PCA095)+" %")
```

```
La puntuación obtenida con KNN es: 96.67 %
```

5.3.4) Decision Tree

```
max_accuracy = 0

for x in range(200):
    dtree = DecisionTreeClassifier(random_state=x)
    dtree.fit(X_train,Y_train)
    Y_pred_dtree = dtree.predict(X_test)
    current_accuracy = round(accuracy_score(Y_pred_dtree,Y_test)*100,2)
    if(current_accuracy>max_accuracy):
        max_accuracy = current_accuracy
        best_x = x

dtree = DecisionTreeClassifier(random_state=best_x)
dtree.fit(X_train,Y_train)
Y_pred_dtree_PCA095 = dtree.predict(X_test)
Y_pred_dtree_PCA095.shape

score_dtree_PCA095 = round(accuracy_score(Y_pred_dtree_PCA095,Y_test)*100,2)

print("La puntuación obtenida con Decision Tree es: "+str(score_dtree_PCA095)+" %")
```

```
La puntuación obtenida con Decision Tree es: 86.67 %
```

5.3.5) Random Forest

```
max_accuracy = 0

for x in range(200):
    rforest = RandomForestClassifier(random_state=x)
    rforest.fit(X_train,Y_train)
    Y_pred_rforest = rforest.predict(X_test)
    current_accuracy = round(accuracy_score(Y_pred_rforest,Y_test)*100,2)
    if(current_accuracy>max_accuracy):
        max_accuracy = current_accuracy
        best_x = x

rforest = RandomForestClassifier(random_state=best_x)
rforest.fit(X_train,Y_train)
Y_pred_rforest_PCA095 = rforest.predict(X_test)
Y_pred_rforest_PCA095.shape

score_rforest_PCA095 = round(accuracy_score(Y_pred_rforest_PCA095,Y_test)*100,2)

print("La puntuación obtenida con Random Forest es: "+str(score_rforest_PCA095)+" %")
```

```
La puntuación obtenida con Random Forest es: 98.33 %
```

5.3.6) Comparación de modelos a través de distintas métricas

```
data = {        'LREGESSION': Y_pred_lregression,
                'LREGESSION_PCA095': Y_pred_lregression_PCA095,
                'LREGESSION_PCAN2': Y_pred_lregression_PCAN2,
                'SVM': Y_pred_svm,
                'SVM_PCA095': Y_pred_svm_PCA095,
                'SVM_PCAN2': Y_pred_svm_PCAN2,
                'KNN': Y_pred_knn,
                'KNN_PCA095': Y_pred_knn_PCA095,
                'KNN_PCAN2': Y_pred_knn_PCAN2,
                'DTREE': Y_pred_dtree,
                'DTREE_PCA095': Y_pred_dtree_PCA095,
                'DTREE_PCAN2': Y_pred_dtree_PCAN2,
                'RFOREST': Y_pred_rforest,
                'RFOREST_PCA095': Y_pred_rforest_PCA095,
                'RFOREST_PCAN2': Y_pred_rforest_PCAN2,
                }

models = pd.DataFrame(data)

for column in models:
    CM=confusion_matrix(Y_test,models[column])

    TN = CM[0][0]
    FN = CM[1][0]
    TP = CM[1][1]
    FP = CM[0][1]
    specificity = TN/(TN+FP)
    loss_log = log_loss(Y_test, models[column])
    acc= accuracy_score(Y_test, models[column])
    roc=roc_auc_score(Y_test, models[column])
    prec = precision_score(Y_test, models[column])
    rec = recall_score(Y_test, models[column])
    f1 = f1_score(Y_test, models[column])

    mathew = matthews_corrcoef(Y_test, models[column])
    results =pd.DataFrame([[column,acc, prec,rec,specificity, f1,roc, loss_log,mathew]],
              columns = ['Model', 'Accuracy','Precision', 'Sensitivity','Specificity', 'F1 Score','ROC','Log_Loss','mathew_corrcoef'])
    model_results = model_results._append(results, ignore_index = True)

