Funciones:

- Funciones gráficas.

In [1]:
```
# librerías necesarias para realizar este estudio.
#librerías generales
import pandas as pd
import numpy as np
import math

#librerías escalar,normalizar,estimadores, etc...
# Creamos el PCA.
import sklearn
from sklearn.compose import ColumnTransformer
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler,MinMaxScaler,RobustScaler
from sklearn.compose import make_column_selector
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression,LogisticRegressionCV,RidgeClassifier
from sklearn.model_selection import RepeatedStratifiedKFold,validation_curve
from sklearn.model_selection import GridSearchCV
from pycaret.classification import *
from sklearn.metrics import f1_score, confusion_matrix,precision_score, auc,roc_curve ,a
from sklearn.metrics import classification_report
from sklearn import metrics
from sklearn.model_selection import RepeatedKFold,learning_curve,cross_val_score
from sklearn.metrics import matthews_corrcoef
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler
from imblearn.combine import SMOTETomek
from imblearn.ensemble import BalancedBaggingClassifier

from collections import Counter



#librerías EDA
from scipy.stats import shapiro,pearsonr,normaltest,anderson,boxcox
from scipy import stats

#librerías escritura
from colorama import init, Fore, Back, Style

#librerías gráficos
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from plotly import colors
import plotly.graph_objects as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots
from yellowbrick.classifier import ROCAUC
from PIL import Image
import dash
from dash import dcc
from dash import html
from dash.dependencies import Input, Output
import signal
import sys
import os
```

```
# Configuration
from sklearn import set_config
%matplotlib inline
set_config(display='diagram')

import warnings
warnings.filterwarnings('ignore')


# Carga del Dataset
#Variables genéricas:
path1=r'C:\Users\Nitropc\IT Academy\Data Science\Proyecto Data Science\Fuente de Datos\h
#path1=r'/kaggle/input/heart-attack-analysis-prediction-dataset/heart.csv'
path2=r'C:\Users\Nitropc\IT Academy\Data Science\Proyecto Data Science\Fuente de Datos\o
#Funciones
# Intento de Mecanizar acciones/herramientas para el analisis de cualquier heartSet
# Guardar información de los pasos EDA


#Cuerpo
#heart = pd.read_csv(path,sep=',',encoding='ISO-8859-1')
heart = pd.read_csv(path1,sep=',',encoding='latin-1')
heart_columns=['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh',
       'exng', 'oldpeak', 'slp', 'caa', 'thall', 'target']
heart.columns = heart_columns

heart.columns
o2sat = pd.read_csv(path2,sep=',',encoding='latin-1')
o2sat.columns=['Sat_level']

#Añadimos la columna del dataset o2sat a heart
heart["Sat_level"] = o2sat["Sat_level"]
#Reordenar atributos
heart = heart[['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh',
       'exng', 'oldpeak', 'slp', 'caa', 'thall','Sat_level', 'target']]
#Definir Colores
color_palette = colors.qualitative.Plotly
cols = px.colors.DEFAULT_PLOTLY_COLORS
colors= color_palette
```

In [2]:

```
def dibujartabla(title,df):
    #print(Style.BRIGHT +title+Style.RESET_ALL)
    max_widths = [max([len(str(value)) for value in df[column]])+3 for column in df.colu
    fig = go.Figure(data=[go.Table(
      #columnorder = [1,2],
      columnwidth = max_widths,
      header = dict(
                values = list(df.columns),
                 # line_color=colors[4],
                 #fill_color='royalblue',
                align='left',
                font=dict(#color=colors[6],
                          size=14),
                 height=40
                ),
      cells=dict(
                values=df.transpose().values.tolist(),
                #line_color=colors[4],
                #fill=dict(color=colors[0]),
                align='left',
                font_size=12,
                height=30)
                )
                ])
    fig.update_layout(title =  title,
```

```
                      title_font_size = 20, title_x = 0.5)
    fig.show()
```

In [3]:
```
def crearhistograma(df,columnas):


    fig, axes = plt.subplots(3, 5, figsize=(20, 10))

    for idx, (col, ax) in enumerate(zip(columnas, axes.flatten())):
        sns.histplot(df[columnas].iloc[: , idx],  ax = ax, kde_kws=dict(linewidth=3))
        plt.subplots_adjust(wspace=.5, hspace=.5)
        ax.set_xlabel(col,fontsize=16,fontweight ='bold')
        ax.set_ylabel('Frecuencia', fontsize=16,fontweight ='bold')
    else:
        [ax.set_visible(False) for ax in axes.flatten()[idx+1:]]
    return
```

In [4]:
```
def estudio_atributos (data, valor, leyen_p, leyen_s, titul_p, titul_s, ejeX1, ejeY1 , e
    fig = make_subplots(rows=1, cols=2
                    ,subplot_titles=['Key1','Key2']
                    ,specs=[[{'type': 'domain'},{'type': 'xy'}]]
                    ,vertical_spacing=0.001
                    ,print_grid=False
                   )

    fig.add_trace(go.Pie(labels=data[valor].value_counts().index,
                    values=data[valor].value_counts(),
                    legendgroup="group",  # this can be any string, not just "group"
                    legendgrouptitle_text=leyen_p,
                    title=" ",hole=.3

                    ),row=1, col=1)

    fig.update_traces( textfont_size=15,  marker=dict(colors=colors,line=dict(color='#00

    fig.add_trace(go.Histogram(histfunc="count",
                            y=data.query('target_descrip == "Less chance of Heart Attack"
                            x=data.query('target_descrip == "Less chance of Heart Attack"
                            legendgroup="group2",
                            legendgrouptitle_text=leyen_s,
                            name="Less chance of Heart Attack",
                            marker={'color': colors[4]}),row=1, col=2
             )
    fig.add_trace(go.Histogram(histfunc="count",
                            y=data.query('target_descrip == "More chance of Heart Attack"
                            x=data.query('target_descrip == "More chance of Heart Attack"
                            legendgroup="group2",
                            legendgrouptitle_text=leyen_s,
                            name="More chance of Heart Attack",
                            marker={'color': colors[5]}),row=1, col=2
             )
    #fig.layout.annotations[0].update(text="Porcentaje por Género.").update(y=1.2)
    #fig.layout.annotations[1].update(text="Probabilidad de Infarto en función del Géner

    fig.update_annotations(selector={"text":"Key1"},text=titul_p, x=ejeX1, y=ejeY1)
    fig.update_annotations(selector={"text":"Key2"},text=titul_s, x=ejeX2, y=ejeY2)



    fig.update_layout(height=500, width=950,)
    fig.show()

    return
```

In [5]:
```
def estudio_grupo(data, valor, valor2, leyen_p, leyen_s, titul_p, titul_s, cate_array, e

    fig = make_subplots(rows=2, cols=1,subplot_titles=['Key1','Key2']
                    #,subplot_titles=("Probabilidad de infarto según la Edad.", "Probabi
                    #,specs=[[{'type': 'xy'},{'type': 'xy'}]]
                    #,vertical_spacing=0.001

                   )

    fig.add_trace(go.Bar(x=data.query('target_descrip == "Less chance of Heart Attack"')
                     y=data.query('target_descrip == "Less chance of Heart Attack"')['co
                     name="Less chance of Heart Attack",offsetgroup=0,
                     marker={'color': colors[4]}),row=1, col=1)
    fig.add_trace(go.Bar(x=data.query('target_descrip == "More chance of Heart Attack"')
                     y=data.query('target_descrip == "More chance of Heart Attack"')['co
                     name="More chance of Heart Attack",offsetgroup=1,
                     marker={'color': colors[5]}),row=1, col=1)

    fig.update_layout(legend_title_text=leyen_s)


    fig.add_trace(go.Histogram(histfunc="sum",
                           y=data.query('target_descrip == "Less chance of Heart Attack"
                           x=data.query('target_descrip == "Less chance of Heart Attack"
                           #y=df1[df1['target_descrip']== 'target_descrip'].sort_values(
                           #x=df1.query('target_descrip == "Less chance of Heart Attack"
                           #legendgroup="group2",
                           #legendgrouptitle_text="Target",
                           #name="Less chance of Heart Attack",
                           showlegend=False,
                           marker={'color': colors[4]}
                           ),row=2, col=1)
    fig.add_trace(go.Histogram(histfunc="sum",
                           y=data.query('target_descrip == "More chance of Heart Attack"
                           x=data.query('target_descrip == "More chance of Heart Attack"
                           #legendgroup="group2",
                           #legendgrouptitle_text="Target",
                           #name="More chance of Heart Attack",
                           #category_orders={'grupo_edad':['(20, 30]','(30, 40]','(40, 5
                           showlegend=False,
                           marker={'color': colors[5]}
                           ),row=2, col=1)

    fig.update_xaxes(categoryorder='array', categoryarray= cate_array)

    # select based on text on we're ok
    fig.update_annotations(selector={"text":"Key1"}, text=titul_p, x=ejeX1,y=ejeY1)
    fig.update_annotations(selector={"text":"Key2"}, text=titul_s, x=ejeX2,y=ejeY2)

    #
    fig.update_layout(height=600, width=900,legend_traceorder="reversed")
    fig.show()
    return
```

In [6]:
```
def crearquantile(df,columnas):


    fig, axes = plt.subplots(3, 5, figsize=(20, 10))

    for idx, (col, ax) in enumerate(zip(columnas, axes.flatten())):
        stats.probplot(df[columnas].iloc[: , idx],dist='norm',plot=ax)
        plt.subplots_adjust(wspace=.5, hspace=.5)
        ax.set_xlabel(col,fontsize=16,fontweight ='bold')
        ax.set_ylabel('Ordered Values', fontsize=16,fontweight ='bold')
```

```
    else:
        [ax.set_visible(False) for ax in axes.flatten()[idx+1:]]
```

In [7]:
```
#Bucle para Shapiro-Wilk a todas las columnas del dataset
def crearShapiro(df,columnas):
    #dataset para la prueba de Shapiro
    datoShapiro=[]
    alfa=0.05
    #print(Style.BRIGHT +'Resultado del Test de Hipótesis:'+Style.RESET_ALL)
    #print(Style.BRIGHT + Fore.GREEN+'Shapiro-Wilk: \n'+Style.RESET_ALL)

    for i in df[columnas]:
        stat, p = shapiro(df[i])
        #print(Style.BRIGHT + Fore.BLACK+f'{i}:'+Style.RESET_ALL)
        #print(Fore.RESET+'t-statistic = %.3f\np-value = %.6f' % (stat, p))
        if p > alfa:
            #print(Fore.BLUE+f'No podemos rechazar Ho con un nivel de significancia del
            sha_datos=[i,round(stat,8),round(p,8),'Probably Gaussian']
            datoShapiro.append(sha_datos)
        else:
            #print(Fore.RED+f'Podemos rechazar Ho con un nivel de significancia del {alf
            sha_datos=[i,round(stat,8),round(p,8),'Not Probably Gaussian']
            datoShapiro.append(sha_datos)

    TablaShapiro=pd.DataFrame(datoShapiro,
         columns=['Atributo','Stat','p-value','Resultado'])
    #TablaShapiro = TablaShapiro.style.set_properties(**{'text-align': 'left'})
    #display(TablaShapiro)

    #print(Fore.RESET+'---------- \n')
    return(TablaShapiro)
```

In [8]:
```
def Setconfusion_matrix(Vtitulo,ytest,ypred):
    print('\n\nMatriz de Confusión:\n')
    y_cambio = ytest
    fig, axes = plt.subplots(3, 2, figsize=(20, 20))
    fontsize=15
    label=['Less chance H.A', 'More chance H.A']

   #for idx, (col, ax) in enumerate(zip(ypred, axes.flatten())):

    for idx, ax in enumerate(axes.flat):
        y_cambio = ytest
        label = None
        cf_matrix = confusion_matrix(y_cambio, ypred[idx], labels=label)
        ax.xaxis.set_label_position('top')
        sns.heatmap(cf_matrix, annot=True, ax = ax, annot_kws = {'size':25},cmap=colors)

        if Vtitulo[idx] == 'None':
            ax.set_title('Sin Balanceo de clases en el módelo', fontsize=20)
        elif Vtitulo[idx] == 'Balanced':
            ax.set_title('Con Balanceo de clases en el módelo', fontsize=20)
        else:
            ax.set_title(f'Balanceo de clases con {Vtitulo[idx]}', fontsize=20)

        ax.set_ylabel('Valor Actual', fontsize=fontsize)
        ax.set_xlabel('Valor predicho', fontsize=fontsize)

    else:
        [ax.set_visible(False) for ax in axes.flatten()[idx+1:]]

    #plt.text(1.5,257.44,'Predicción', fontsize=fontsize)
    fig.tight_layout()
    plt.show()
```

In [9]:
```
# visualicemos los errores de este árbol en una matriz de confusión
def grafica_matrix (ytest,ypred):
    cf_matrix = confusion_matrix(ytest, ypred)
    print('\n\nMatriz de Confusión:\n')
    #print(cf_matrix)
    fig, ax = plt.subplots(figsize=(8, 4))
    ax.xaxis.set_label_position('top')
    sns.heatmap(cf_matrix, annot=True,cmap=colors,fmt='d');
    plt.tight_layout()
    plt.title('Matriz de confusion', y=1.1)
    plt.ylabel('Valor Actual')
    plt.xlabel('Valor predicho')
    plt.Text(1.5,257.44,'Predicción')
    plt.show()
```

- Funciones de cálculo.

In [10]:
```
def grafica_ROC_curve(model, xtrain, ytrain, xtest, ytest):

    # Creating visualization with the readable labels
    visualizer = ROCAUC(model, is_fitted=True)# ,encoder={1: '1',2: '2', 3: '3'})

    # Fitting to the training data first then scoring with the test data
    visualizer.fit(xtrain, ytrain)
    visualizer.score(xtest, ytest)
    visualizer.show()

    return visualizer
```

In [11]:
```
def EstaDescrip(num):
    #Añadimos describe
    df = num.describe().T
    #Añadimos la mediana
    df['median'] = num.median()
    #Reordenamos para que la mediana esté al lado de la media
    df = df.iloc[:,[0,1,8,2,3,4,5,6,7]]

    return(df)
```

In [12]:
```
def probar_balanceadores(balanceo, Xtrain, X_test, ytrain, y_test):
    resultados = []


    target_names = ['Less chance H.A', 'More chance H.A']
    i=0
    for nombre, balancedor in balanceo.items():
        X_train=Xtrain
        y_train=ytrain
        #print(balancedor["nomenclatura"])
        if balancedor["nomenclatura"] == "cbal":
            class_weight='balanced'
        elif balancedor["nomenclatura"]=="sbal":
             class_weight=None
        elif balancedor["nomenclatura"]=="bagging":
             class_weight=None
             balanceadores = balancedor["balanceador"]
             balanceadores.fit(X_train, y_train)
        else:
             class_weight=None
             balanceadores = balancedor["balanceador"]
             X_train, y_train = balanceadores.fit_resample(X_train, y_train)
```

```
        # run_model(X_train, X_test, y_train, y_test, **None o 'balanced' o cualquier va
        # Entrenar el modelo
        if  balancedor["nomenclatura"]!="bagging":
            modelo = run_model(X_train, X_test, y_train, y_test, class_weight)




        # Realizar predicciones en el conjunto de prueba
        if  balancedor["nomenclatura"]!="bagging":
            y_pred = modelo.predict(X_test)
        else:
            y_pred =balanceadores.predict(X_test)


        Vprediciones2.append(y_pred)


        # Calcular el informe de clasificación
        reporte_clasificacion = classification_report(y_test, y_pred,output_dict=True,ta

        # Calcular Matthews_corrcoef
        mcc = round(matthews_corrcoef(y_test, y_pred),2)

        # Obtener las métricas de accuracy, macro avg y weighted avg como filas en un Da
        df_resultado = pd.DataFrame(reporte_clasificacion).transpose()
        df_resultado['Métrica'] = df_resultado.index
        df_resultado['Estimador'] = nombre
        df_resultado.set_index(['Estimador', 'Métrica'], inplace=True)
        # Agregar Matthews_corrcoef como una fila en el DataFrame
        df_resultado.loc[(nombre, 'Matthews_corrcoef'), 'Valor'] = mcc
        #df_resultado.loc[(nombre, 'y_pred'), 'Valor'] = str(y_pred)

        # Agregar el DataFrame al resultado
        resultados.append(round(df_resultado,2))

    # Concatenar los DataFrames de resultados en uno solo
    df_final = pd.concat(resultados).fillna('')

    return df_final
```

In [13]:
```
def run_model(X_train, X_test, y_train, y_test, classWeight):

    clf_base = RidgeClassifier(alpha=1.0, class_weight=classWeight, copy_X=True, fit_int
                max_iter=None, positive=False, random_state=42, solver='auto',
                tol=0.0001)
    clf_base.fit(X_train, y_train)
    return clf_base
```

In [14]:
```
def model_evaluation(model, X, y):
    # define cross validation prodedure
    cv = RepeatedKFold(n_splits=10, n_repeats=5, random_state=42)
    scores = cross_val_score(model, X, y, scoring= 'f1_macro', cv = cv)

    return np.mean(scores), np.std(scores)
```

In [15]:
```
def GuardarMetricas(ytest,prediciones):
    target_names = ['Less chance H.A', 'More chance H.A']
    # Calcular el informe de clasificación
    reporte_clasificacion = classification_report(ytest, prediciones,output_dict=True,ta

    # Calcular Matthews_corrcoef
    mcc = round(matthews_corrcoef(ytest, prediciones),2)
```

```
    # Obtener las métricas de accuracy, macro avg y weighted avg como filas en un DataFr
    df_resul = pd.DataFrame(reporte_clasificacion).transpose()
    df_resul =round(df_resul,2)
    df_resul['Métrica'] = df_resul.index

    df_resul.set_index(['Métrica'], inplace=True)

    # Agregar Matthews_corrcoef como una fila en el DataFrame
    df_resul.loc['Matthews_corrcoef', 'Valor'] = mcc


    df_resul=df_resul.fillna('')
    df_resul=df_resul.reset_index()

    return (df_resul)
```

IT Academy - Ciència de Dades (online)

Memoría del Proyecto Data Science:

Heart Attack Analysis & Prediction Dataset.

Juan Javier Hidalgo Gómez

Resumen

Un ataque al corazón es la necrosis isquémica del corazón, generalmente causada por la obstrucción de las arterias que lo irrigan. La detección temprana de esta enfermedad cardiovascular aumenta las posibilidades de cura y pede salvar miles de vidas.

Muchos factores de riesgo que pueden desencadenar un infarto tienen que ver con nuestro estilo de vida actual. el problema es que en muchos casos no lo podemos detectar a tiempo porque carecemos de las herramientas necesarias para predecir cuándo ocurrirá un infarto.

Contexto

Se trata de un conjunto de datos de tipo multivariante, lo que significa que proporciona o implica una variedad de variables matemáticas o estadísticas separadas, análisis de datos numéricos multivariantes. Se compone de 14 atributos que son la edad, el sexo, el tipo de dolor torácico, la tensión arterial en reposo, el colesterol sérico, la glucemia en ayunas, los resultados electrocardiográficos en reposo, la frecuencia cardiaca máxima alcanzada, la angina inducida por el ejercicio, el oldpeak - depresión del ST inducida por el ejercicio en relación con el reposo, la pendiente del pico del segmento ST del ejercicio, el número de vasos principales y la talasemia. Esta base de datos incluye 76 atributos, pero todos los estudios publicados se refieren al uso de un subconjunto de 14 de ellos. La base de datos de Cleveland es la única utilizada por los investigadores de ML hasta la fecha. Una de las principales tareas de este conjunto de datos consiste en predecir, basándose en los atributos dados de un paciente, si esa persona en concreto padece o no una enfermedad cardiaca, y otra es la tarea experimental de diagnosticar y averiguar varias ideas a partir de este conjunto de datos que podrían ayudar a comprender mejor el problema.

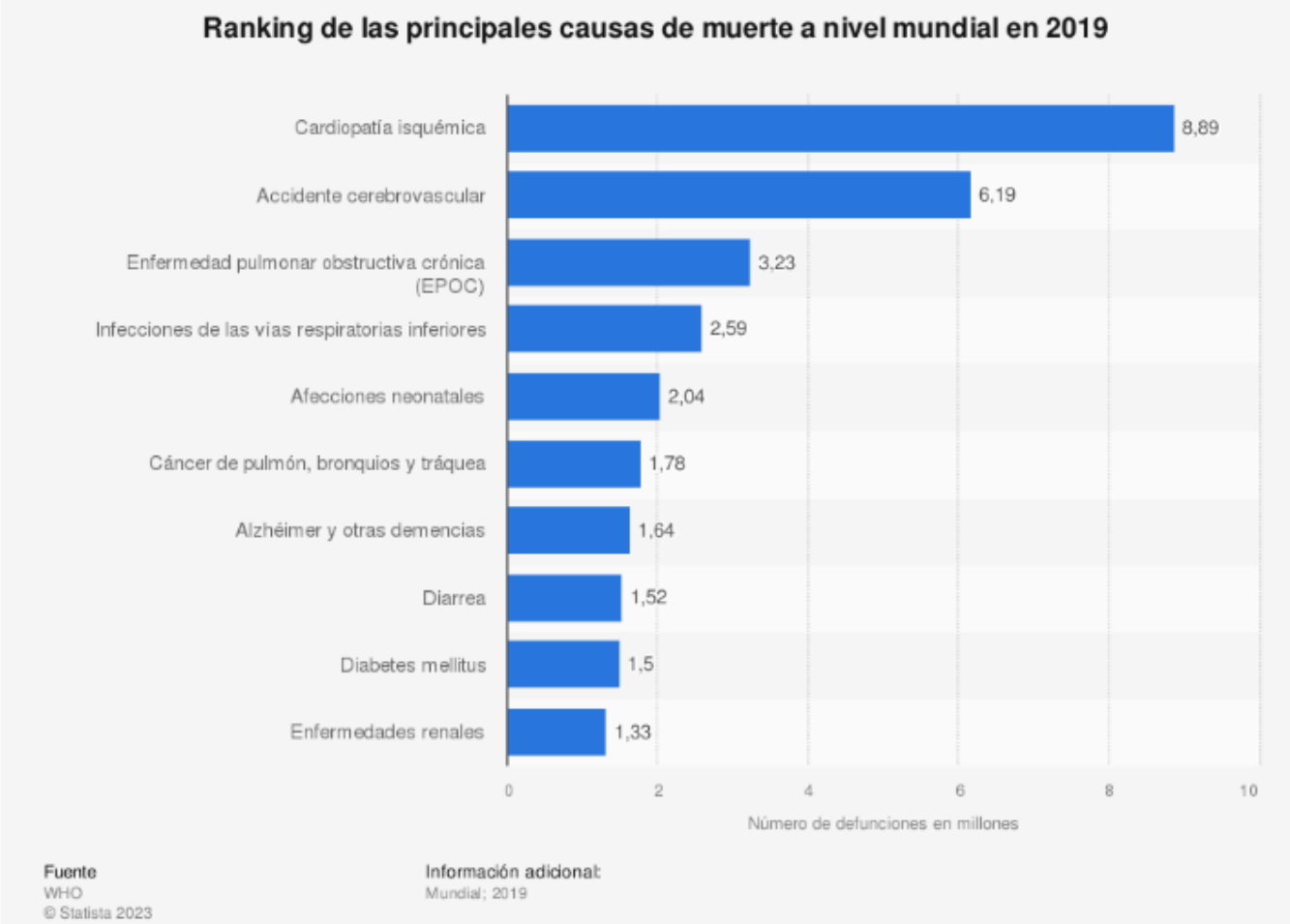
Indice general

Introducción

Un infarto es la muerte del tejido (necrosis) debido a un suministro de sangre inadecuado en el área agectada. Puede ocurrir en cualquier órgano o músculo, pero los casos más comunes son corazón, cerebro, intestino, riñón o pulmón.

Un ataque cardíaco ocurre cuando el flujo de sangre oxigenada se obstruye repentinamente en una o más de las arterias coronarias que abastecen al músculo cardíaco y una sección del músculo no puede obtener suficiente oxígeno. La obstrucción usualmente ocurre cuando una placa se rompe.

Las enfermedades cardiovasculares son la principal causa de muerte tanto en hombres como en mujeres en todo el mundo. Existen muchos factores de riesgo como hipertensión arterial, tabaquismo, diabetes, etc..



La detección precoz de un infarto puede ser vital para que la hipoxia (falta de oxigeno por la falta de riego sanguineo) dure lo menos posible y también poder realizar una intervención a tiempo.

Volver Índice general

Objetivo

Los principales objetivos de este proyecto son los siguientes:

- **Desarrollar un modelo de Machine Learning** con la técnica de aprendizaje supervizado, que pueda predecir a partir de una serie de datos de entrada si una persona va a sufrir un infarto.
- **Estudiar y analizar** los datos de nuestro DataSet. La creación de nuestro modelo de ML, así como las herramientas necesarias para su elaborar y optimizar el mismo.
- **Definir** una serie de pruebas con diferentes módelos de aprendizaje supervisado, para encontrar el más idóneo.

Volver Índice general

1. Metodología

Este capitulo incluye el estudio de los atributos de nuestro Dataset, lectura, visualización y descripción.

1.1 Descripción del DataSet

- **Fichero heart.csv:**

Age : Edad del paciente.

Sex : Género del paciente.

cp : Chest Pain type chest pain type.

- **1:** typical angina.
- **2:** atypical angina.
- **3:** non-anginal pain.
- **4:** asymptomatic.

trtbps : resting blood pressure (in mm Hg).

chol : cholestoral in mg/dl fetched via BMI sensor.

fbs : (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false).

restecg : resting electrocardiographic results.

- **0:** normal.
- **1:** having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV).
- **2:** showing probable or definite left ventricular hypertrophy by Estes' criteria.

thalachh : maximum heart rate achieved.

exng : Exercise induced angina (1 = yes; 0 = no).

oldpeak : Depresión del ST inducida por el ejercicio en relación con el reposo.

slp : Pendiente del Pico Ejercicios Segmento ST.

caa : number of major vessels (0-3).

thall : maximum heart rate achieved.

target : 0= less chance of heart attack 1= more chance of heart attack.

- **Fichero o2Saturation.csv:**

Sat_level : O2 saturation.

1.2 Lectura y visualización de los datos

Para realizar está investigación, analizamos un dataset obtenido a través de la plataforma Kaggle, dicha plataforma podemos obtener y publicar conjuntos de datos. Este dataset contiene la información de un total de 303 registros médicos de pacientes, a los cuales se ha realizado una prueba de esfuerzo.

In [16]:
```
# Primeras cinco filas
df_tabla = heart[0:5]
dibujartabla('\nDataSet Heart:\n',df_tabla)
```

DataSet Heart:

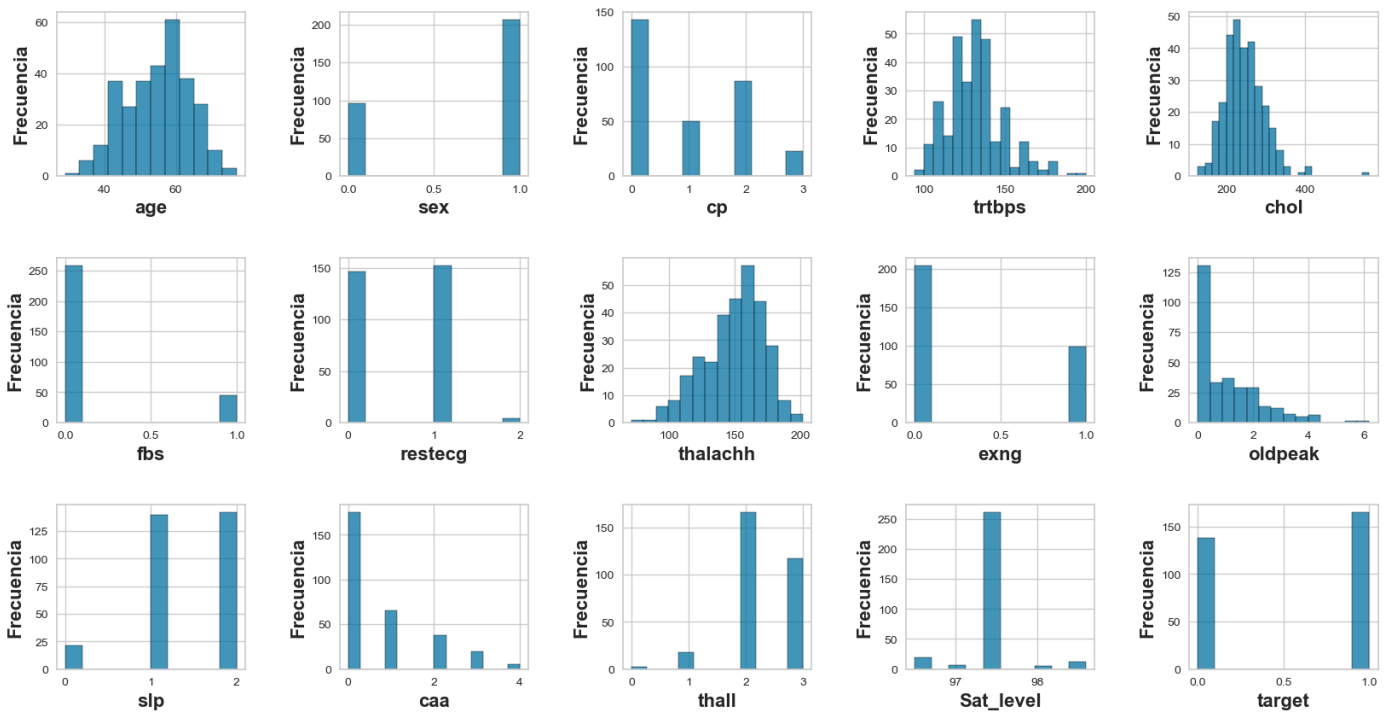
age	sex	cp	trtbp	chol	fbs	res	thala	exi	oldpe	slp	caa	tha	Sat_le	tar
63	1	3	145	233	1	0	150	0	2.3	0	0	1	98.6	1
37	1	2	130	250	0	1	187	0	3.5	0	0	2	98.6	1
41	0	1	130	204	0	0	172	0	1.4	2	0	2	98.6	1
56	1	1	120	236	0	1	178	0	0.8	2	0	2	98.1	1
57	0	0	120	354	0	1	163	1	0.6	2	0	2	97.5	1

1.3 Análisis de los datos

Los 15 atributos están compuestos tanto por datos discretos como continuos.

In [17]:
```
#print(Style.BRIGHT +'\nPruebas de Contraste de Normalidad:\n '+Style.RESET_ALL)
print(Style.BRIGHT+'\nDistribución de Atributos:\n '+Style.RESET_ALL)
crearhistograma(heart,list(heart.columns))
sns.set_theme(style="white", rc=None)
```

Distribución de Atributos:



El atributo que nos interesa predecir es **target**, que nos indica si la prueba de esfuerzo determina si el paciente tiene más o menos posibilidades de tener un ataque de miocardio.
Vamos a analizar el resto de atributos con más profundidad para ver el contenido de los mismos y como se relacionan con nuestro **target**.

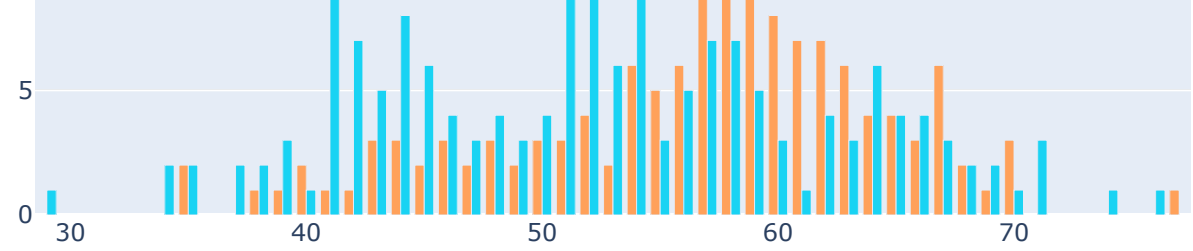
In [18]:
```
df = heart.copy()
df=df.assign(target_descrip=df['target'])
cambio = {0: 'Less chance of Heart Attack',1:'More chance of Heart Attack'}
df.target_descrip = [cambio[item] for item in df.target_descrip]
```

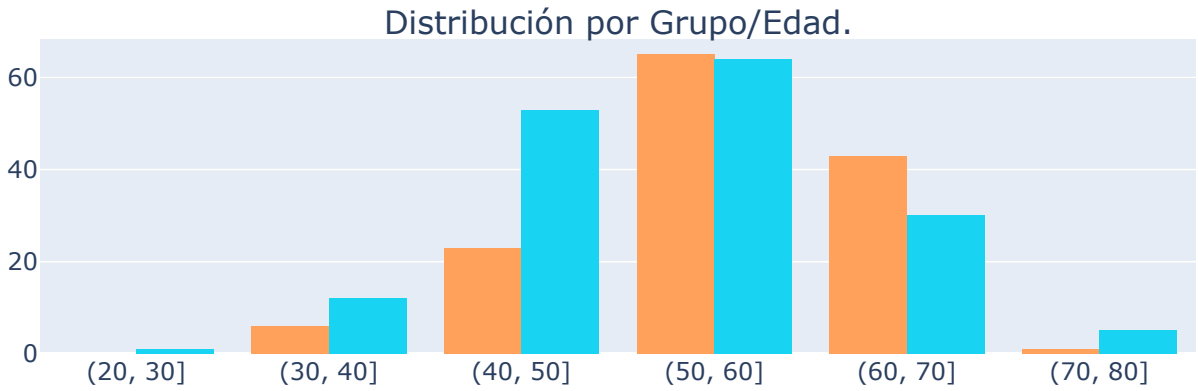
Edad, (Age):

Indica la edad del paciente.

In [19]:
```
df1=df.groupby(by=["age","target_descrip"]).size().reset_index(name="counts")
df1['grupo_edad'] = pd.cut(df1['age'], bins=range(0,df['age'].max()+10, 10)) #bins=range
#df1['grupo_edad'].unique()
#df['age'].max()//10+1
```

In [20]:
```
#datos a pasar dataframe,atributo,leyendas,titulos)
atributo='age'
atributo2='grupo_edad'
leyenda_p=" "
leyenda_s="Target"
titulo_p="Distribución por Edad."
titulo_s="Distribución por Grupo/Edad."
array=['(20, 30]','(30, 40]','(40, 50]','(50, 60]','(60, 70]','(70, 80]']
X1, Y1 , X2, Y2 = 0.5,1,0.5,0.37
estudio_grupo(df1, atributo, atributo2, leyenda_p, leyenda_s, titulo_p, titulo_s, array,
```





Podemos ver en las dos gráficas, que entre los 40 años y menos de 60 años tenemos la máxima posibilidad de un ataque al corazón.

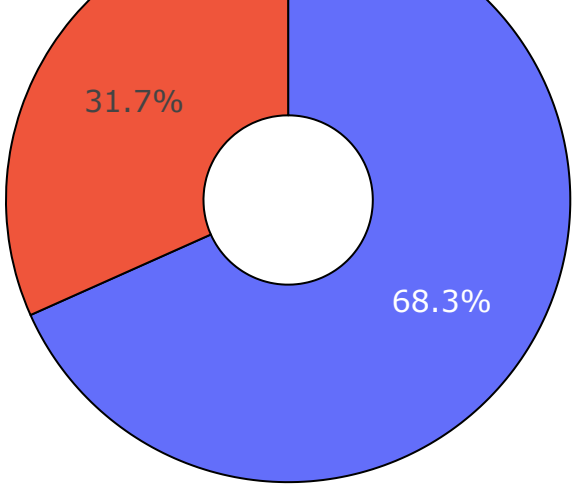
Género, (Sex):

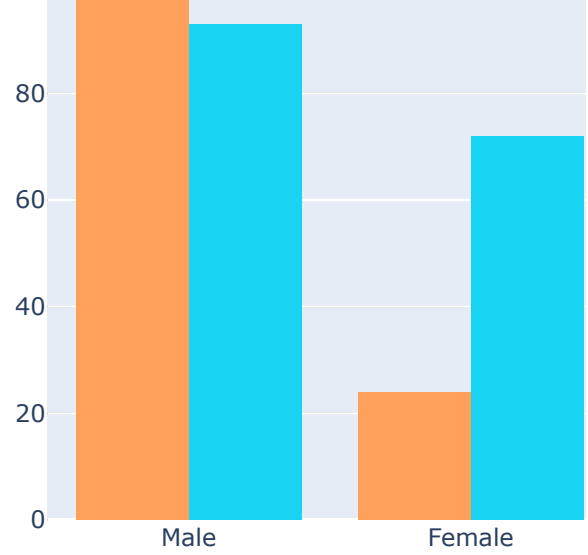
Indica el género del paciente.

- 0: Mujer.
- 1: Hombre.

In [21]:
```
df=df.assign(genero=df['sex'])
cambio = {0: 'Female',1:'Male'}
df.genero = [cambio[item] for item in df.genero]
#df = px.data.tips()
```

In [22]:
```
#datos a pasar dataframe,atributo,leyendas,titulos)
atributo='genero'
leyenda_p="Género"
leyenda_s="Target"
titulo_p="Porcentaje por Género."
titulo_s="Probabilidad de Infarto en función del Género."
X1, Y1 , X2, Y2 = 0.09,1.1,0.8,1.1
estudio_atributos(df,atributo,leyenda_p,leyenda_s,titulo_p,titulo_s,X1, Y1 , X2, Y2)
```





Tras realizar la prueba de esfuerzo y con los datos que nos aportan este dataset las mujeres en proporción tienen más posibilidades de tener un ataque de miocardio.

- Hombres = 68.3%
- Mujeres = 31.7%

Tipo de Dolor torácico, (cp):

Indica el dolor del paciente.

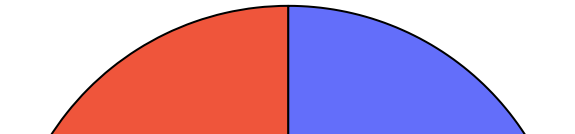
- 0: Asymptomatic.
- 1: Typical Angina.
- 2: Atypical Angina.
- 3: Non-Anginal Pain.

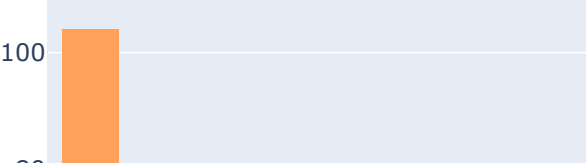
In [23]:
```
df=df.assign(cp_descrip=df['cp'])
cambio = {0: 'Asymptomatic' , 1: 'Typical Angina' ,  2: 'Atypical Angina', 3: 'Non-Angin
df.cp_descrip = [cambio[item] for item in df.cp_descrip]
```

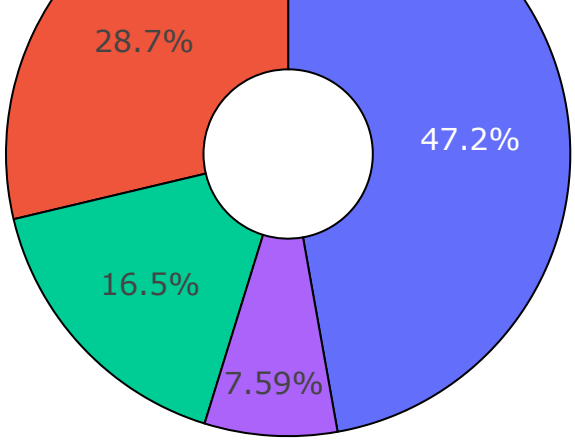
In [24]:
```
#datos a pasar dataframe,atributo,leyendas,titulos)
atributo='cp_descrip'
leyenda_p="Chest Pain Type"
leyenda_s="Target"
titulo_p="Porcentaje por el Chest Pain."
titulo_s="Distribución por Heart Attack en función del Chest Pain."
X1, Y1 , X2, Y2 = 0.09,1.1,0.9,1.1
estudio_atributos(df,atributo,leyenda_p,leyenda_s,titulo_p,titulo_s,X1, Y1 , X2, Y2)
```

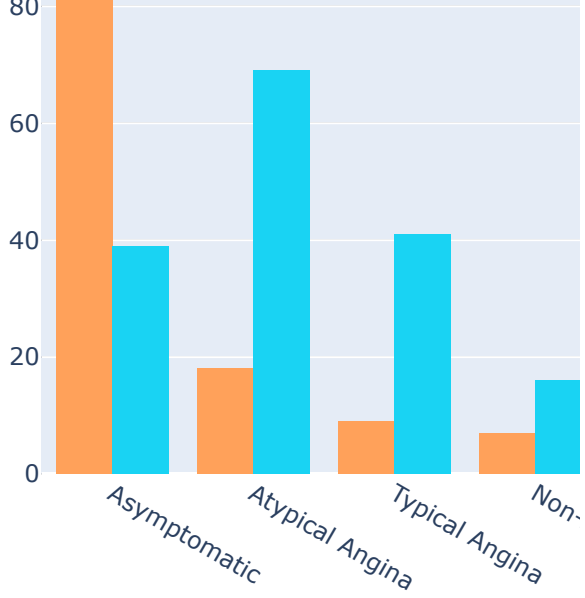
Porcentaje por el Chest Pain.

Distribución por Heart Attack en fu









Vemos que el tipo de dolor torácico Angina Atípica es el que ha dado lugar al mayor número de pacientes con riesgo de sufrir un infarto de miocardio.

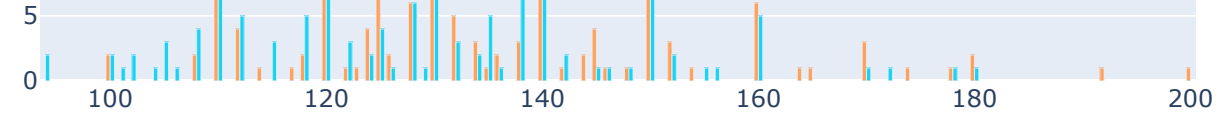
- 0: Asymptomatic = 47.2%
- 1: Typical Angina = 16.5%
- 2: Atypical Angina = 2.7%
- 3: Non-Anginal Pain = 7.59%

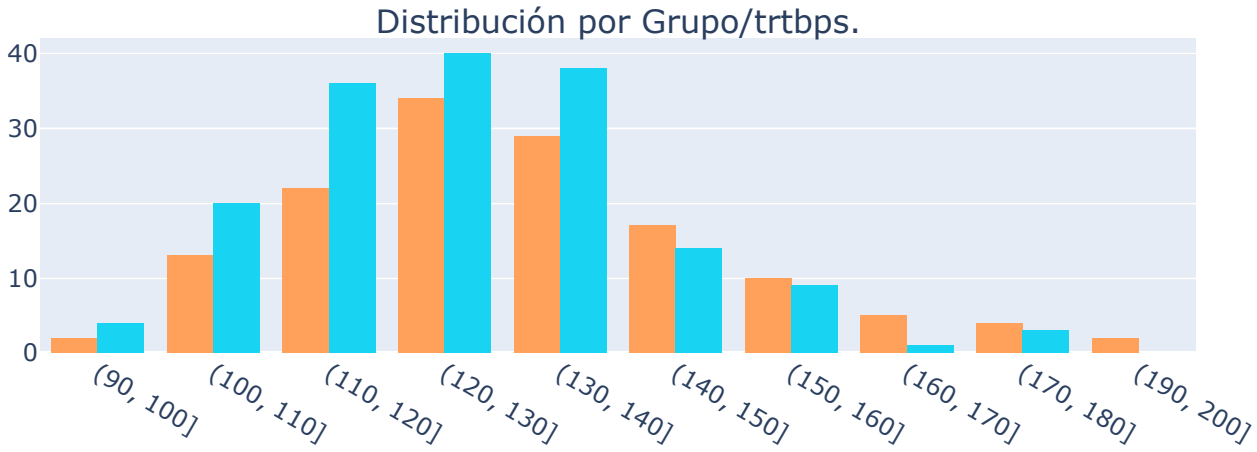
Presión arterial en reposo, (trtbps):

Indica la presión arterial en reposo (en mm Hg al ingreso en el hospital) del paciente.

In [25]:
```
df1=df.groupby(by=["trtbps","target_descrip"]).size().reset_index(name="counts")
df1['grupo_trtbps'] = pd.cut(df1['trtbps'], bins=range(0,df['trtbps'].max()+10,10))
#df1['grupo_trtbps'].unique() #para añadir en el array ( a estudiar para hacerlo automát
```

In [26]:
```
#datos a pasar dataframe,atributo,leyendas,titulos)
atributo='trtbps'
atributo2='grupo_trtbps'
leyenda_p=" "
leyenda_s="Target"
titulo_p="Distribución por trtbps."
titulo_s="Distribución por Grupo/trtbps."
array=['(90, 100]','(100, 110]','(110, 120]','(120, 130]','(130, 140]','(140, 150]','(15
#array=df1['grupo_trtbps'].uniqu'e()
X1, Y1 , X2, Y2 = 0.5,1,0.5,0.37
estudio_grupo(df1, atributo, atributo2, leyenda_p, leyenda_s, titulo_p, titulo_s, array,
```





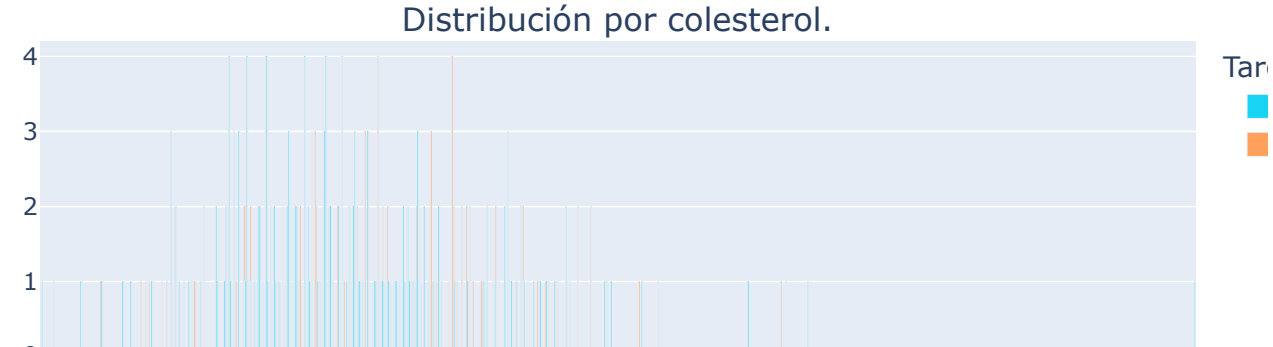
Según los datos del Dataset, entre los grupos de 110 mm/Hg a 139 mm/Hg presión arterial en reposo, tendriamos los pacientes con mayor riesgo de infarto de miocardio.