model_results
```

	Model	Accuracy	Precision	Sensitivity	Specificity	F1 Score	ROC	Log_Loss	mathew_corrcoef
0	Random Forest	0.933333	0.941176	0.941176	0.923077	0.941176	0.932127	2.402910	0.864253
1	LREGESSION	0.866667	0.882353	0.882353	0.846154	0.882353	0.864253	4.805820	0.728507
2	SVM	0.883333	0.864865	0.941176	0.807692	0.901408	0.874434	4.205093	0.763250
3	KNN	0.750000	0.806452	0.735294	0.769231	0.769231	0.752262	9.010913	0.500298
4	DTREE	0.833333	0.852941	0.852941	0.807692	0.852941	0.830317	6.007276	0.660633
5	RFOREST	0.933333	0.941176	0.941176	0.923077	0.941176	0.932127	2.402910	0.864253
6	LREGESSION	0.866667	0.882353	0.882353	0.846154	0.882353	0.864253	4.805820	0.728507
7	LREGESSION_PCA095	0.983333	1.000000	0.970588	1.000000	0.985075	0.985294	0.600728	0.966768
8	LREGESSION_PCAN2	0.933333	0.941176	0.941176	0.923077	0.941176	0.932127	2.402910	0.864253
9	SVM	0.883333	0.864865	0.941176	0.807692	0.901408	0.874434	4.205093	0.763250
10	SVM_PCA095	0.983333	1.000000	0.970588	1.000000	0.985075	0.985294	0.600728	0.966768
11	SVM_PCAN2	0.950000	0.969697	0.941176	0.961538	0.955224	0.951357	1.802183	0.899162
12	KNN	0.750000	0.806452	0.735294	0.769231	0.769231	0.752262	9.010913	0.500298
13	KNN_PCA095	0.966667	0.970588	0.970588	0.961538	0.970588	0.966063	1.201455	0.932127
14	KNN_PCAN2	0.933333	0.941176	0.941176	0.923077	0.941176	0.932127	2.402910	0.864253
15	DTREE	0.850000	0.903226	0.823529	0.884615	0.861538	0.854072	5.406548	0.702212
16	DTREE_PCA095	0.866667	0.882353	0.882353	0.846154	0.882353	0.864253	4.805820	0.728507
17	DTREE_PCAN2	0.850000	0.878788	0.852941	0.846154	0.865672	0.849548	5.406548	0.696343
18	RFOREST	0.883333	0.909091	0.882353	0.884615	0.895522	0.883484	4.205093	0.763950
19	RFOREST_PCA095	0.983333	1.000000	0.970588	1.000000	0.985075	0.985294	0.600728	0.966768
20	RFOREST_PCAN2	0.900000	0.937500	0.882353	0.923077	0.909091	0.902715	3.604365	0.800018

Como conclusión parcial, nuestros mejores algoritmos con mejores métricas se dan después de la implementación de PCA(0,95) :

- **Random Forest con PCA(0,95)** con una precisión de 1, una sensibilidad de 0,9705, especificidad de 1, una puntuación F1 de 0,985 y una perdida algortimica de 0,6
- **Regresion Logistica con PCA(0,95)** con metricas similares RandomForest_PCA095
- **SVM con PCA(0,95)** con metricas similares a RandomForest_PCA095 y Regresion_Logistic_PCA095

No obstante, esto es parcial, ya que después de aplicar Cross Validation, vamos a entender que modelo puede llegar a ser nuestra mejor opción.

6) Cross-Validation

La validación cruzada (cross-validation) es una técnica utilizada en machine learning para evaluar el rendimiento de un modelo estadístico y para minimizar el riesgo de sobreajuste. La idea principal es dividir el conjunto de datos en múltiples subconjuntos, entrenar y evaluar el modelo en varias particiones, y luego calcular medidas de rendimiento agregadas.

En este punto construiremos diferentes modelos de referencia y realizaremos una validación cruzada de 10 folds para filtrar los modelos de referencia de mayor rendimiento. Seleccionando los 5 modelos que estamos utilizando

- Random Forest
- Regresión Logistica
- SVM
- KNN
- Desicion Tree