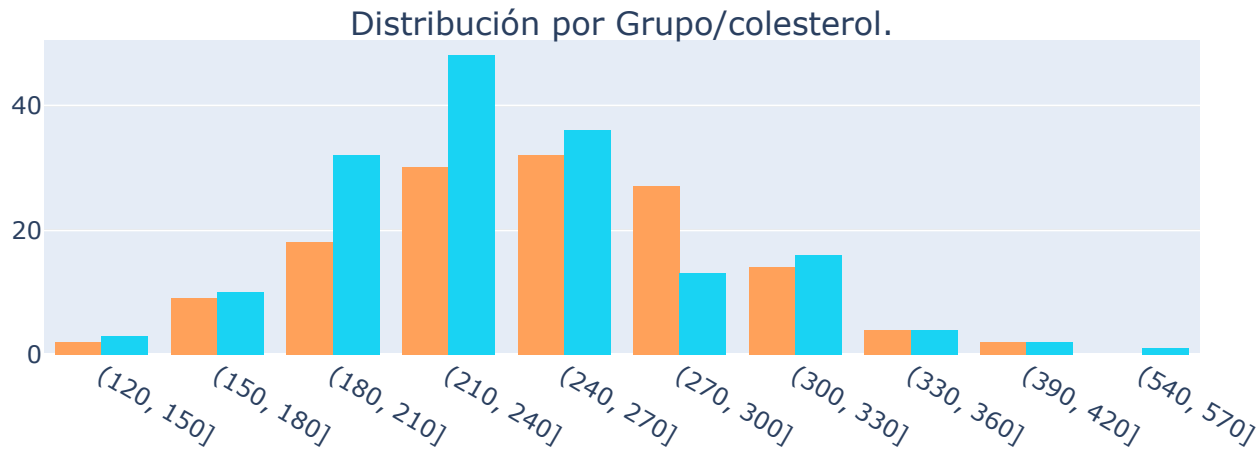
Colesterol sérico en mg/dl, (chol):

Indica el colesterol del paciente.

In [27]:
```
df1=df.groupby(by=["chol","target_descrip"]).size().reset_index(name="counts")
df1['grupo_chol'] = pd.cut(df1['chol'], bins=range(0,df['chol'].max()+10,30))
#df1['grupo_chol'].unique() #para añadir en el array ( a estudiar para hacerlo automátic
```

In [28]:
```
#datos a pasar dataframe,atributo,leyendas,titulos)
atributo='chol'
atributo2='grupo_chol'
leyenda_p=" "
leyenda_s="Target"
titulo_p="Distribución por colesterol."
titulo_s="Distribución por Grupo/colesterol."
array=['(120, 150]','(150, 180]','(180, 210]','(210, 240]','(240, 270]','(270, 300]','(3
#array=df1['grupo_trtbps'].unique()
X1, Y1 , X2, Y2 = 0.5,1,0.5,0.37
estudio_grupo(df1, atributo, atributo2, leyenda_p, leyenda_s, titulo_p, titulo_s, array,
```





Según los datos del Dataset, entre los grupos de 180 mg/dl a 269 mg/dl de cholestoral tendriamos los pacientes con mayor riesgo de infarto de miocardio.

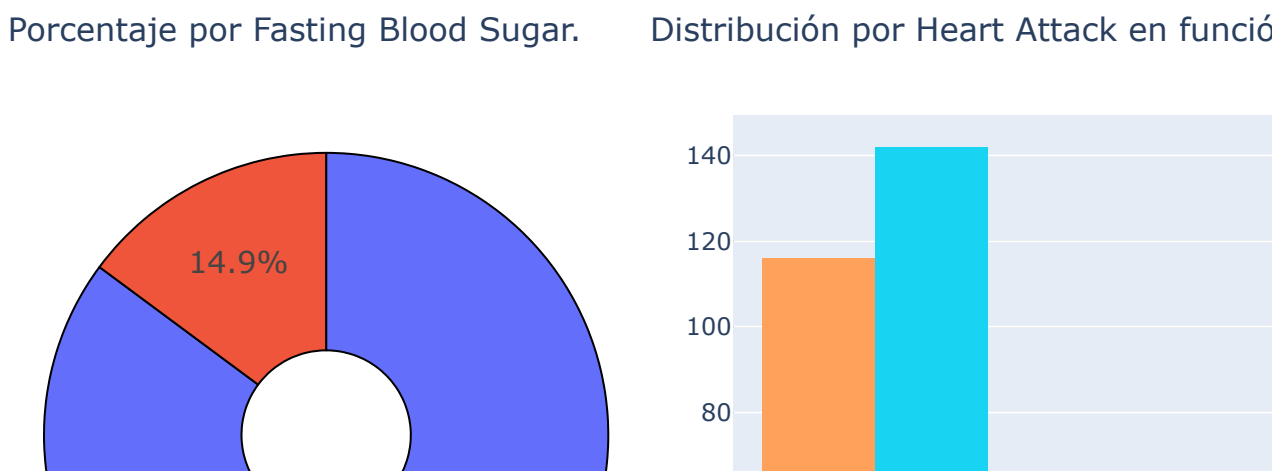
Azucar en la Sangre, (fbs):

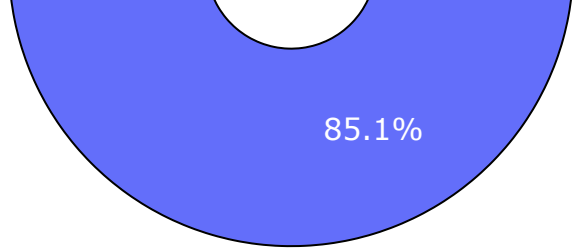
Indica el nivel de glucosa del paciente.

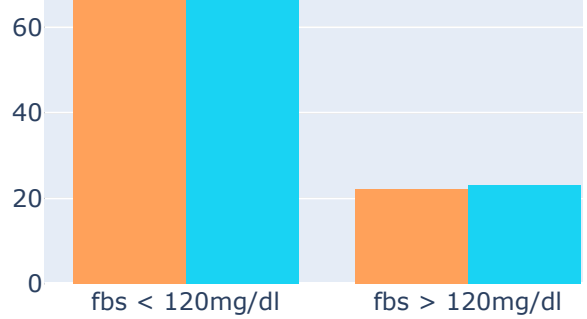
- False (0): Fasting Blood Sugar < 120mg/dl
- True (1): Fasting Blood Sugar > 120mg/dl

In [29]:
```
df=df.assign(fbs_descrip=df['fbs'])
cambio = {0: 'fbs < 120mg/dl' , 1: 'fbs > 120mg/dl'}
df.fbs_descrip = [cambio[item] for item in df.fbs_descrip]
```

In [30]:
```
#datos a pasar dataframe,atributo,leyendas,titulos)
atributo='fbs_descrip'
leyenda_p="Fasting Blood Sugar"
leyenda_s="Target"
titulo_p="Porcentaje por Fasting Blood Sugar."
titulo_s="Distribución por Heart Attack en función del Fasting Blood Sugar."
X1, Y1 , X2, Y2 = 0.2,1.1,0.9,1.1
estudio_atributos(df,atributo,leyenda_p,leyenda_s,titulo_p,titulo_s,X1, Y1 , X2, Y2)
```







Según los datos de este dataset, un paciente con más de 120mg/dl no aporta valor, aunque podría ser diabético (se necesitarian realizar más pruebas). Pero existe una probabilidad mayor de sufrir un infarto si la glucemia es inferior a 120mg/dl.)

- fbs < 120mg/dl = 85.1%
- fbs > 120mg/dl = 14.9%

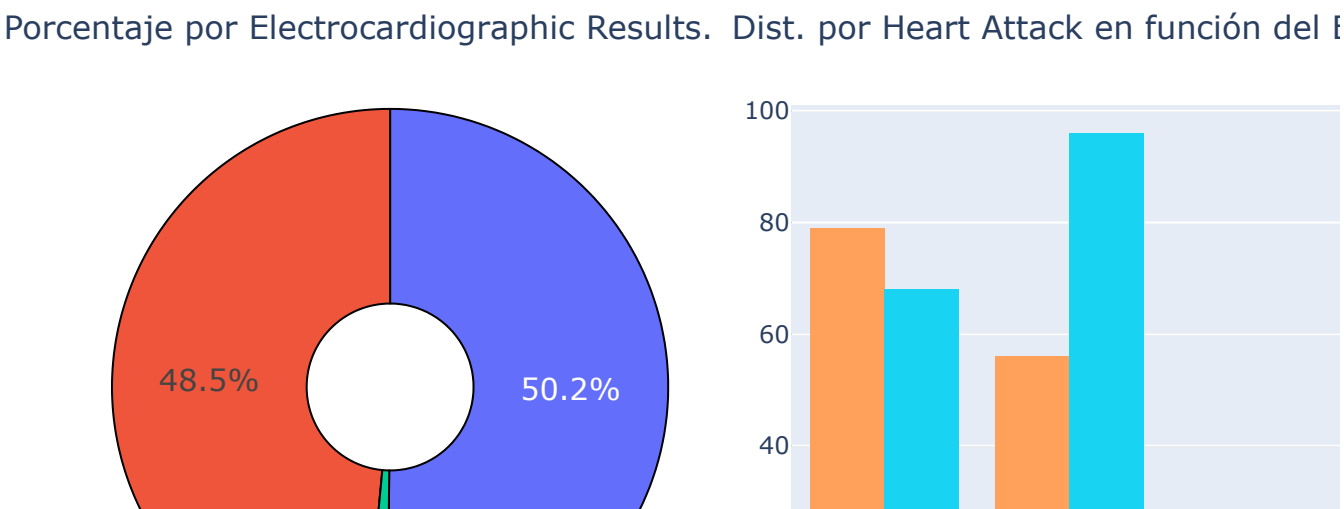
Resultados electrocardiográficos, (restecg):

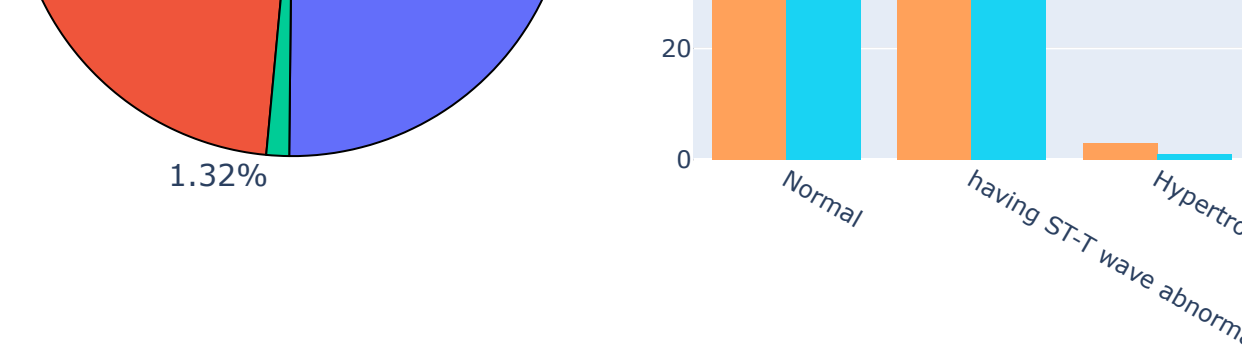
Resultado electrocardiográfico del paciente.

- 0: normal.
- 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV).
- 2: showing probable or definite left ventricular hypertrophy by Estes' criteria.

In [31]:
```
df=df.assign(restecg_descrip=df['restecg'])
cambio = {0: 'Normal' , 1: 'having ST-T wave abnormality', 2: 'Hypertrophy'}
df.restecg_descrip = [cambio[item] for item in df.restecg_descrip]
```

In [32]:
```
#datos a pasar dataframe,atributo,leyendas,titulos)
atributo='restecg_descrip'
leyenda_p="Electrocardiographic Results"
leyenda_s="Target"
titulo_p="Porcentaje por Electrocardiographic Results."
titulo_s="Dist. por Heart Attack en función del Electro. Results."
X1, Y1 , X2, Y2 = 0.2,1.1,0.85,1.1
estudio_atributos(df,atributo,leyenda_p,leyenda_s,titulo_p,titulo_s,X1, Y1 , X2, Y2)
```





Vemos que cuando un paciente tiene un eletrogram de reposo con un resultado 'having ST-T wave abnormality', hay mayor probabilidad de riesgo de sufrir un infarto de miocardio.

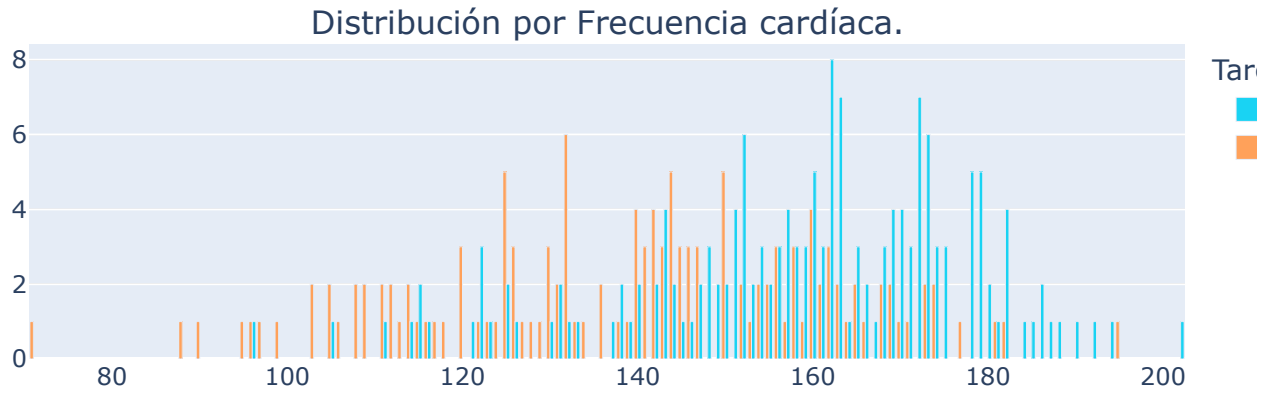
- 0: normal = 48.5%
- 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV) = 50.2%
- 2: showing probable or definite left ventricular hypertrophy by Estes criteria = 1.32%

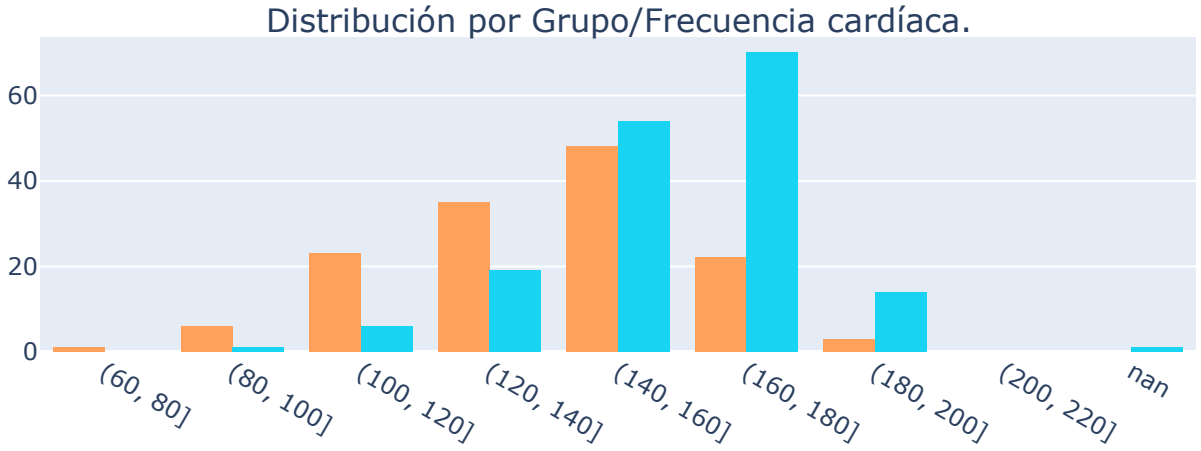
Frecuencia cardíaca máxima alcanzada, (thalachh):

Indica el colesterol del paciente.

In [33]:
```
df1=df.groupby(by=["thalachh","target_descrip"]).size().reset_index(name="counts")
df1['grupo_thalachh'] = pd.cut(df1['thalachh'], bins=range(0,df['thalachh'].max()+10,20)
#df1['grupo_thalachh'].unique() #para añadir en el array ( a estudiar para hacerlo autom
```

In [34]:
```
#datos a pasar dataframe,atributo,leyendas,titulos)
atributo='thalachh'
atributo2='grupo_thalachh'
leyenda_p=" "
leyenda_s="Target"
titulo_p="Distribución por Frecuencia cardíaca."
titulo_s="Distribución por Grupo/Frecuencia cardíaca."
array=['(60, 80]','(80, 100]','(100, 120]','(120, 140]','(140, 160]','(160, 180]','(180,
#array=df1['grupo_trtbps'].unique()
X1, Y1 , X2, Y2 = 0.5,1,0.5,0.37
estudio_grupo(df1, atributo, atributo2, leyenda_p, leyenda_s, titulo_p, titulo_s, array,
```





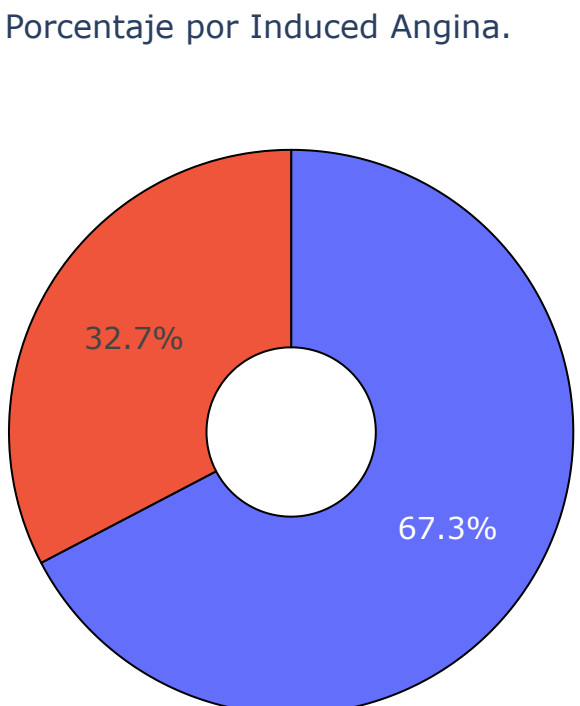
Según los datos del Dataset, entre los grupos de 140 a 180 de Frecuencia cardíaca tendriamos los pacientes con mayor riesgo de infarto de miocardio.

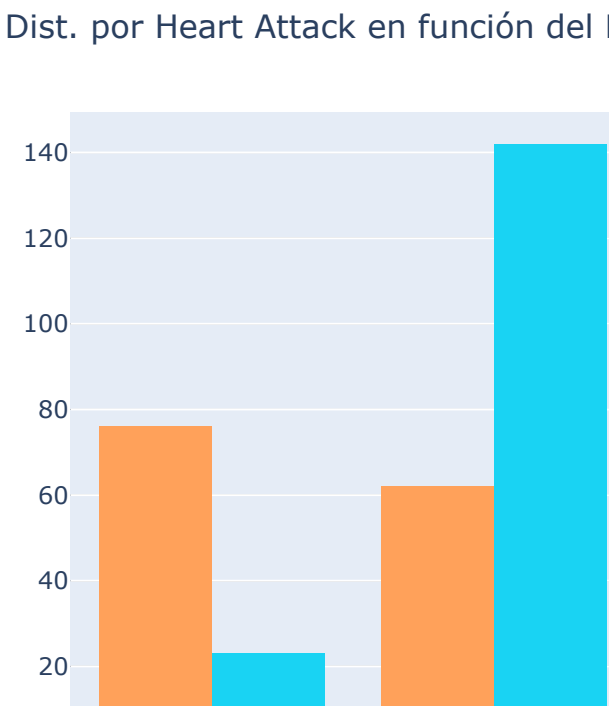
Angina inducida por el ejercicio, (exng):

- 0: No.
- 1: Yes.

In [35]:
```
df=df.assign(exng_descrip=df['exng'])
cambio = {0: 'No' , 1: 'Yes'}
df.exng_descrip = [cambio[item] for item in df.exng_descrip]
```

In [36]:
```
#datos a pasar dataframe,atributo,leyendas,titulos)
atributo='exng_descrip'
leyenda_p="Exercise Induced Angina"
leyenda_s="Target"
titulo_p="Porcentaje por Induced Angina."
titulo_s="Dist. por Heart Attack en función del Exercise Induced Angina."
X1, Y1 , X2, Y2 = 0.2,1.1,0.9,1.1
estudio_atributos(df,atributo,leyenda_p,leyenda_s,titulo_p,titulo_s,X1, Y1 , X2, Y2)
```





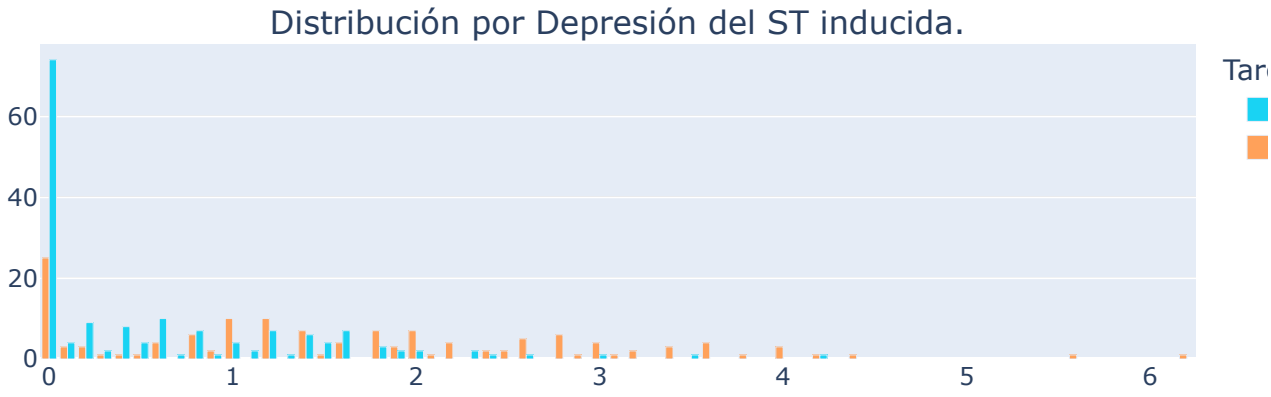
Esta variable nos indica que cuando la angina no es inducida por el ejercicio, hay muchas más probabilidades de que un paciente corra el riesgo de sufrir un infarto de miocardio.

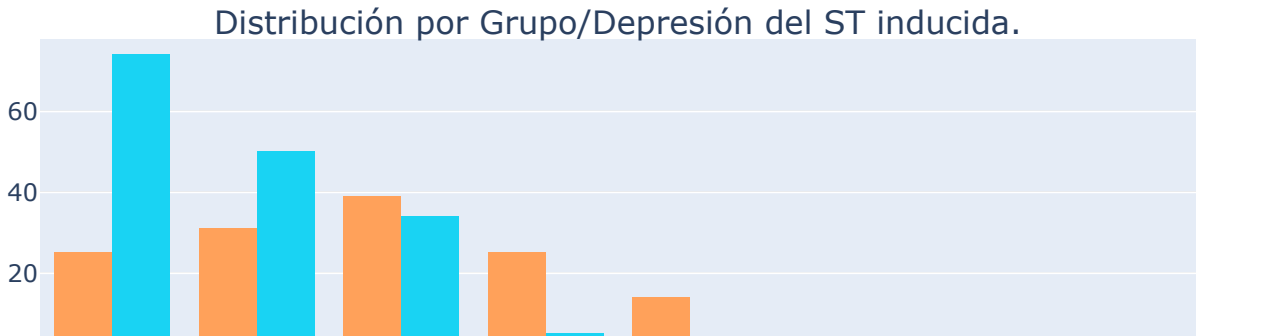
- No = 67.3%
- Yes = 32.7%

Depresión del ST inducida por el ejercicio en relación con el reposo, (oldpeak).

In [37]:
```
df1=df.groupby(by=["oldpeak","target_descrip"]).size().reset_index(name="counts")
df1['grupo_oldpeak'] = pd.cut(df1['oldpeak'], bins=range(int(math.modf(df['oldpeak'].min
#df1['grupo_oldpeak'].unique() #para añadir en el array ( a estudiar para hacerlo automá
```

In [38]:
```
#datos a pasar dataframe,atributo,leyendas,titulos)
atributo='oldpeak'
atributo2='grupo_oldpeak'
leyenda_p=" "
leyenda_s="Target"
titulo_p="Distribución por Depresión del ST inducida."
titulo_s="Distribución por Grupo/Depresión del ST inducida."
array=['(0, 1.0]','(0, 1.0]','(1.0, 2.0]','(2.0, 3.0]','(3.0, 4.0]','(4.0, 5.0]','(5.0,
#array=df1['grupo_trtbps'].unique()
X1, Y1 , X2, Y2 = 0.5,1,0.5,0.37
estudio_grupo(df1, atributo, atributo2, leyenda_p, leyenda_s, titulo_p, titulo_s, array,
```





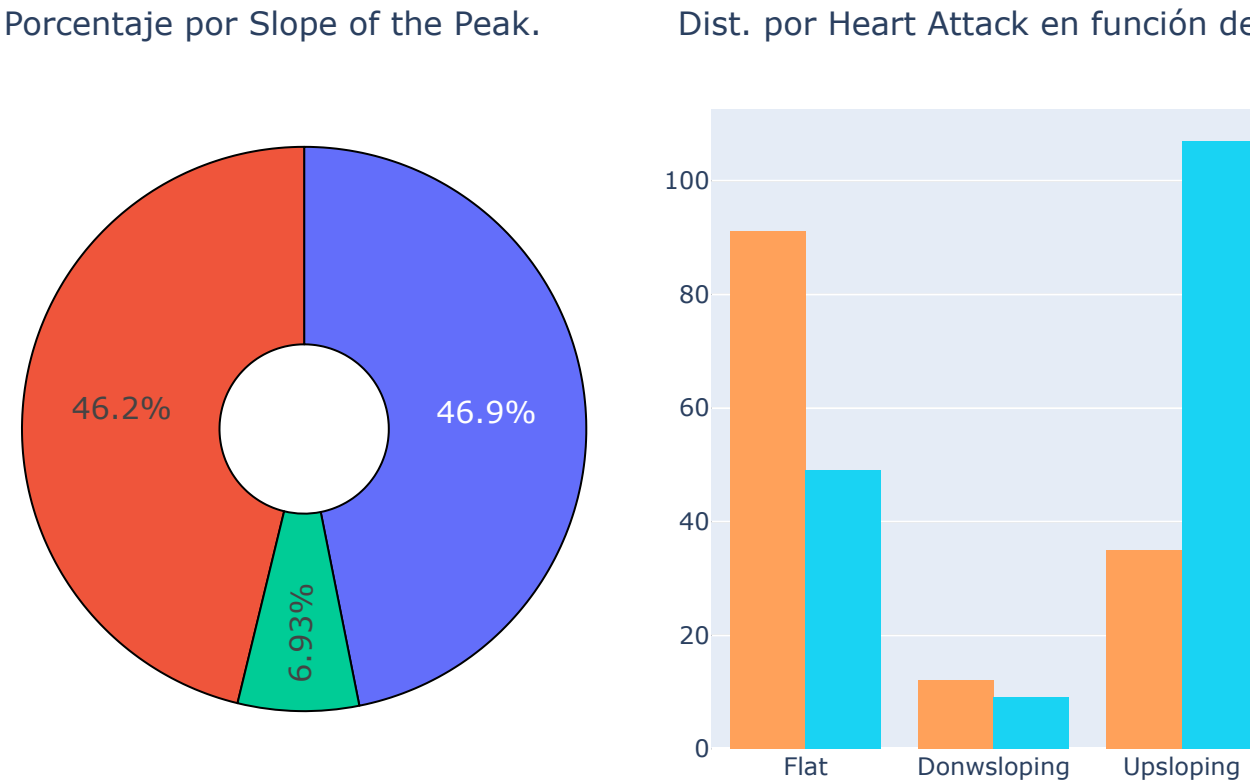
Pendiente del Pico Ejercicios Segmento ST, (slp):

Prueba de esfuerzo electrocardiográfica.

- 0: Donwsloping.
- 1: Flat.
- 2: Upsloping.

In [39]:
```
df=df.assign(slp_descrip=df['slp'])
cambio = {0:'Donwsloping',1:'Flat' ,2:'Upsloping' }
df.slp_descrip = [cambio[item] for item in df.slp_descrip]
```

In [40]:
```
#datos a pasar dataframe,atributo,leyendas,titulos)
atributo='slp_descrip'
leyenda_p="ST Segment"
leyenda_s="Target"
titulo_p="Porcentaje por Slope of the Peak."
titulo_s="Dist. por Heart Attack en función del Slope of the Peak."
X1, Y1 , X2, Y2 = 0.2,1.1,0.88,1.1
estudio_atributos(df,atributo,leyenda_p,leyenda_s,titulo_p,titulo_s,X1, Y1 , X2, Y2)
```



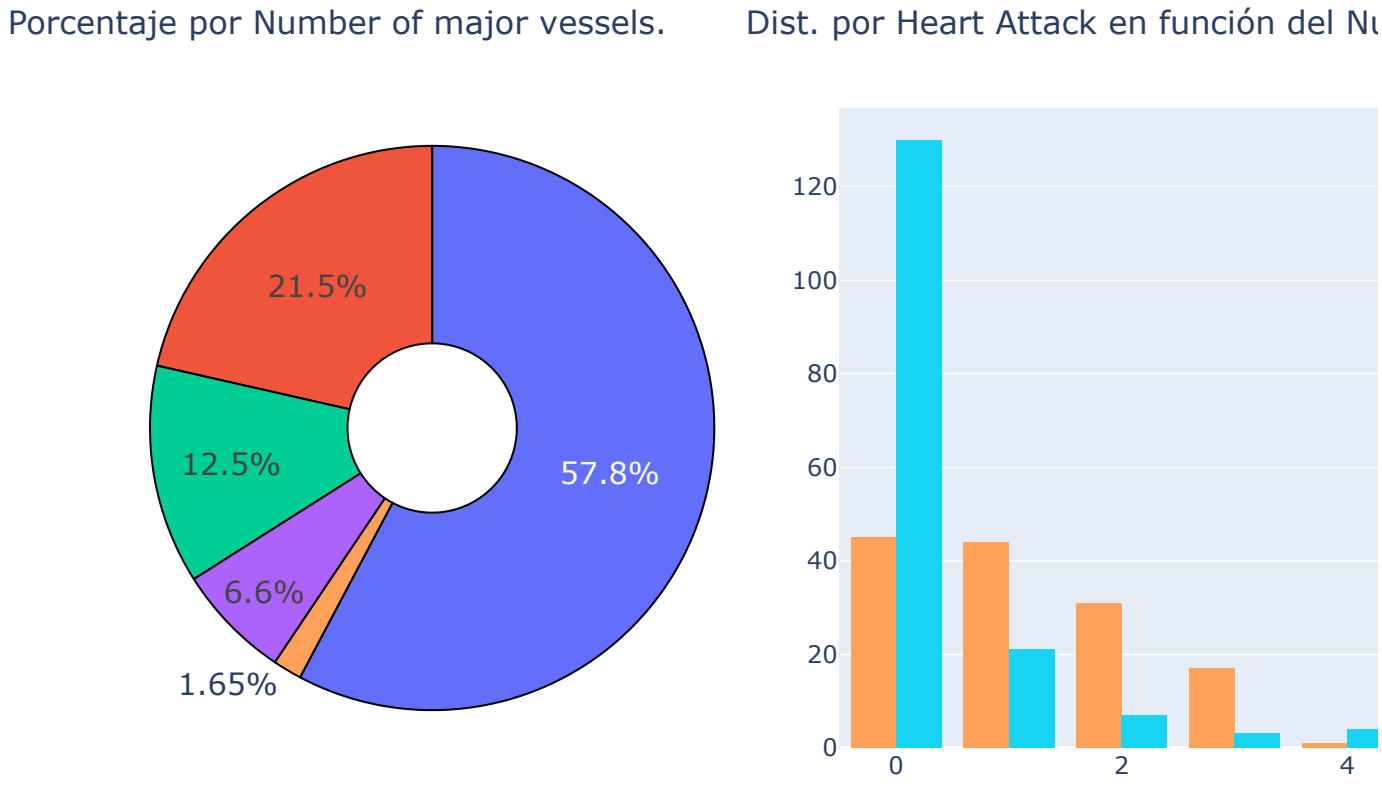
Vemos que cuando la pendiente es ascendente, hay una probabilidad mucho mayor de que un paciente sufra un infarto de miocardio.

- 0: Donwsloping = 6.95%
- 1: Flat = 46.4%
- 2: Upsloping = 46.7%

Number of major vessels, (caa):

Indica el número de vasos principales (0-4) coloreados por la fluroscopia. Cuanto mayor sea el movimiento sanguíneo, mejor, de modo que las personas con ***caa*** igual a 0 tienen más probabilidades de padecer enfermedades cardíacas.

In [41]:
```
#datos a pasar dataframe,atributo,leyendas,titulos)
atributo='caa'
leyenda_p="Number of major vessels"
leyenda_s="Target"
titulo_p="Porcentaje por Number of major vessels."
titulo_s="Dist. por Heart Attack en función del Number of major vessels."
X1, Y1 , X2, Y2 = 0.15,1.1,0.88,1.1
estudio_atributos(df,atributo,leyenda_p,leyenda_s,titulo_p,titulo_s,X1, Y1 , X2, Y2)
```



Los pacientes que tienen el vessels igual a cero, tienen mayor riesgo de sufrir un infarto de miocardio.

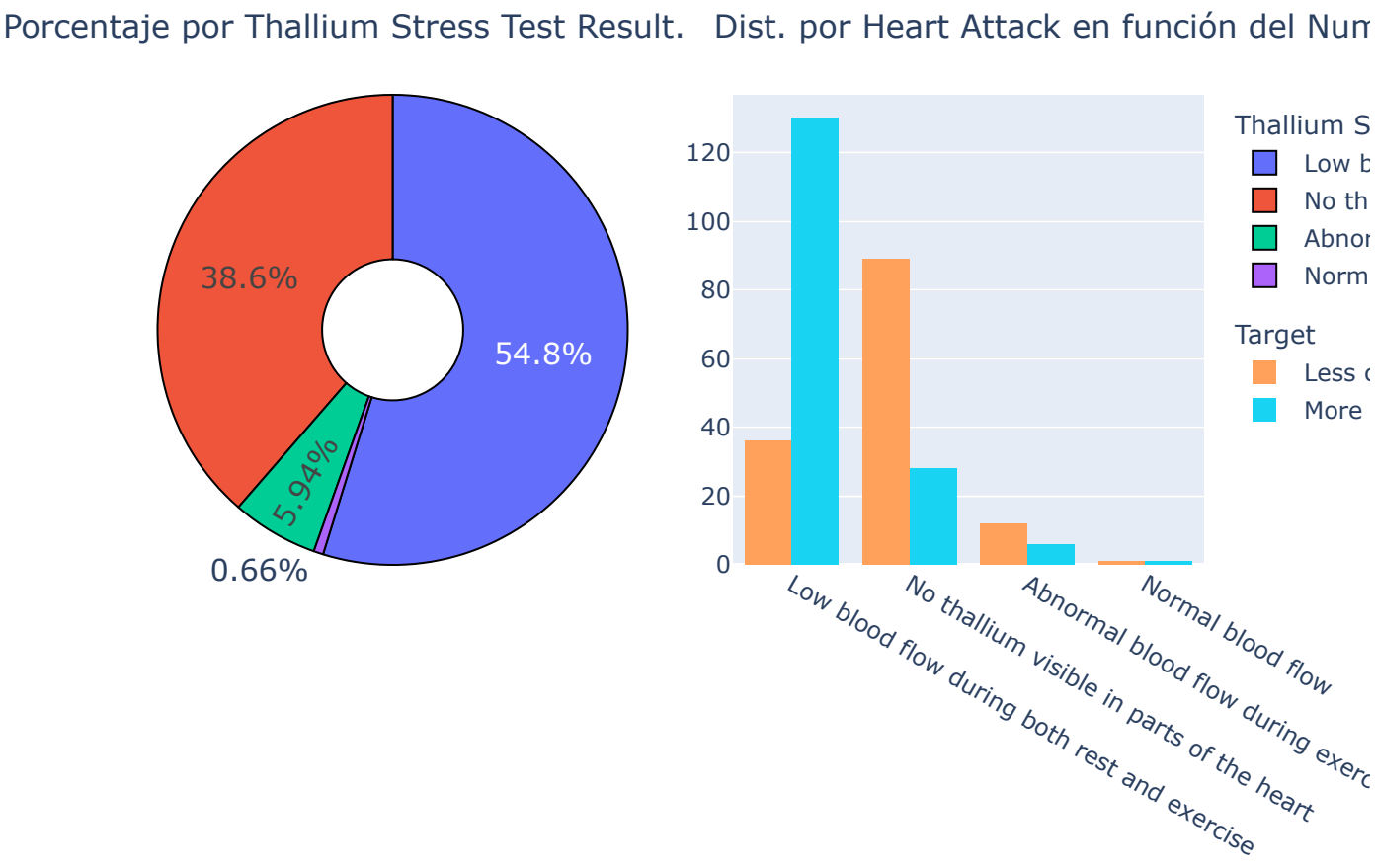
- 0 = 57.8%
- 1 = 21.5%
- 2 = 12.5%
- 3 = 6.6%
- 4 = 1.65%

Thallium Stress Test Result, (thall):

La prueba de esfuerzo con talio es un estudio imagenológico que muestra a su médico qué tan bien fluye la sangre hacia su corazón. Mide el flujo sanguíneo durante el descanso y después del ejercicio. La prueba de esfuerzo con talio también se llama prueba de esfuerzo nuclear, prueba de cinta rodante, prueba de perfusión de esfuerzo o SPECT cardíaca.

In [42]:
```
df=df.assign(thall_descrip=df['thall'])
cambio = {0:'Normal blood flow',1:'Abnormal blood flow during exercise' ,2:'Low blood fl
df.thall_descrip = [cambio[item] for item in df.thall_descrip]
```

In [43]:
```
#datos a pasar dataframe,atributo,leyendas,titulos)
atributo='thall_descrip'
leyenda_p="Thallium Stress Test Result"
leyenda_s="Target"
titulo_p="Porcentaje por Thallium Stress Test Result."
titulo_s="Dist. por Heart Attack en función del Number of major vessels."
X1, Y1 , X2, Y2 = 0.18,1.1,1.01,1.1
estudio_atributos(df,atributo,leyenda_p,leyenda_s,titulo_p,titulo_s,X1, Y1 , X2, Y2)
```



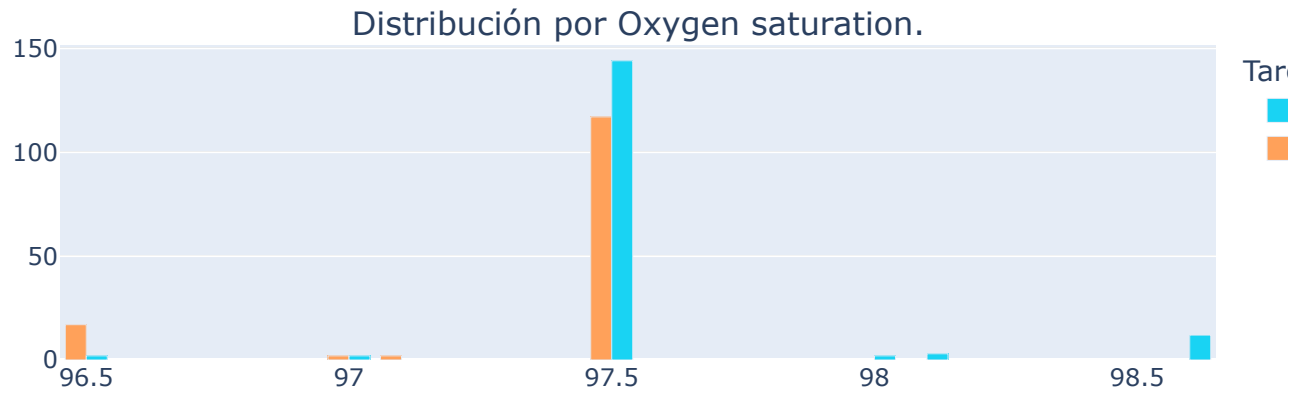
En este atributo hemos interpretado los valores según la información Thallium Stress Test Result., donde el valor 2 ('Low blood flow during both rest and exercise'), son los pacientes que tienen la probabilidad más alta de sufrir un infarto.

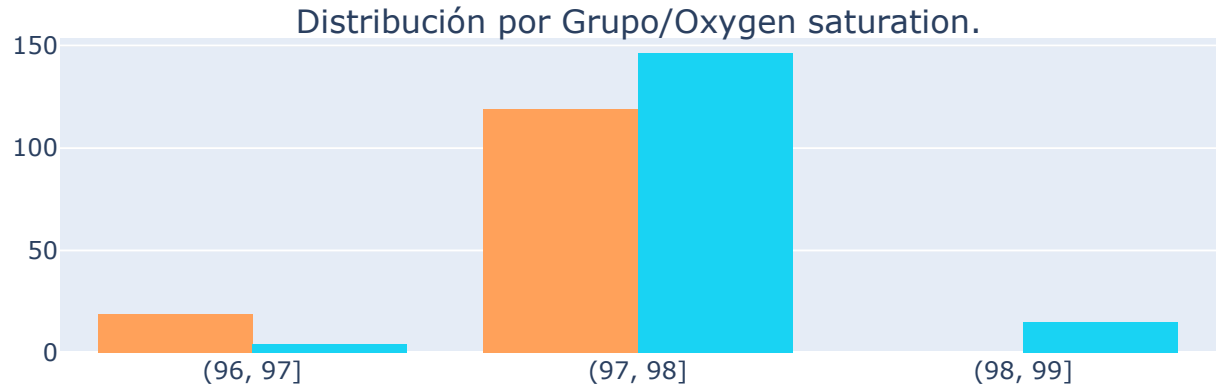
- 0 = 0.66%
- 1 = 5.94%
- 2 = 54.8%
- 3 = 38.6%

Oxygen saturation, (Sat_level):

In [44]:
```
df1=df.groupby(by=["Sat_level","target_descrip"]).size().reset_index(name="counts")
df1['grupo_Sat_level'] = pd.cut(df1['Sat_level'], bins=range(int(math.modf(df['Sat_level
#df1['grupo_oldpeak'].unique() #para añadir en el array ( a estudiar para hacerlo automá
```

In [45]:
```
#datos a pasar dataframe,atributo,leyendas,titulos)
atributo='Sat_level'
atributo2='grupo_Sat_level'
leyenda_p=" "
leyenda_s="Target"
titulo_p="Distribución por Oxygen saturation."
titulo_s="Distribución por Grupo/Oxygen saturation."
array=['(0, 1.0]','(0, 1.0]','(1.0, 2.0]','(2.0, 3.0]','(3.0, 4.0]','(4.0, 5.0]','(5.0,
#array=df1['grupo_trtbps'].unique()
X1, Y1 , X2, Y2 = 0.5,1,0.5,0.37
estudio_grupo(df1, atributo, atributo2, leyenda_p, leyenda_s, titulo_p, titulo_s, array,
```



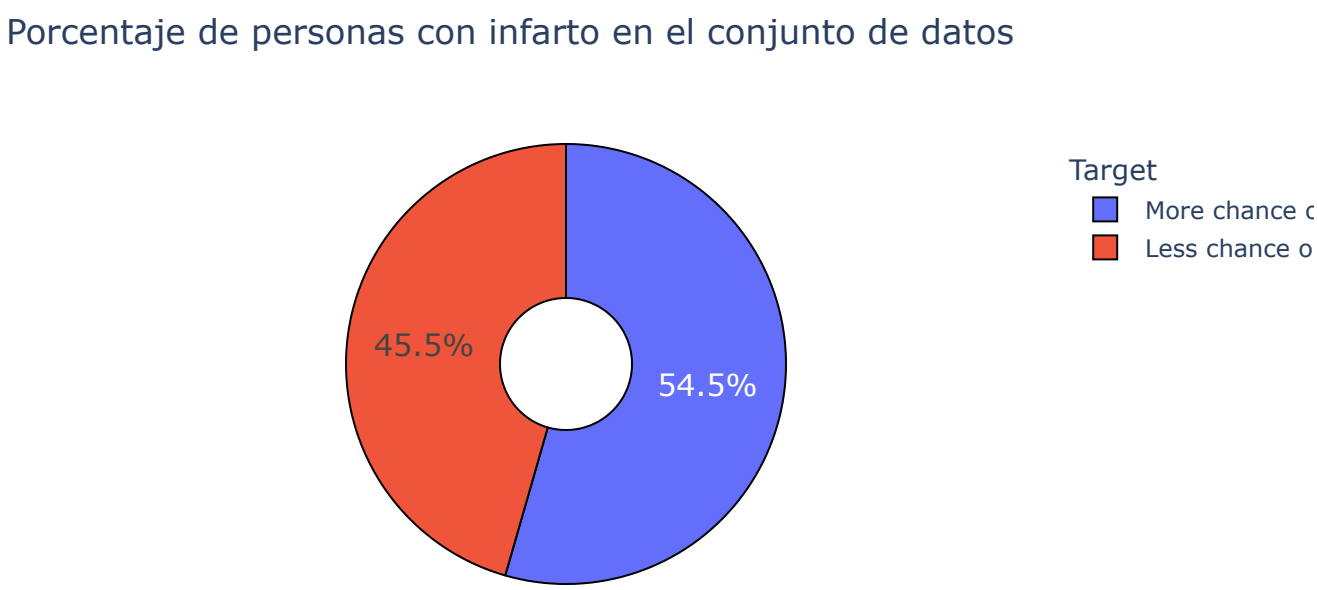


Tarjet:

Indica la probabilidad de un ataque al corazón.