```
from sklearn import model_selection
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score


def GetBasedModel():
    basedModels = []
    basedModels.append(('RANDOM_FOREST_CROSSV'    , RandomForestClassifier(criterion='entropy',n_estimators=200)))
    basedModels.append(('LOGISTIC_REGRESSION_DTREE_CROSSV'    , LogisticRegression(penalty='l2')))
    basedModels.append(('SVM Linear_DTREE_CROSSV'   , SVC(kernel='linear',gamma='auto',probability=True)))
    basedModels.append(('SVM RBF_DTREE_CROSSV'   , SVC(kernel='rbf',gamma='auto',probability=True)))
    basedModels.append(('KNN2_DTREE_CROSSV'   , KNeighborsClassifier(2)))
    basedModels.append(('KNN5_DTREE_CROSSV'   , KNeighborsClassifier(5)))
    basedModels.append(('DTREE_CROSSV'    , DecisionTreeClassifier(criterion='entropy', splitter='best')))
    return basedModels
```

Luego vamos a crear una función para evaluar varios modelos utilizando validación cruzada de 10 divisiones (Folds) y calculando la precisión de cada modelo:

```python
def BasedLine2(X_train, Y_train, models):
    num_folds = 10
    scoring = 'accuracy'
    seed = 7
    results = []
    names = []
    for name, model in models:
        kfold = model_selection.KFold(n_splits=10, shuffle=True, random_state=seed)
        cv_results = model_selection.cross_val_score(model, X_train, Y_train, cv=kfold, scoring=scoring)
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)


    return results,msg


models = GetBasedModel()
names,results = BasedLine2(X_train, Y_train, models)
```

```
RANDOM_FOREST_CROSSV: 0.898732 (0.057661)
LOGISTIC_REGRESSION_DTREE_CROSSV: 0.902899 (0.057300)
SVM Linear_DTREE_CROSSV: 0.898732 (0.057661)
SVM RBF_DTREE_CROSSV: 0.885870 (0.060497)
KNN2_DTREE_CROSSV: 0.835688 (0.059931)
KNN5_DTREE_CROSSV: 0.881522 (0.068310)
DTREE_CROSSV: 0.873551 (0.066999)
```

Si comparamos los resultados obtenidos de nuestros modelos con PCA (0,95);

- **Random Forest con PCA(0,95)** con una precisión de 1, una sensibilidad de 0,9705, especificidad de 1, una puntuación F1 de 0,985 y una perdida algortimica de 0,6
- **Regresion Logistica con PCA(0,95)** con metricas similares RandomForest_PCA095
- **SVM con PCA(0,95)** con metricas similares a RandomForest_PCA095 y Regresion_Logistic_PCA095

Con los resultados obtenidos a través de nuestro Cross Validation;

- RANDOM_FOREST_CROSSV: 0.898732 (0.057661)
- LOGISTIC_REGRESSION_DTREE_CROSSV: 0.902899 (0.057300)
- SVM Linear_DTREE_CROSSV: 0.898732 (0.057661)
- SVM RBF_DTREE_CROSSV: 0.885870 (0.060497)
- KNN2_DTREE_CROSSV: 0.835688 (0.059931)
- KNN5_DTREE_CROSSV: 0.881522 (0.068310)
- DTREE_CROSSV: 0.873551 (0.066999)

Los resultados de PCA hacen referencia al rendimiento del modelo después de aplicar una reducción de dimensionalidad a los datos originales, esto es útil para acelerar el entrenamiento del modelo y reducir la complejidad, pero no necesariamente proporciona una evaluación precisa del rendimiento del modelo con datos reales.

En cambio, con Cross Validation tenemos una evaluación más sólida del rendimiento del modelo con datos reales, ya que se basa en la utilización de promedios a lo largo de múltiples iteraciones con datos segmentados en entrenamiento y prueba. Esto es muy útil ya que nos certeza de cómo se desempeñará el modelo en la práctica.

Por lo tanto, para nuestro análisis final, vamos a tener desestimar los resultados obtenidos con PCA (0,95) y observar y analizar los obtenidos con nuestro PCA n_components = 2, de donde podemos llegar a concluir que nuestro set de datos puede tener resultados muy positivos con una reducción de dimensionalidad bastante importante.

7) Conclusión final

	Model	Accuracy	Precision	Sensitivity	Specificity	F1 Score	ROC	Log_Loss	mathew_corrcoef
0	Random Forest	0.933333	0.941176	0.941176	0.923077	0.941176	0.932127	2.402910	0.864253
1	LREGESSION	0.866667	0.882353	0.882353	0.846154	0.882353	0.864253	4.805820	0.728507
2	SVM	0.883333	0.864865	0.941176	0.807692	0.901408	0.874434	4.205093	0.763250
3	KNN	0.750000	0.806452	0.735294	0.769231	0.769231	0.752262	9.010913	0.500298
4	DTREE	0.833333	0.852941	0.852941	0.807692	0.852941	0.830317	6.007276	0.660633
5	RFOREST	0.933333	0.941176	0.941176	0.923077	0.941176	0.932127	2.402910	0.864253
6	LREGESSION	0.866667	0.882353	0.882353	0.846154	0.882353	0.864253	4.805820	0.728507
7	LREGESSION_PCA095	0.983333	1.000000	0.970588	1.000000	0.985075	0.985294	0.600728	0.966768
8	LREGESSION_PCAN2	0.933333	0.941176	0.941176	0.923077	0.941176	0.932127	2.402910	0.864253
9	SVM	0.883333	0.864865	0.941176	0.807692	0.901408	0.874434	4.205093	0.763250
10	SVM_PCA095	0.983333	1.000000	0.970588	1.000000	0.985075	0.985294	0.600728	0.966768
11	SVM_PCAN2	0.950000	0.969697	0.941176	0.961538	0.955224	0.951357	1.802183	0.899162
12	KNN	0.750000	0.806452	0.735294	0.769231	0.769231	0.752262	9.010913	0.500298
13	KNN_PCA095	0.966667	0.970588	0.970588	0.961538	0.970588	0.966063	1.201455	0.932127
14	KNN_PCAN2	0.933333	0.941176	0.941176	0.923077	0.941176	0.932127	2.402910	0.864253
15	DTREE	0.850000	0.903226	0.823529	0.884615	0.861538	0.854072	5.406548	0.702212
16	DTREE_PCA095	0.866667	0.882353	0.882353	0.846154	0.882353	0.864253	4.805820	0.728507
17	DTREE_PCAN2	0.850000	0.878788	0.852941	0.846154	0.865672	0.849548	5.406548	0.696343
18	RFOREST	0.883333	0.909091	0.882353	0.884615	0.895522	0.883484	4.205093	0.763950
19	RFOREST_PCA095	0.983333	1.000000	0.970588	1.000000	0.985075	0.985294	0.600728	0.966768
20	RFOREST_PCAN2	0.900000	0.937500	0.882353	0.923077	0.909091	0.902715	3.604365	0.800018