- 0: less chance of heart attack.
- 1: more chance of heart attack.

In [46]:
```
fig = px.pie(df, names='target_descrip',title='Porcentaje de personas con infarto en el
fig.update_traces( textfont_size=15,  marker=dict(line=dict(color='#000000', width=1)))
fig.update_layout(height=400, width=800,legend_title_text='Target')
fig.show()
```



En este dataset según la prueba de esfuerzo, tenemos un 45,7% con menos probabilidad de infarto frente al 54,3% con más probabilidad.

1.4 Análisis Exploratorio de Datos

Se ha analizado un dataset (conjunto de datos) obtenido de la plataforma Kaggle. Este dataset contiene los registros médicos de pacientes que les han realizado una prueba de esfuerzo.

Tamaño y tipos de Datos.

In [47]:
```
print(Style.BRIGHT +'DataSet Heart:'+Style.RESET_ALL)
print('El DataSet Heart contiene:', heart.shape[0], 'filas y', heart.shape[1], 'columnas
print(heart.info())
```

```
DataSet Heart:
El DataSet Heart contiene: 303 filas y 15 columnas.

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 15 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   trtbps     303 non-null    int64  
 4   chol       303 non-null    int64  
 5   fbs        303 non-null    int64  
 6   restecg    303 non-null    int64  
 7   thalachh   303 non-null    int64  
 8   exng       303 non-null    int64  
 9   oldpeak    303 non-null    float64
```

```
 10  slp         303 non-null    int64
 11  caa         303 non-null    int64
 12  thall       303 non-null    int64
 13  Sat_level   303 non-null    float64
 14  target      303 non-null    int64
dtypes: float64(2), int64(13)
memory usage: 35.6 KB
None
```

Comprobamos nulos:

In [48]:
```
print(Style.BRIGHT +'DataSet Heart:'+Style.RESET_ALL)
display(heart.isnull().sum())
```

DataSet Heart:

```
age          0
sex          0
cp           0
trtbps       0
chol         0
fbs          0
restecg      0
thalachh     0
exng         0
oldpeak      0
slp          0
caa          0
thall        0
Sat_level    0
target       0
dtype: int64
```

Eliminamos valores duplicados.

In [49]:
```
print(Style.BRIGHT +'DataSet Heart:\n'+Style.RESET_ALL)
print(heart.duplicated().sum())
heart=heart.drop_duplicates()
print(heart.duplicated().sum())
```

DataSet Heart:

```
1
0
```

Estadística descriptiva del DataSet.

In [50]:
```
df_tabla=EstaDescrip(heart[list(heart.columns)])
df_tabla=df_tabla.reset_index()
df_tabla=df_tabla.rename(columns={'index':'Atributos'})
dibujartabla('\n\nEstadística descriptiva del DataSet Heart:\n',round(df_tabla,2))
```

Estadística descriptiva del DataSet Heart:

Atributos	count	mean	media	std	min	25%	50%	75%	max
age	302	54.42	55.5	9.05	29	48	55.5	61	77
sex	302	0.68	1	0.47	0	0	1	1	1
cp	302	0.96	1	1.03	0	0	1	2	3

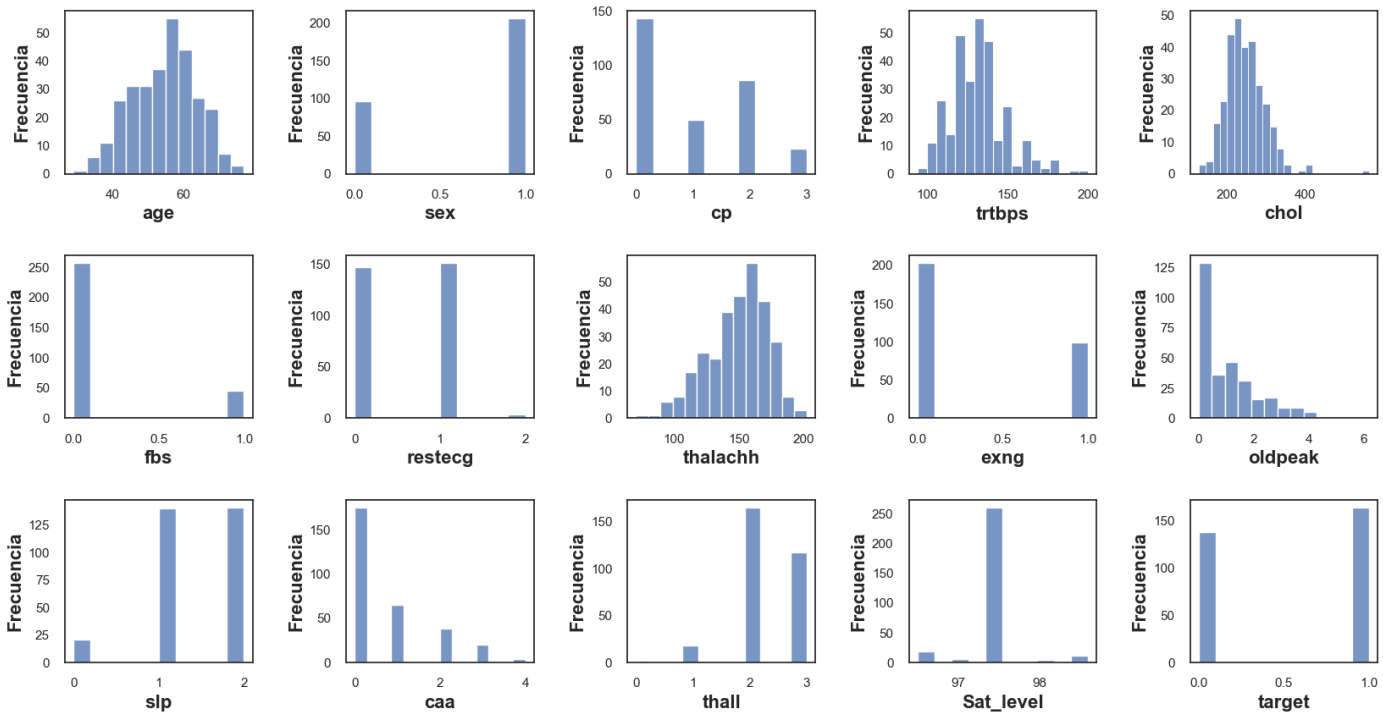
trtbps	302	131.6	130	17.56	94	120	130	140	200
chol	302	246.5	240.5	51.75	126	211	240.5	274.75	564
fbs	302	0.15	0	0.36	0	0	0	0	1
restecg	302	0.53	1	0.53	0	0	1	1	2
thalachh	302	149.57	152.5	22.9	71	133.25	152.5	166	202
exng	302	0.33	0	0.47	0	0	0	1	1
oldpeak	302	1.04	0.8	1.16	0	0	0.8	1.6	6.2

Prueba de Contraste de Normalidad.

- Gráfica de Histograma.
- Gráfico Quantile-Quantile.
- Prueba Shapiro-Wilks.

In [51]:
```
#print(Style.BRIGHT +'\nPruebas de Contraste de Normalidad:\n '+Style.RESET_ALL)
print(Style.BRIGHT+'\nDistribución de Atributos:\n '+Style.RESET_ALL)
crearhistograma(heart,list(heart.columns))
sns.set_theme(style="white", rc=None)
```

Distribución de Atributos:



Podemos ver la distribución de cada predictor, algunos de ellos parecen cercanos a una distribución normal, podemos comprobar si siguen una distribución normal utilizando la Prueba de Normalidad de Shapiro-Wilk. Nuestra Hipótesis es:

- H_0: Los datos se distribuyen normalmente.

- H_1: Los datos no se distribuyen normalmente.

Crearemos una función que compruebe la distribución para cada objetivo:

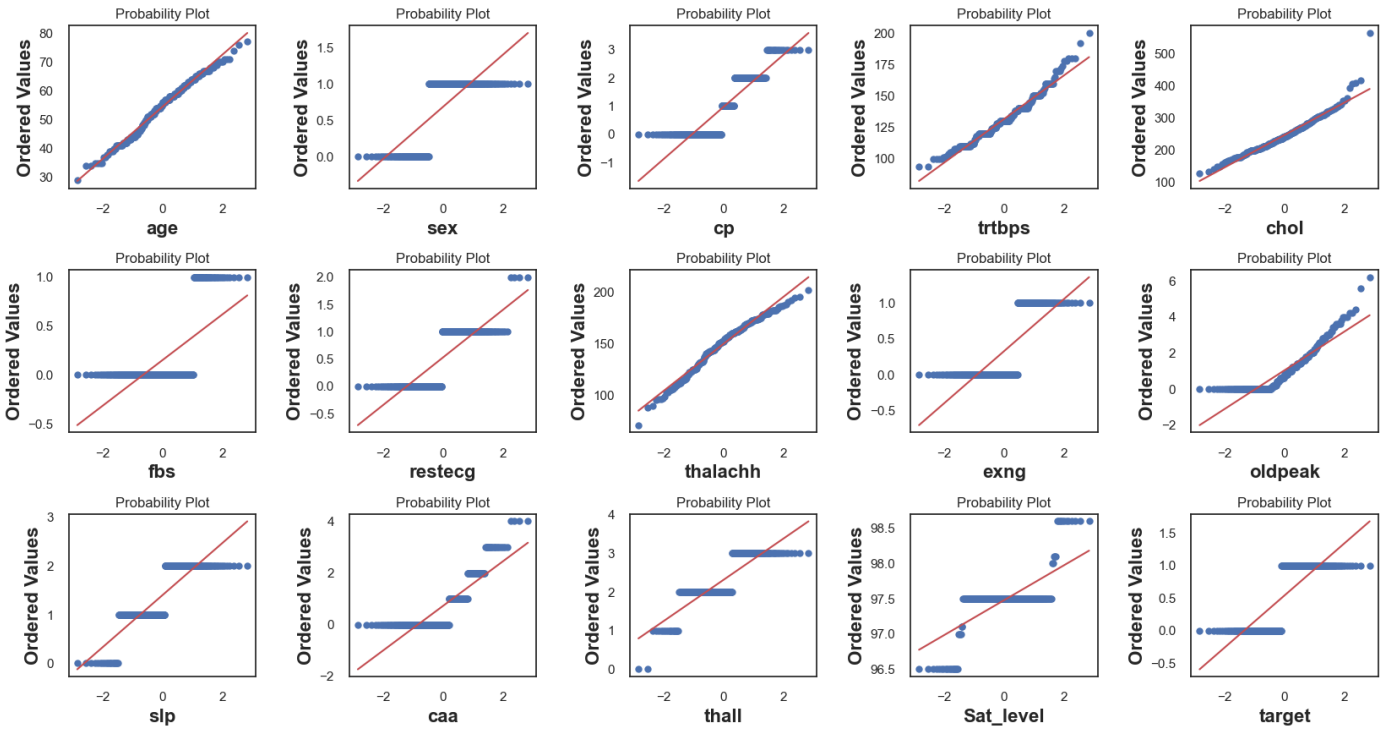
In [52]:
```
df_tabla=crearShapiro(heart,list(heart.columns))
#df_tabla=df_tabla.rename(columns={'index':'Atributos'})
dibujartabla('\n\nPrueba Shapiro-Wilk:\n',round(df_tabla,6))
```

Prueba Shapiro-Wilk:

Atributo	Stat	p-value	Resultado
age	0.986637	0.006744	Not Probably Gaussian
sex	0.586334	0	Not Probably Gaussian
cp	0.7897	0	Not Probably Gaussian
trtbps	0.965726	0.000001	Not Probably Gaussian
chol	0.94658	0	Not Probably Gaussian
fbs	0.424705	0	Not Probably Gaussian
restecg	0.679373	0	Not Probably Gaussian
thalachh	0.97679	0.000083	Not Probably Gaussian
exng	0.591848	0	Not Probably Gaussian
oldpeak	0.84522	0	Not Probably Gaussian

In [53]:
```
print(Style.BRIGHT +'\n\nGráfica Quantile-Quantile:\n '+Style.RESET_ALL)
crearquantile(heart,list(heart.columns))
sns.set_theme(style="white", rc=None)
```

Gráfica Quantile-Quantile:



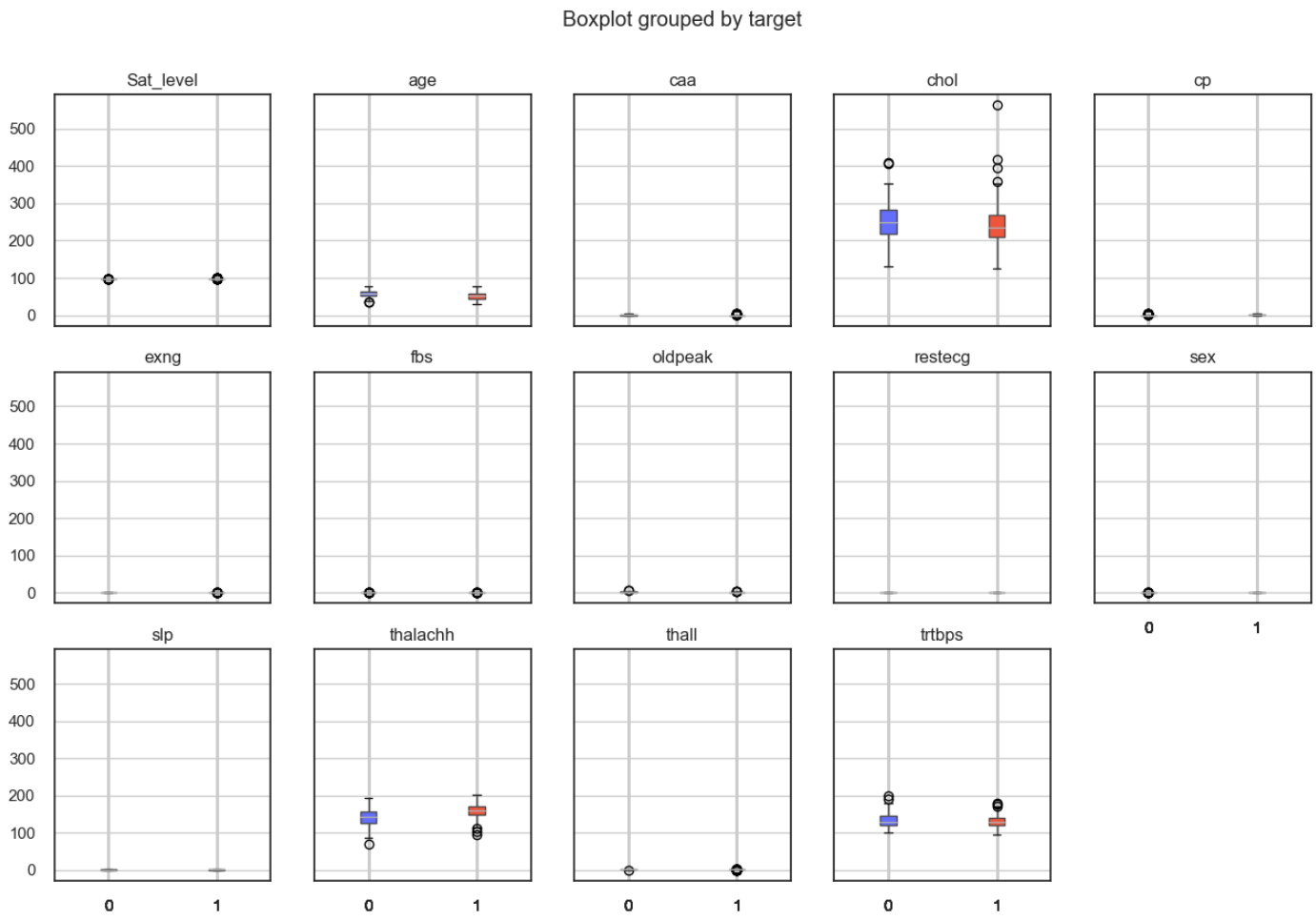
Observaciones:

- Las escalas de los atributos son diferentes, será necesario escalar las mismas.
- Las distribuciones no se distribuyen normalmente (es decir, Gaussian), `StandardScaler` no se debe utilizar para escalar los datos. Usaremos `MinMaxScaler` en su lugar.
- Tras la visualización de los atributos, tenemos atributos numéricos y categóricos.
- **Atributos discretos:** `sex` , `cp` , `fbs` , `restecg` , `exng` , `slp` , `caa` , `thall` , and `target` .
- **Atributos continuas:** `age` , `trtbps` , `chol` , `thalachh` , `oldpeak` , `Sat_level` .

Nota:

- Para ver otras características de los atributos que se nos hayan podido pasar por alto, también podemos hacer un gráfico boxplot, en este caso separaremos los boxplots para cada objetivo.

In [54]:
```
bp_dict = heart.boxplot(
by="target",layout=(3,5),figsize=(15,10),
return_type='both',
patch_artist = True,
)
#colors =["#9979c1", "#eba2ae", "#8dc146", '#ffa455', '#ffe358']

for row_key, (ax,row) in bp_dict.iteritems():
    ax.set_xlabel('')

    for i,box in enumerate(row['boxes']):
        box.set_facecolor(colors[i])

plt.show()
```

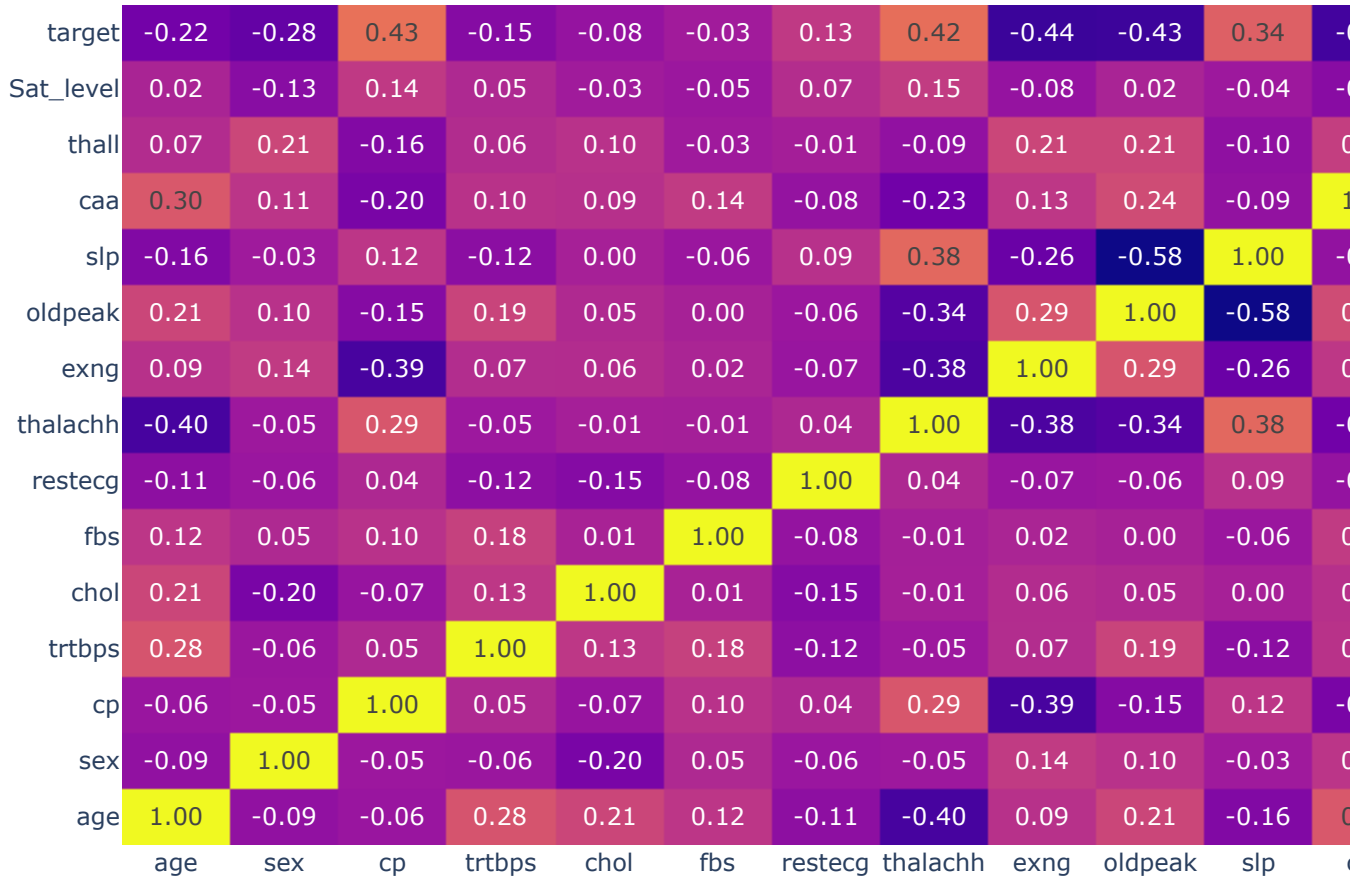


Podemos observar que tenemos algunos atributos con posibles outliers.

1.5 Matrix de correlación

La **matrix de correlación** muestra el valor de correlación de Pearson, que mide el grado de relacion lineal entre cada par de atributos. Esta matriz nos ayuda a realizar un análisis adicional de los datos ya que proporciona información relevante sobre las dependencias entre los diferentes atributos. El valor de correlación puede estar entre -1 y +1. Si estos dos elementos tienden a aumentar o disminuir al mismo tiempo, el valor de correlación es positivo porque indica una relación directa. Si el valor es negativo, esto indica una relación inversa. Un valor nulo o cercano a cero indica que no existe una tendencia entre ambos atributos.

In [55]:
```
df_corr = heart.corr()
fig = go.Figure()
fig.add_trace(
    go.Heatmap(
        x = df_corr.columns,
        y = df_corr.index,
        z = np.array(df_corr),
        text=df_corr.values,
        texttemplate='%{text:.2f}'
    )
)
fig.update_layout(title='Matrix de Correlación',height=600, width=1000,)

fig.show()
```



Ordenamos los valores de correlación de las parejas mediante el algoritmo de quicksort.

In [56]:
```
parejas_corr = df_corr.unstack()
orden_parejas = parejas_corr.sort_values(kind='quicksort')
#print(orden_parejas)
```

Nos quedamos únicamente con los atributos que más correlación directa o inversa tengan.

In [57]:
```
corr_parejas= orden_parejas[(orden_parejas<-0.5)|(orden_parejas>0.5)]#&((orden_parejas!=
corr_parejas=corr_parejas.reset_index(name='Correlación')
df_parejas=pd.DataFrame(corr_parejas)
df_parejas=df_parejas.rename(columns={'level_0':'Atributos_A','level_1':'Atributos_B'})
```

In [58]:
```
dibujartabla('\n\nCorrelación de Atributos +/- 0.5:\n',round(df_parejas,2))
```

Correlación de Atributos +/- 0.5:

Atributos_A	Atributos_B	Correlación
slp	oldpeak	-0.58
oldpeak	slp	-0.58
age	age	1

thalachh	thalachh	1
thall	thall	1
caa	caa	1
slp	slp	1
oldpeak	oldpeak	1
exng	exng	1
restecg	restecg	1

Las correlaciones mayores de 0.7 que se muestran son las correlaciones perfectas que se producen entre los atributos consigo mismos. Las demas correlaciones no son superiores o inferiores a +/- 0.7, por lo que ninguna correlación es lo sufiecientemente relevante.

Volver Índice general

2. Preparación del DataSet

En este capitulo vamos a someter a los atributo a un análisis de los componetes, un prepocesado, a decidir nuestro modelo, comprobar si nuestro **target** está balanceado, comprobaremos el **overfitting y underfitting**, y las métricas que nos permitan necesarias para comprobar el comportamiento del modelo.

2.1 Análisis de los Componentes Principales (PCA)

El algoritmo de Análisis de Componente Principal (PCA), es una técnica de aprendizaje no supervisado que transforma un gran conjunto de atributos en uno más pequeño que aun contiene la mayor parte de la información del conjunto original. Al ser un conjunto de datos más pequeño, facilita el análisis de los mismos y es más rapido para los algoritmos de Machine learning. En resumen, la idea de PCA es reducir la cantidad de atributos del conjunto de datos, mientras conserva la mayor cantidad de información.

Estandarizamos los atributos.

Nos aseguramos que los atributos no se califiquen con más importancia debido a su diferencia de escala. Todo el conjuntos de datos será estandarizado, transformando a una distribución normal con la media centrada a **0** y la desviación tipica a **1**.

In [59]:
```
index=['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh','exng', 'oldpe
df_mean = pd.DataFrame(heart.mean(axis=0),columns=['mean'],index=index)
df_std = pd.DataFrame(heart.std(axis=0),columns=['std'],index=index)
```

In [60]:
```
#Preparamos los datos para estandarizar
x = heart.drop(columns =['target'])
x= pd.DataFrame(StandardScaler().fit_transform(x))
x.columns=index
```

In [61]:
```
#index={'0':'age','1':'sex','2':'cp','3':'trtbps','4':'chol','5':'fbs','6':'restecg','7'
df_mean_esc = pd.DataFrame(x.mean(axis=0),columns=['mean_scaler'],index=index)
df_std_esc = pd.DataFrame(x.std(axis=0),columns=['std_scaler'],index=index)
```

In [62]:
```
df_estan = pd.concat([df_mean,df_mean_esc,df_std,df_std_esc],axis=1)
dibujartabla('\n\nValores de la Media/Desviación Típica Antes/Despues de estandarización
```

ores de la Media/Desviación Típica Antes/Despues de estandarizac

mean	mean_scal	std	std_scaler
54.42053	0	9.04797	1.00166
0.682119	0	0.466426	1.00166
0.963576	0	1.032044	1.00166
131.602649	0	17.563394	1.00166
246.5	0	51.753489	1.00166
0.149007	0	0.356686	1.00166
0.52649	0	0.526027	1.00166
149.569536	0	22.903527	1.00166
0.327815	0	0.470196	1.00166
1.043046	0	1.161452	1.00166

La tabla anterior muestra como la estandarización a centrado la media a **0** y la desviación típica a **1**.

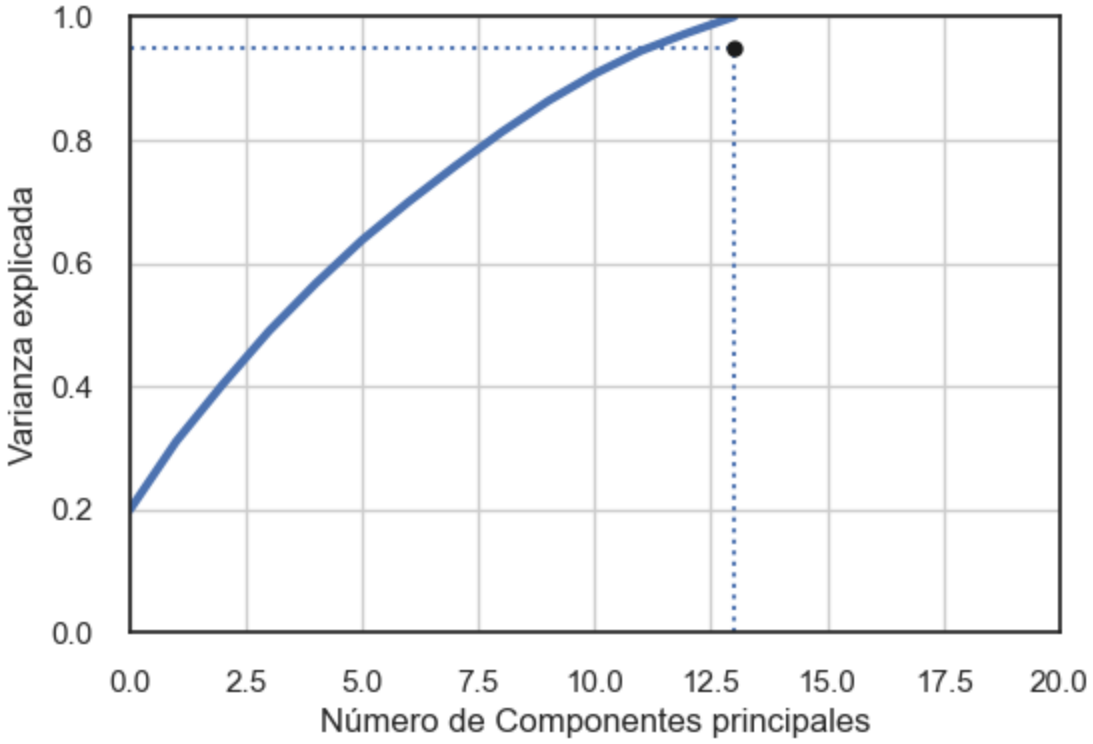
Aplicamos PCA.

Los componentes son nuevas variables que se construyen como combinaciones lineales o mezclas de las variables iniciales, evitando que estas nuevas variables no esten correlacionadas y la mayor parte de la información está dentro de las variables iniciales se comprime en los primeros componentes.

El objetivo es eligir los suficientes atributos para que expliquen una varianza superior a un **90%**.

In [63]:
```
# Creamos el PCA.
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

#Preparamos x e y
x = heart.loc[:,heart.columns != 'target'].values
y = heart.loc[:,['target']].values

#Escalamos los datos
x= pd.DataFrame(StandardScaler().fit_transform(x))
y=pd.DataFrame(y)
```

```
pca = PCA(n_components=14).fit(x)

plt.figure(figsize=(6,4))
#Run PCA.

x=np.arange(1,15,step=1)
y=np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(y >= 0.95) + 1

#plt.plot(x,y)
plt.plot(y, linewidth=3)
plt.axis([0, 20, 0, 1])
plt.xlabel('Número de Componentes principales')
plt.ylabel('Varianza explicada')
plt.plot([d, d], [0, 0.95], "b:")
plt.plot([0, d], [0.95, 0.95], "b:")
plt.plot(d, 0.95, "ko")
plt.grid(True)
plt.show()
```



Como podemos observar en el diagrama obtenido, con 1 componente principal obtenemos el 20% de la información y para explicar el 90% de la varianza, o lo que es lo mismo, para obtener el 90% de la información, es necesario utilizar **todas los componentes principales en nuestro caso 14 atributos**.

2.2 Pre-procesamiento de los datos

Explicación de como vamos a pre-procesar los datos:

Los datos continuos los sometemos a un proceso de estandarización o un proceso de escalado para limitarlos a un rango de valores que les permita ser comparados entre sí, independientemente de su naturaleza. Los datos discretos permanecerán igual.

In [64]:
```
heart.columns
```

Out[64]:
```
Index(['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh',
       'exng', 'oldpeak', 'slp', 'caa', 'thall', 'Sat_level', 'target'],
      dtype='object')
```

Indice de los atributos:

- Atributos con datos continuos: ['age[0]','trtbps[3]','chol[4]','thalach[7]','oldpeak[9]','Sat_level[13]']
- Atributos con datos discretos: ['sex[1]','cp[2]','fbs[5]','restecg[6]','exng[8]','slp[10]','caa[11]','thall[12]']

Los Atributos con datos discretos no vamos a realizar ninguna transformación. Los atributo **age[0]** y **Sat_level[13]**, tienen una desviación típica pequeña utilizaremos MinMaxScaler. El atributo oldpeak tiene los datos agrupados más en un extremo, aplicamos StandardScaler y el resto contienen outliers donde aplicaremos RobustScaler.

Estandarización de los datos.

Definimos la transformación, haciendo uso de la función 'ColumnTransformer' de Sklearn. Indicando para cada columna que transformación vamos a aplicar.

In [65]:
```
columnas_Transformar = ['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalach
       'exng', 'oldpeak', 'slp', 'caa', 'thall','Sat_level','target']
column_transformer= sklearn.compose.ColumnTransformer([
    ("min-max_1", sklearn.preprocessing.MinMaxScaler(),[0]),
    ("passthrough1", 'passthrough',[1]),
    ("passthrough2", 'passthrough',[2]),
    ("scale1", sklearn.preprocessing.RobustScaler(),[3]),
    ("scale2", sklearn.preprocessing.RobustScaler(),[4]),
    ("passthrough3", 'passthrough',[5]),
    ("passthrough4", 'passthrough',[6]),
    ("scale3", sklearn.preprocessing.RobustScaler(),[7]),
    ("passthrough5", 'passthrough',[8]),
    ("scale4", sklearn.preprocessing.StandardScaler(),[9]),
    ("passthrough6", 'passthrough',[10]),
    ("passthrough7", 'passthrough',[11]),
    ("passthrough8", 'passthrough',[12]),
    ("min-max_2", sklearn.preprocessing.MinMaxScaler(),[13]),
    ("passthrough9", 'passthrough',[14]),
])
```

In [66]:
```
x_scaled = column_transformer.fit_transform(heart)
df_scaled=pd.DataFrame(x_scaled,columns=columnas_Transformar)
dibujartabla('\n\nDatos de la estandarización:\n',round(df_scaled[0:5],6))
```

Datos de la estandarización:

age	se	cp	trtb	chol	fbs	res	thalac	ex	oldpea	slp	ca	tha	Sat_l	tar
0.708	1	3	0.7	-0.117	1	0	-0.076	0	1.0840	0	0	1	1	1
0.166	1	2	0	0.1490	0	1	1.0534	0	2.1189	0	0	2	1	1
0.25	0	1	0	-0.572	0	0	0.5954	0	0.3078	2	0	2	1	1
0.562	1	1	-0.5	-0.070	0	1	0.7786	0	-0.209	2	0	2	0.761	1
0.583	0	0	-0.5	1.7803	0	1	0.3206	1	-0.382	2	0	2	0.476	1

2.3 Contruimos nuestro modelo

Utilizaremos la librería Pycaret para hacer una comparación entre diferentes modelos y elegir el que nos dé un mejor rendimiento. Como hemos comprobado anteriormente nuestro predictores no siguen una distribución normal, por lo que introduciremos el parámetro **normalize = true** para el dataframe sin estandarización **(heart)**, y lo compararemos con el estandarizado **(df_scaled)**.

In [67]:
```
from pycaret.classification import *

# use normalize=True to evaluate model selection with normalized data
clf = setup(heart, target='target', session_id=42, normalize = True)
```

	Description	Value
0	Session id	42
1	Target	target
2	Target type	Binary
3	Original data shape	(302, 15)
4	Transformed data shape	(302, 15)
5	Transformed train set shape	(211, 15)
6	Transformed test set shape	(91, 15)
7	Numeric features	14
8	Preprocess	True
9	Imputation type	simple
10	Numeric imputation	mean
11	Categorical imputation	mode
12	Normalize	True
13	Normalize method	zscore
14	Fold Generator	StratifiedKFold
15	Fold Number	10
16	CPU Jobs	-1
17	Use GPU	False
18	Log Experiment	False
19	Experiment Name	clf-default-name
20	USI	144c

In [68]:
```
# print the result for best models
best= compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
ridge	Ridge Classifier	0.8671	0.0000	0.9136	0.8570	0.8802	0.7298	0.7416	0.0710
lda	Linear Discriminant Analysis	0.8671	0.9151	0.9136	0.8570	0.8802	0.7298	0.7416	0.0720
knn	K Neighbors Classifier	0.8626	0.8980	0.9409	0.8425	0.8841	0.7167	0.7361	0.0730
lr	Logistic Regression	0.8576	0.9123	0.8962	0.8548	0.8712	0.7108	0.7199	0.4060
rf	Random Forest Classifier	0.8485	0.9081	0.8886	0.8510	0.8653	0.6906	0.7001	0.0900
et	Extra Trees Classifier	0.8392	0.9080	0.8636	0.8501	0.8534	0.6758	0.6817	0.0890
gbc	Gradient Boosting Classifier	0.8297	0.8902	0.8803	0.8246	0.8483	0.6536	0.6618	0.0790
xgboost	Extreme Gradient Boosting	0.8247	0.8995	0.8705	0.8243	0.8446	0.6432	0.6488	0.0770
lightgbm	Light Gradient Boosting Machine	0.8201	0.8894	0.8621	0.8229	0.8402	0.6334	0.6381	0.0780
ada	Ada Boost Classifier	0.8147	0.8452	0.8598	0.8135	0.8305	0.6234	0.6369	0.0790
nb	Naive Bayes	0.8106	0.8965	0.8530	0.8122	0.8286	0.6153	0.6236	0.0720
qda	Quadratic Discriminant Analysis	0.7868	0.8713	0.7833	0.8321	0.7943	0.5725	0.5917	0.0720
svm	SVM - Linear Kernel	0.7866	0.0000	0.7788	0.8323	0.7986	0.5725	0.5812	0.0700
dt	Decision Tree Classifier	0.7342	0.7356	0.7235	0.7714	0.7421	0.4710	0.4743	0.0720
dummy	Dummy Classifier	0.5450	0.5000	1.0000	0.5450	0.7052	0.0000	0.0000	0.0740

In [69]:
```
clf = setup(df_scaled, target='target', session_id=42)
```

	Description	Value
0	Session id	42
1	Target	target
2	Target type	Binary
3	Original data shape	(302, 15)
4	Transformed data shape	(302, 15)
5	Transformed train set shape	(211, 15)
6	Transformed test set shape	(91, 15)
7	Numeric features	14
8	Preprocess	True
9	Imputation type	simple
10	Numeric imputation	mean
11	Categorical imputation	mode
12	Fold Generator	StratifiedKFold
13	Fold Number	10
14	CPU Jobs	-1
15	Use GPU	False

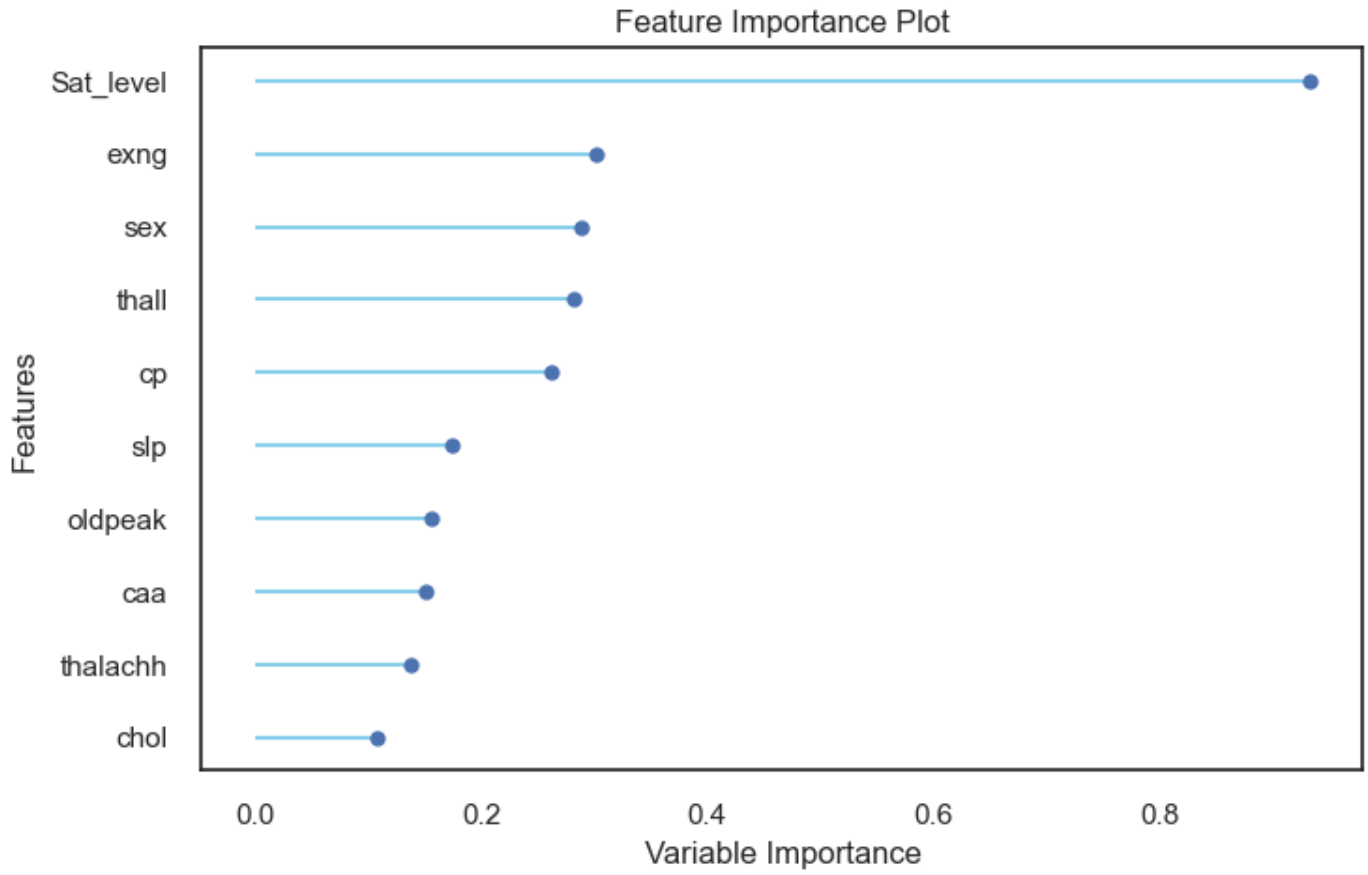
16	Log Experiment	False
17	Experiment Name	clf-default-name
18	USI	0cd7

In [70]:
```
# print the result for best models
best= compare_models()
best
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
ridge	Ridge Classifier	0.8671	0.0000	0.9136	0.8570	0.8802	0.7298	0.7416	0.0680
lda	Linear Discriminant Analysis	0.8671	0.9151	0.9136	0.8570	0.8802	0.7298	0.7416	0.0680
lr	Logistic Regression	0.8626	0.9152	0.9136	0.8506	0.8772	0.7198	0.7321	0.0730
rf	Random Forest Classifier	0.8485	0.9080	0.8970	0.8449	0.8670	0.6895	0.6989	0.0840
et	Extra Trees Classifier	0.8392	0.9080	0.8636	0.8501	0.8534	0.6758	0.6817	0.0860
gbc	Gradient Boosting Classifier	0.8297	0.8912	0.8803	0.8246	0.8483	0.6536	0.6618	0.0780
knn	K Neighbors Classifier	0.8247	0.8838	0.9061	0.7990	0.8462	0.6429	0.6588	0.0730
xgboost	Extreme Gradient Boosting	0.8247	0.8995	0.8705	0.8243	0.8446	0.6432	0.6488	0.0710
lightgbm	Light Gradient Boosting Machine	0.8247	0.8912	0.8614	0.8304	0.8433	0.6429	0.6481	0.0710
svm	SVM - Linear Kernel	0.8195	0.0000	0.9136	0.8021	0.8471	0.6283	0.6577	0.0670
ada	Ada Boost Classifier	0.8195	0.8461	0.8598	0.8217	0.8344	0.6331	0.6469	0.0790
nb	Naive Bayes	0.8106	0.8965	0.8530	0.8122	0.8286	0.6153	0.6236	0.0690
qda	Quadratic Discriminant Analysis	0.7868	0.8713	0.7833	0.8321	0.7943	0.5725	0.5917	0.0680
dt	Decision Tree Classifier	0.7294	0.7311	0.7144	0.7703	0.7364	0.4617	0.4658	0.0690
dummy	Dummy Classifier	0.5450	0.5000	1.0000	0.5450	0.7052	0.0000	0.0000	0.0700

Out[70]:
```
RidgeClassifier
RidgeClassifier(alpha=1.0, class_weight=None, copy_X=True, fit_intercept=True,
                max_iter=None, positive=False, random_state=42, solver='auto',
                tol=0.0001)
```

In [71]:
```
plot_model(best, plot='feature')
```



In [72]:
```
# feature.importances
model = create_model('ridge')

# Obtener las importancias de las características
#importances = feature_importances(model, X=heart.drop('target', axis=1), y=data['target

# Imprimir las importancias de las características
print(model)
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.9091	0.0000	1.0000	0.8571	0.9231	0.8136	0.8281
1	0.8571	0.0000	1.0000	0.8000	0.8889	0.6957	0.7303
2	0.9048	0.0000	1.0000	0.8571	0.9231	0.8000	0.8165
3	0.7619	0.0000	0.8333	0.7692	0.8000	0.5070	0.5095
4	0.7619	0.0000	0.6667	0.8889	0.7619	0.5333	0.5556
5	0.8095	0.0000	0.8182	0.8182	0.8182	0.6182	0.6182
6	0.8571	0.0000	0.8182	0.9000	0.8571	0.7149	0.7182
7	0.9524	0.0000	1.0000	0.9167	0.9565	0.9041	0.9083
8	0.9524	0.0000	1.0000	0.9167	0.9565	0.9041	0.9083
9	0.9048	0.0000	1.0000	0.8462	0.9167	0.8073	0.8228
Mean	0.8671	0.0000	0.9136	0.8570	0.8802	0.7298	0.7416
Std	0.0669	0.0000	0.1142	0.0475	0.0644	0.1345	0.1340

```
RidgeClassifier(alpha=1.0, class_weight=None, copy_X=True, fit_intercept=True,
```

```
max_iter=None, positive=False, random_state=42, solver='auto',
tol=0.0001)
```

Realizaremos el ejercicio con el estimador **RidgeClassifier**, es uno de los tres mejores en puntuación después de realizar la comparativa de modelos.

2.4 División de los datos en train y test

En está sección se explica de que forma se va a dividir el conjunto de datos, qué es el sobreajuste (overfitting) y como evitarlo.

In [73]:
```
#Utilizamos el df estandarizado.