```
RANDOM_FOREST_CROSSV: 0.898732 (0.057661)
LOGISTIC_REGRESSION_DTREE_CROSSV: 0.902899 (0.057300)
SVM Linear_DTREE_CROSSV: 0.898732 (0.057661)
SVM RBF_DTREE_CROSSV: 0.885870 (0.060497)
KNN2_DTREE_CROSSV: 0.835688 (0.059931)
KNN5_DTREE_CROSSV: 0.881522 (0.068310)
DTREE_CROSSV: 0.873551 (0.066999)
```

Si hacemos una comparativa entre los resultados obtenidos por CV (Cross-Validation) y nuestro evaluación de todos los modelos, observamos que el algoritmo Random Forest claramente ofrece la mayor precisión, en principio cuando tomamos nuestro Random Forest con PCA(n=2) y nuestro Random Forest sin reducción de dimensionalidad.

Y de las 13 características que examinamos en una primera instancia, nuestras 4 características más significativas que nos ayudan a clasificar entre un diagnóstico positivo y negativo con mayor precisión son:

- el tipo de dolor torácico (cp),
- la frecuencia cardiaca máxima alcanzada (thalach),
- el número de vasos principales (ca)
- y la depresión del ST inducida por el ejercicio en relación con el reposo (oldpeak)

Esto también lo comprobamos cuando aplicamos nuestro “Forward Selection” que nos agregó algunas features más, por lo cual es lógico seguir haciendo foco en dichas características para generar un algoritmo que pueda preveer afecciones cardiacas con tiempo y así generar algún tratamiento que ayude a los pacientes detectados con este tiempo de afección.