# define predictors
X = df_scaled.drop(['target'], axis = 1)
# define target
y = df_scaled['target']

print('df_scaled.shape: ', df_scaled.shape)
print('y.shape: ', y.shape)
print('X.shape', X.shape)
```

```
df_scaled.shape:  (302, 15)
y.shape:  (302,)
X.shape (302, 14)
```

In [74]:
```
# choose test size of 20%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state= 4
```

In [75]:
```
print('X_train.shape: ', X_train.shape)
print('X_test.shape: ', X_test.shape)
print('y_train.shape: ', y_train.shape)
print('y_test.shape: ', y_test.shape)
```

```
X_train.shape:  (241, 14)
X_test.shape:  (61, 14)
y_train.shape:  (241,)
y_test.shape:  (61,)
```

2.5 Balanceado de clases

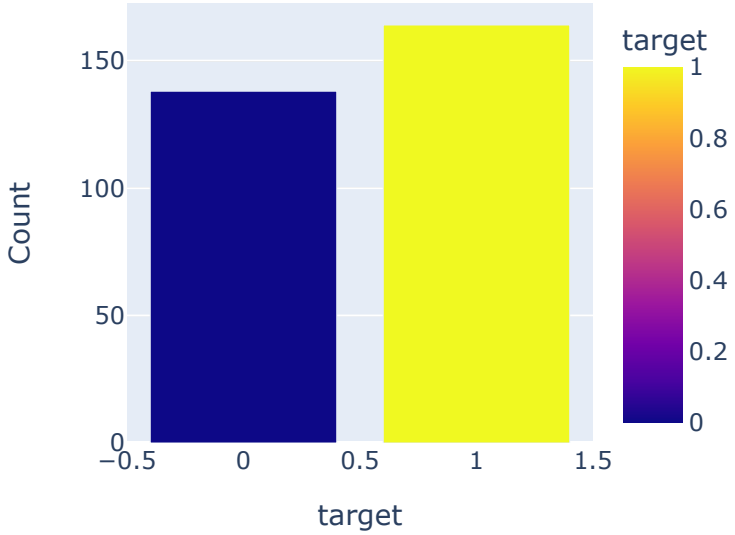
Aunque en nuestro conjunto de datos las muestras que tenemos son pequeñas, en el ambito de la salud debemos ser cocientes que podemos encontrar conjuntos de datos con miles de registros con pacientes **negativos** y unos pocos casos **positivos** es decir, que padecen la enfermedad que queremos clasificar.
A continuación vamos a estudiar el balanceo de las dos clases que tenemos en nuestro **target**, tanto en el dataset **df_scaled** como en **y_train**.
En la Gráfica 1, tenemos el balanceo del DataSet y en la gráfica 2 tenemos los datos después de realizar la división en train y test.

In [76]:
```
print(df_scaled['target'].value_counts())
```

```
1.0    164
0.0    138
Name: target, dtype: int64
```

In [77]:
```
# Calcular el recuento de cada atributo
count_data = df_scaled['target'].value_counts().reset_index()
count_data.columns = ['target', 'Count']
```

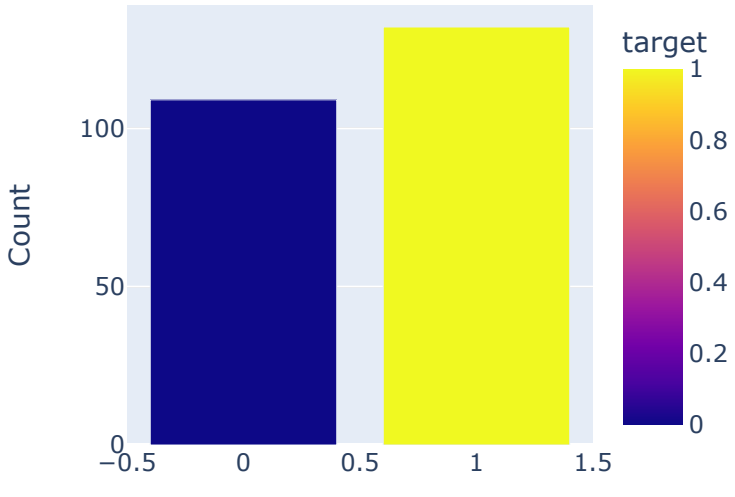
```
fig = px.bar(count_data, x="target", y='Count', title="Balanceado de clases. Gráfica 1"
fig.show()
```



In [78]:
```
# Balanceado de clases de y_train
unique, counts = np.unique(y_train, return_counts=True)

dict(zip(unique, counts))
```

Out[78]:
```
{0.0: 109, 1.0: 132}
```

In [79]:
```
df_ytrain = pd.DataFrame(y_train,columns=['target'])
# Calcular el recuento de cada atributo
count_data = df_ytrain['target'].value_counts().reset_index()
count_data.columns = ['target', 'Count']

fig = px.bar(count_data, x="target", y='Count', title="Balanceado de clases. Gráfica 2",
fig.show()
```



target

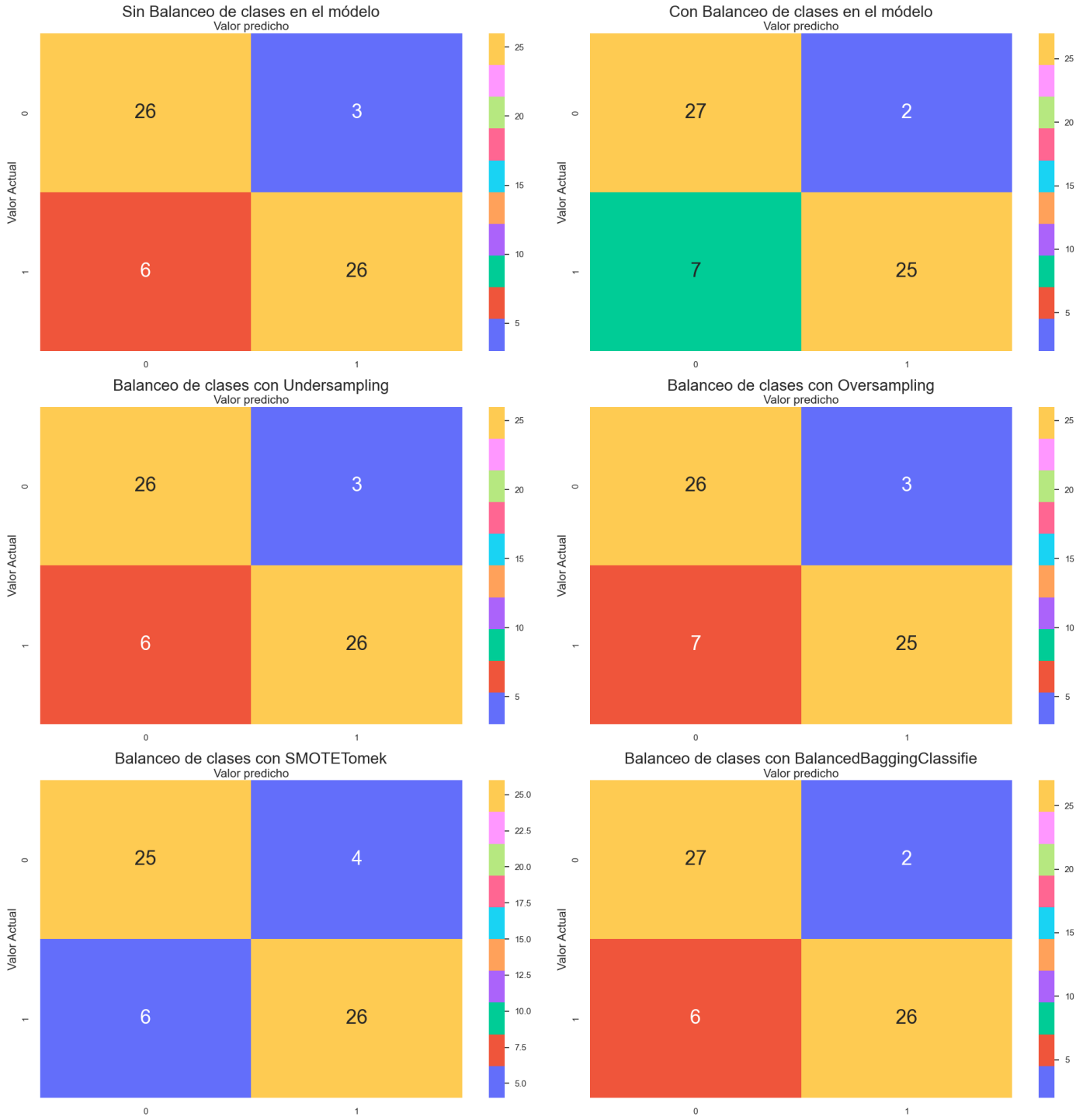
Realizamos un estudio del balanceo utilizando una prueba con los siguientes balanceadores sobre el estimador que hemos elegido para nuestro ejercicio. Donde obtendremos unas métricas que nos permitiran elegir aquel que mejores resultados obtengamos.

In [80]:
```
#class_weight = {None,'Balanced',None,None,None,None}


balanceo = {
    "None": {
        "titulo": "Sín Balanceo de clases en el módelo",
        "nomenclatura": "sbal",
        "balanceador": None
    },
    "Balanced": {
        "titulo": "'Con Balanceo de clases en el módelo'",
        "nomenclatura": "cbal",
        "balanceador": 'Balanced'
    },
    "Undersampling": {
        "titulo": "Balanceo de clases con Undersampling",
        "nomenclatura": "us",
        "balanceador": RandomUnderSampler(random_state=42)
    },
    "Oversampling": {
        "titulo": "Balanceo de clases con Oversampling",
        "nomenclatura": "os",
        "balanceador": RandomOverSampler(random_state=42)
    },
    "SMOTETomek": {
        "titulo": "Balanceo de clases con Smote-Tomek",
        "nomenclatura": "smote",
        "balanceador": SMOTETomek(random_state=42)
    },
    "BalancedBaggingClassifie": {
        "titulo": "Balanceo de clases con BalancedBaggingClassifier",
        "nomenclatura": "bagging",
        "balanceador": BalancedBaggingClassifier(base_estimator=RidgeClassifier(),
                                sampling_strategy='auto',
                                replacement=False,
                                random_state=42)
    }
}
```

In [81]:
```
Vprediciones2 =[]
df_resultados = probar_balanceadores(balanceo, X_train, X_test, y_train, y_test)
```

In [82]:
```
Vtitulo2=df_resultados.index.get_level_values('Estimador').unique()

Setconfusion_matrix(Vtitulo2,y_test,Vprediciones2)
```

```
Matriz de Confusión:
```



Las dos primeras gráficas nos muestran los datos obtenidos con el estimador **Ridge Classifier** utilizando el parámetro **class_weight** a **None** o **Balanced**. Las siguientes gráficas utilizamos 4 técnicas para mejorar el balanceo. Como observamos la clase minoritaria 0 conseguimos mejorarla con el último método **BalancedBaggindClassifie**, la clase mayoritaria permanece igual, ningún método a mejorado la original (sin balancear). Interpretación de la matriz de confusión **BalancedBaggindClassifie**:

- Si nos fijamos en la target 0, en nuestro conjunto de pruebas, hay 29 observaciones que miden el objetivo 0 ('LESS CHANCE OF HEART ATTACK'). 27 de estas observaciones han sido predichas correctamente y 2 han obtenido predicciones erróneas.

- Si nos fijamos en la target 1, en nuestro conjunto de pruebas, hay 32 observaciones que miden el objetivo 1 ('MORE CHANCE OF HEART ATTACK'). 26 de estas observaciones han sido predichas correctamente y 6 han obtenido predicciones erróneas.

A continuación tenemos las métricas obtenidas con **classification_report** y el coeficiente de correlación de

Matthews(MCC).

In [83]:
```
df= df_resultados.query("Valor != '' ").sort_values(by=['Valor'],ascending=False,)
df=df.reset_index()
titulo = 'Métricas Matthews correlation coefficient (MCC)'
dibujartabla(titulo,df[['Estimador','Métrica','Valor']])
```

Métricas Matthews correlation coefficient (MCC)

Estimador	Métrica	Valor
BalancedBaggingClassifie	Matthews_corrcoef	0.75
Balanced	Matthews_corrcoef	0.72
None	Matthews_corrcoef	0.71
Undersampling	Matthews_corrcoef	0.71
Oversampling	Matthews_corrcoef	0.68
SMOTETomek	Matthews_corrcoef	0.67

In [84]:
```
df= df_resultados.query("Métrica == 'accuracy'").sort_values(by=['precision'],ascending=
df=df.reset_index()
titulo = 'Métricas accuracy'
dibujartabla(titulo,df[['Estimador','Métrica','precision','recall','f1-score','support']
```

Métricas accuracy

Estimador	Métrica	precisi	recall	f1-sco	suppor
BalancedBaggingClassifie	accuracy	0.87	0.87	0.87	0.87
None	accuracy	0.85	0.85	0.85	0.85
Balanced	accuracy	0.85	0.85	0.85	0.85
Undersampling	accuracy	0.85	0.85	0.85	0.85
Oversampling	accuracy	0.84	0.84	0.84	0.84
SMOTETomek	accuracy	0.84	0.84	0.84	0.84

In [85]:
```
df= df_resultados.query("Métrica in ['Less chance H.A','More chance H.A']").sort_values(
df=df.reset_index()
titulo = 'Métricas Matrix de Correlación'
dibujartabla(titulo,df[['Estimador','Métrica','precision','recall','f1-score','support']
```

Métricas Matrix de Correlación

Estimador	Métrica	precis	recall	f1-sc	suppo
BalancedBaggingClassifie	Less chance H.A	0.82	0.93	0.87	29
BalancedBaggingClassifie	More chance H.A	0.93	0.81	0.87	32
Balanced	Less chance H.A	0.79	0.93	0.86	29
None	Less chance H.A	0.81	0.9	0.85	29
None	More chance H.A	0.9	0.81	0.85	32
Balanced	More chance H.A	0.93	0.78	0.85	32
Undersampling	Less chance H.A	0.81	0.9	0.85	29
Undersampling	More chance H.A	0.9	0.81	0.85	32
Oversampling	Less chance H.A	0.79	0.9	0.84	29
SMOTETomek	More chance H.A	0.87	0.81	0.84	32

In [86]:
```
# choose test size of 20%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state= 4
```

- Base del modelo elegido **RidgeClassifier**:

In [87]:
```
# Instanciar el módelo
#modelo = LogisticRegression(random_state=42)
modelo = RidgeClassifier()
```

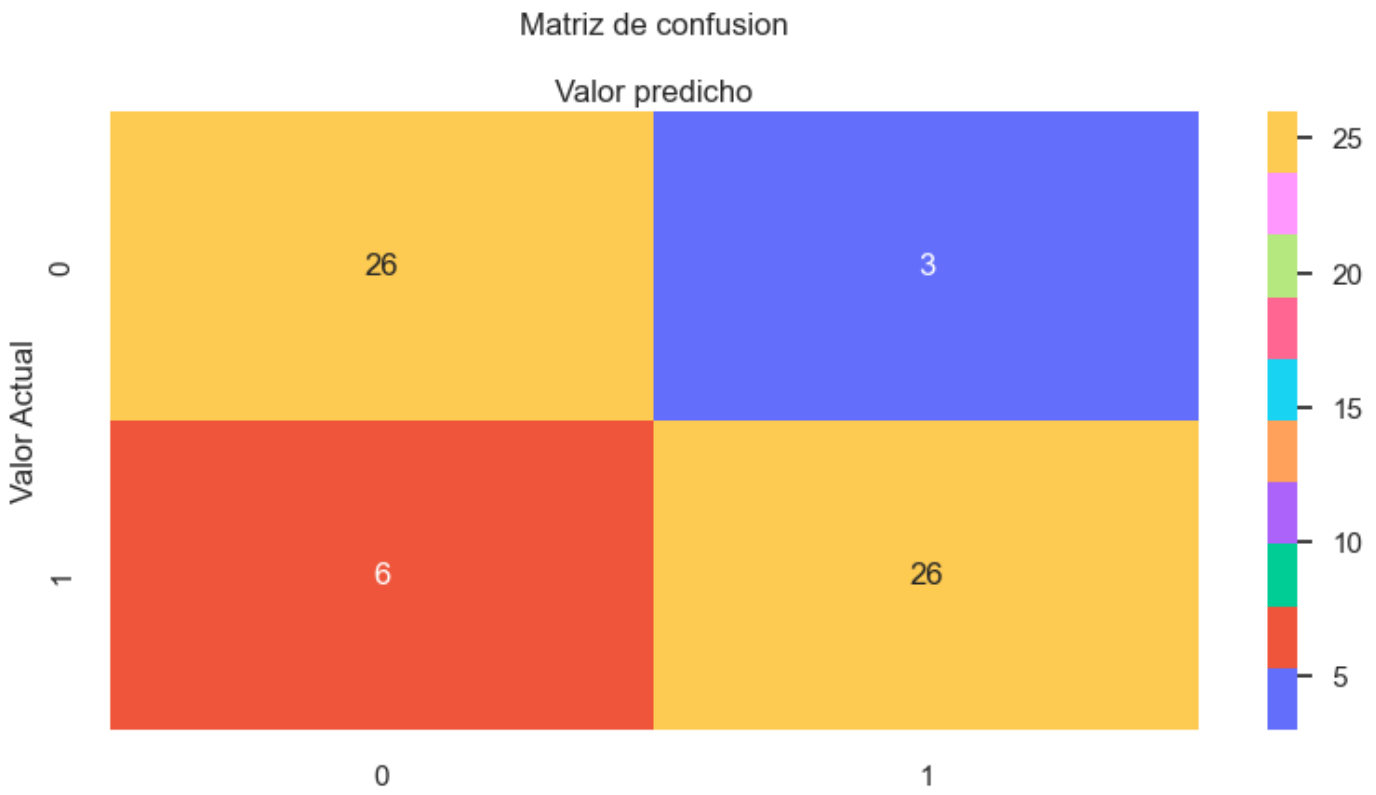
```
# entrenamos el módelo
modelo.fit(X_train, y_train)

# predictions on test set with our trained model
modelo_predictions = modelo.predict(X_test)

#utilizamos nuestro balanceador elegido anteriormente

#modelobbc = BalancedBaggingClassifier(base_estimator=modelo)
# entrenamos el módelo
#modelobbc.fit(X_train, y_train)

# predictions on test set with our trained model
#modelobbc_predictions = modelobbc.predict(X_test)
```

In [88]:
```
print('RidgeClassifier f1-score en test: {:.4f}'.format(f1_score(y_test, modelo_predicti
```

```
RidgeClassifier f1-score en test: 0.8525
```

In [89]:
```
grafica_matrix (y_test,modelo_predictions)
```

```
Matriz de Confusión:
```



2.6 Comprobación Sobreajuste o Subajuste del modelo, (Overfitting, Underfitting)

In [90]:
```
#<https://rubialesalberto.medium.com/regresi%C3%B3n-log%C3%ADstica-con-sklearn-4384c7070

sobreajuste = []
#modelo = LogisticRegression(random_state=42)
#modelobbc = BalancedBaggingClassifier(base_estimator=modelo)
#modelobbc.fit(X_train, y_train)
y_train_pred = modelo.predict(X_train)
y_test_pred = modelo.predict(X_test)
```

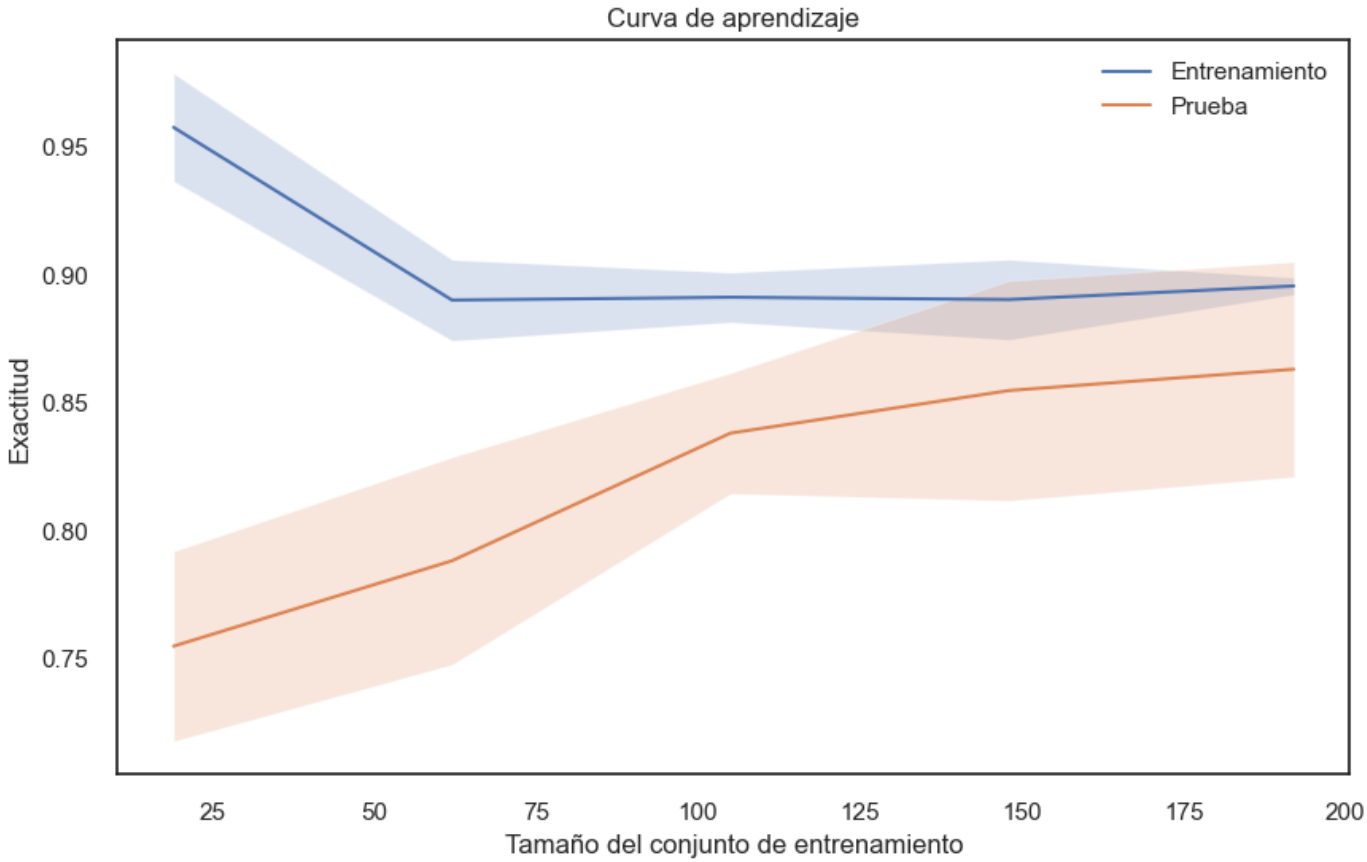
In [91]:
```
# Métricas en el conjunto de entrenamiento
print("Conjunto de entrenamiento:")
print("Precisión:", precision_score(y_train, y_train_pred))
```

```
print("Exactitud:", accuracy_score(y_train, y_train_pred))
print("Sensibilidad:", recall_score(y_train, y_train_pred))
print("Especificidad:", recall_score(y_train, y_train_pred, pos_label=0))

# Calculate the decision scores (equivalent to predicting probabilities for RidgeClassif
decision_scores = modelo.decision_function(X_train)
print("Área bajo la curva ROC:", roc_auc_score(y_train, decision_scores))


# Métricas en el conjunto de prueba
print("Conjunto de prueba:")
print("Precisión:", precision_score(y_test, y_test_pred))
print("Exactitud:", accuracy_score(y_test, y_test_pred))
print("Sensibilidad:", recall_score(y_test, y_test_pred))
print("Especificidad:", recall_score(y_test, y_test_pred, pos_label=0))
# Calculate the decision scores (equivalent to predicting probabilities for RidgeClassif
decision_scores = modelo.decision_function(X_test)
print("Área bajo la curva ROC:", roc_auc_score(y_test, decision_scores))
```

```
Conjunto de entrenamiento:
Precisión: 0.8920863309352518
Exactitud: 0.9045643153526971
Sensibilidad: 0.9393939393939394
Especificidad: 0.8623853211009175
Área bajo la curva ROC: 0.9428690575479567
Conjunto de prueba:
Precisión: 0.896551724137931
Exactitud: 0.8524590163934426
Sensibilidad: 0.8125
Especificidad: 0.896551724137931
Área bajo la curva ROC: 0.947198275862069
```

In [92]:

```
# Calcular la curva de aprendizaje
train_sizes, train_scores, test_scores = learning_curve(modelo, X_train, y_train, cv=5)

# Calcular las medias y desviaciones estándar de las puntuaciones
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Graficar la curva de aprendizaje
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_mean, label='Entrenamiento')
plt.plot(train_sizes, test_mean, label='Prueba')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.2)
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.2)
plt.xlabel('Tamaño del conjunto de entrenamiento')
plt.ylabel('Exactitud')
plt.title('Curva de aprendizaje')
plt.legend()
plt.show()
```



Las dos lineas convergen a medida que aumenta el tamaño del conjundo de entrenamiento, es una señal positiva. Indica que el módelo no esta sufriendo sobreajuste ni subajuste grave, tiene la capacidad de generalizar bien los datos no visto.

- **Cross Validation de nuestro modelo inicial**

In [93]:
```
#modelobbc = BalancedBaggingClassifier(base_estimator=modelo)

result1 = model_evaluation(modelo, X_train, y_train)

#result2 = model_evaluation(modelobbc, X_train, y_train)

#modelo.fit(X_train, y_train)
#modelobbc.fit(X_train, y_train)


print('Conjunto de Datos Inicial: ')
print('f1-score mean (std): %.4f (%.4f)' % (result1[0], result1[1]))
#print('Conjunto de Datos Train: ')
#print('f1-score mean (std): %.4f (%.4f)' % (result2[0], result2[1]))
```

```
Conjunto de Datos Inicial:
f1-score mean (std): 0.8543 (0.0722)
```

Así que nuestro resultado en Validación Cruzada con **0.8543** con una desviación estándar de **0.0722**. Parece que nuestro modelo no está sobreajustado y muestra resultados estables (la desviación estándar de las puntuaciones es baja). Guardamos los datos en la tabla de métricas.

In [94]:
```
results_df =[]

results_df = pd.DataFrame(data = [['Base Model', result1[0], result1[1]]],
                          columns = ['Model', 'CV_f1_macro', 'CV_std'])
results_df
```

Out[94]:

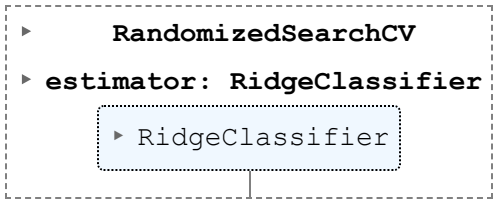
	Model	CV_f1_macro	CV_std
0	Base Model	0.854311	0.072249

2.7 Hyperparameter Tuning

Intentaremos mejorar nuestro modelo utilizando RandomSearch. Primero veamos los parámetros por defecto de nuestro modelo:

In [95]:
```
from sklearn.model_selection import RandomizedSearchCV

modelo= RidgeClassifier()


# Definir los parámetros a ajustar
param_grid = {
    'alpha': [0.1, 1.0, 10.0],  # Valores para el parámetro de regularización alpha
    'solver': ['auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga']  # Métodos
}

# define classifier
#modelo = LogisticRegression()
#modelobbc = BalancedBaggingClassifier(base_estimator=modelo)

# define random search
rg_random = RandomizedSearchCV(modelo,
                               param_distributions = param_grid,
                               n_iter = 10,
                               cv = 5,
                               random_state=42,
                               scoring='f1_macro')
# fit in our train data
rg_random.fit(X_train, y_train)
```

Out[95]:



In [96]:
```
print ('Best Parameters: ', rg_random.best_params_, ' \n')

print ('Best f1 score: ', rg_random.best_score_, ' \n')
```

```
Best Parameters:  {'solver': 'auto', 'alpha': 0.1}

Best f1 score:  0.8704125321894516
```

In [97]:
```
# Evaluamos el resultado del módelo con Cross Validation:

modelo = RidgeClassifier(solver='auto',
                         alpha=0.1
                        )

result3 = model_evaluation(modelo, X_train, y_train)
print(result3)

#modelobbc = BalancedBaggingClassifier(base_estimator=modelo)
```

```
#result4 = model_evaluation(modelobbc, X_train, y_train)
#print(result4)
```

```
(0.8549228403278891, 0.073990975226762)
```

In [98]:
```
results_df1 = pd.DataFrame(data = [['Random Search', result3[0], result3[1]]],
                           columns = ['Model', 'CV_f1_macro', 'CV_std'])

results_df = pd.concat([results_df, results_df1])
results_df
```

Out[98]:

	Model	CV_f1_macro	CV_std
0	Base Model	0.854311	0.072249
0	Random Search	0.854923	0.073991

- Refinamos con gridsearch

In [99]:
```
# Módelo base

modelo= RidgeClassifier()

# Definir los parámetros a ajustar
param_grid = {
    'alpha': [0.1, 1.0, 10.0],  # Valores para el parámetro de regularización alpha
    'solver': ['auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga']  # Métodos
}


#cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=10, random_state=42)

grid_search = GridSearchCV(modelo,
                           param_grid = param_grid,
                           cv=5,
                           scoring='f1_macro')

grid_result = grid_search.fit(X_train, y_train)
```

In [100…
```
print ('Best Parameters: ', grid_search.best_params_, ' \n')

print ('Best f1 score: ', grid_search.best_score_, ' \n')
```

```
Best Parameters:  {'alpha': 0.1, 'solver': 'auto'}

Best f1 score:  0.8704125321894516
```

In [101…
```
# Evaluamos el resultado del módelo con Cross Validation:

modelo = RidgeClassifier(solver='auto',
                         alpha=0.1
                        )

result5 = model_evaluation(modelo, X_train, y_train)
print(result5)

#modelobbc = BalancedBaggingClassifier(base_estimator=modelo)

#result6 = model_evaluation(modelobbc, X_train, y_train)
#print(result4)
```

```
(0.8549228403278891, 0.073990975226762)
```

In [102…
```
# store results
results_df2 = pd.DataFrame(data = [['GridSearchCV', result5[0], result5[1]]],
                           columns = ['Model', 'CV_f1_macro', 'CV_std'])

results_df = pd.concat([results_df, results_df2])
results_df
```

Out[102]:

	Model	CV_f1_macro	CV_std
0	Base Model	0.854311	0.072249
0	Random Search	0.854923	0.073991
0	GridSearchCV	0.854923	0.073991

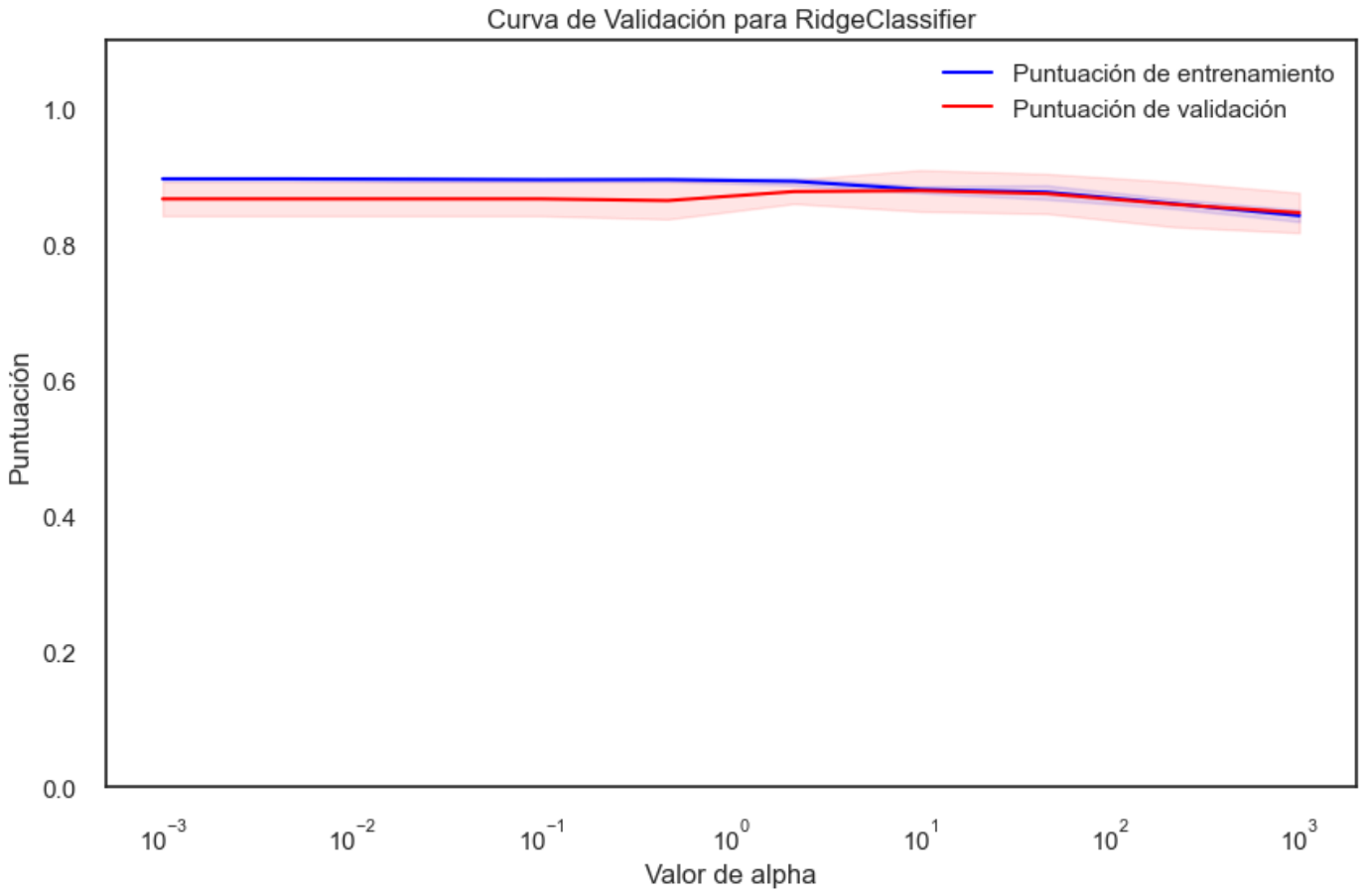
In [103…
```
print ('Parameters of original model: ', modelo.get_params(), ' \n')
```

```
Parameters of original model:  {'alpha': 0.1, 'class_weight': None, 'copy_X': True, 'fit
_intercept': True, 'max_iter': None, 'positive': False, 'random_state': None, 'solver':
'auto', 'tol': 0.0001}
```

- Buscamos mejorar alpha, veremos la curva de validación.

In [104…
```
# Definir los valores de alpha a evaluar
alphas = np.logspace(-3, 3, num=10)

# Crear un clasificador Ridge
ridge = RidgeClassifier()

# Calcular la puntuación de validación cruzada para diferentes valores de alpha
train_scores, valid_scores = validation_curve(
    ridge, X, y, param_name="alpha", param_range=alphas, cv=5, scoring="f1"
)

# Calcular las medias y desviaciones estándar de las puntuaciones
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
valid_mean = np.mean(valid_scores, axis=1)
valid_std = np.std(valid_scores, axis=1)

# Graficar la curva de validación
plt.figure(figsize=(10, 6))
plt.title("Curva de Validación para RidgeClassifier")
plt.xlabel("Valor de alpha")
plt.ylabel("Puntuación")
plt.ylim(0.0, 1.1)
plt.semilogx(alphas, train_mean, label="Puntuación de entrenamiento", color="blue")
plt.fill_between(
    alphas,
    train_mean - train_std,
    train_mean + train_std,
    alpha=0.1,
    color="blue"
)
plt.semilogx(alphas, valid_mean, label="Puntuación de validación", color="red")
plt.fill_between(
    alphas,
    valid_mean - valid_std,
    valid_mean + valid_std,
    alpha=0.1,
    color="red"
)
```

```
plt.legend(loc="best")
plt.show()
```



Modificamos el parámetro alpha, no conseguimos mejorar el modelo.

In [105…

```
# Evaluamos el resultado del módelo con Cross Validation:

modelo = RidgeClassifier(solver='auto',
                         alpha=4
                        )

result6 = model_evaluation(modelo, X_train, y_train)
print(result6)
```

```
(0.8461528617068709, 0.07902498395585794)
```

No hemos mejorado el modelo en comparación al original, dado por pycaret.

En la carpeta adjunta en git hub de pruebas realizadas, se ha probado con RFE (Recursive Feature Elimination), donde nos ha dado los atributos que ha considerado más importantes. Pero el resultado ha sido peor, tanto en la matrix de confusión como en las diferentes métricas.

Volver Índice general

3. Modelo Final

3.1 Modelo definitivo

Decido realizar nuestro modelo con los datos proporcionados con la librería pycaret. Añadiremos el estudio del balanceado en el mismo.

In [106…
```
# define predictors
X = df_scaled.drop(['target'], axis = 1)
# define target
y = df_scaled['target']

# choose test size of 20%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state= 4
```

In [107…
```
model = RidgeClassifier(alpha=1.0, class_weight=None, copy_X=True, fit_intercept=True,
                max_iter=None, positive=False, random_state=42, solver='auto',
                tol=0.0001)

# entrenamos el módelo
#modelo.fit(X_train, y_train)

# predictions on test set with our trained model
#modelo_predictions = modelo.predict(X_test)

#utilizamos nuestro balanceador elegido anteriormente

modelo =  BalancedBaggingClassifier(base_estimator=model,
                                    sampling_strategy='auto',
                                    replacement=False,
                                    random_state=42)
# entrenamos el módelo
modelo.fit(X_train, y_train)

# predictions on test set with our trained model
modelo_predictions = modelo.predict(X_test)
```

In [108…
```
# Calculamos métricas del módelo
df_final=GuardarMetricas(y_test,modelo_predictions)

titulo = 'Métricas del módelo Final'
dibujartabla(titulo,df_final)
```

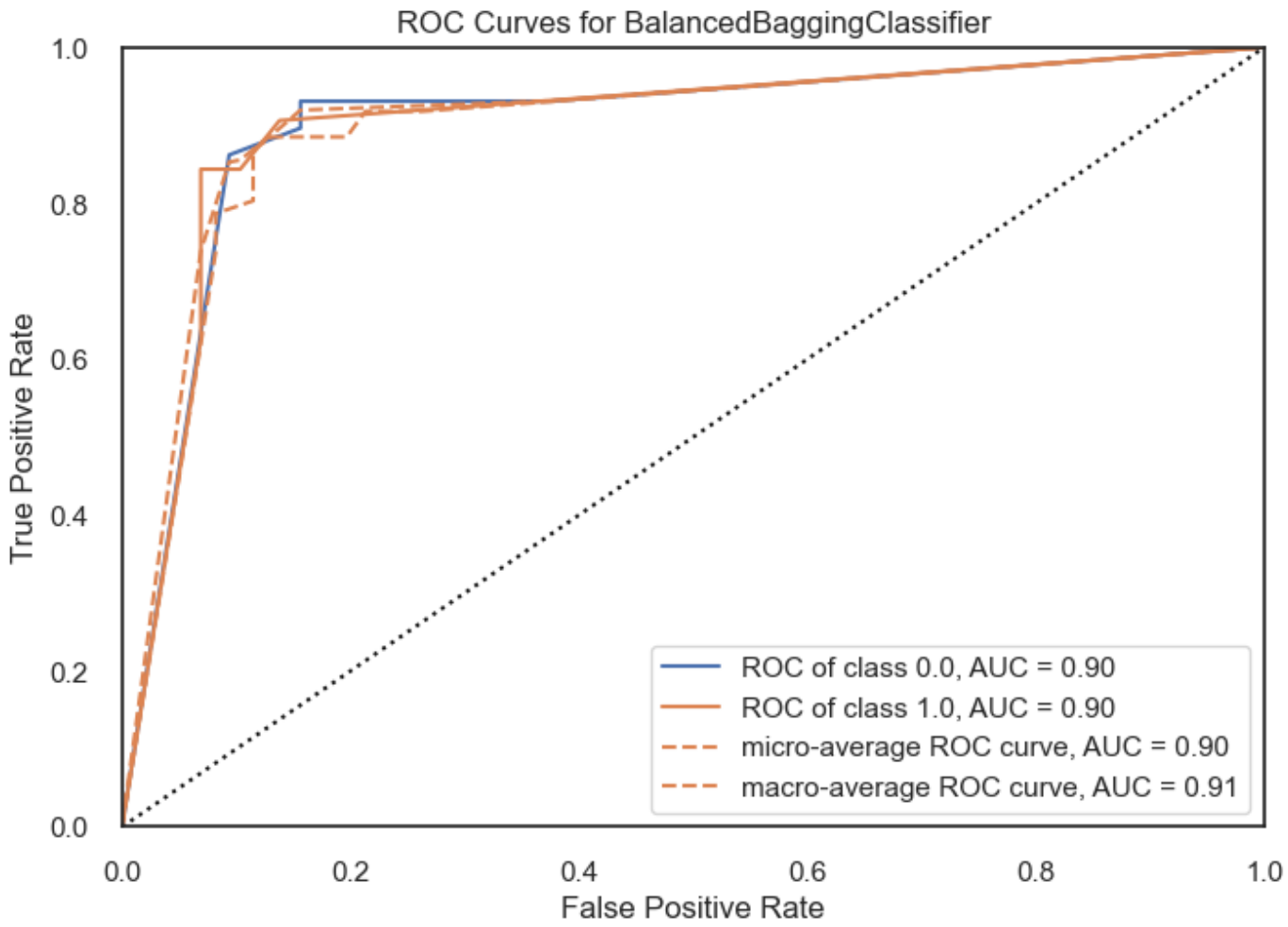
Métricas del módelo Final

Métrica	precisio	recall	f1-score	support	Valor
Less chance H.A	0.82	0.93	0.87	29	
More chance H.A	0.93	0.81	0.87	32	
accuracy	0.87	0.87	0.87	0.87	
macro avg	0.87	0.87	0.87	61	
weighted avg	0.88	0.87	0.87	61	
Matthews_corrcoef					0.75

3.2 Curva ROC-AUC

Una curva ROC es un gráfico que nos muestra el rendimiento de clasificación. Relaciona la sensibilidad con el porcentaje de falsos positivos.

In [109…
```
grafica_ROC_curve(modelo,X_train, y_train, X_test, y_test)
plt.show()
```



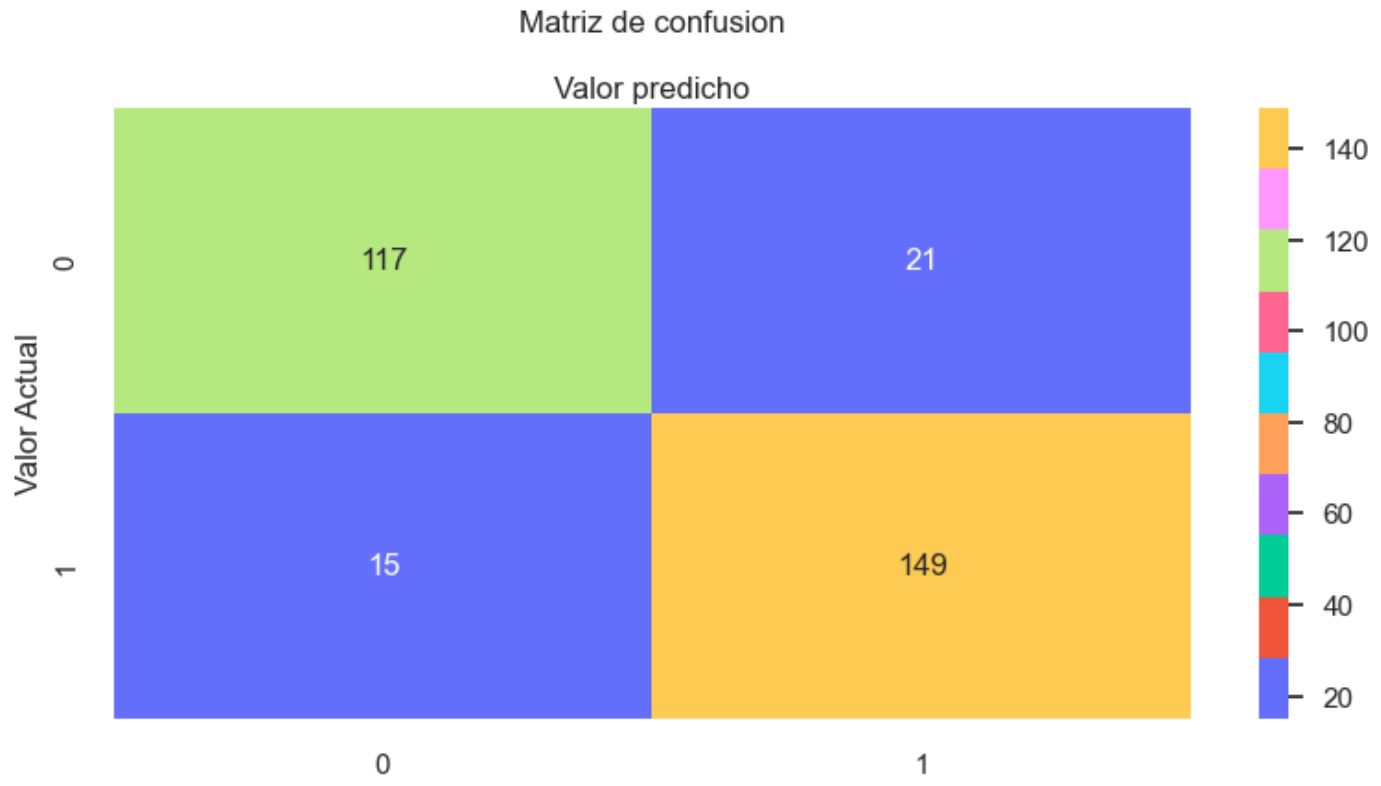
- Preparamos nuestro modelo todo el conjundo de datos del Dataset, utilizaremos df_scaled dónde los atributos están estandarizados.

In [110…
```
# fit on all data
modelo.fit(X, y)
modelo_predictions = modelo.predict(X)
```

3.3 Matrix de Confusión

In [111…
```
grafica_matrix (y,modelo_predictions)
```

```
Matriz de Confusión:
```



In [112…

```
# Calculamos métricas del módelo
df_final=GuardarMetricas(y,modelo_predictions)

titulo = 'Métricas del módelo Final'
dibujartabla(titulo,df_final)
```

Métricas del módelo Final

Métrica	precisio	recall	f1-score	support	Valor
Less chance H.A	0.89	0.85	0.87	138	
More chance H.A	0.88	0.91	0.89	164	
accuracy	0.88	0.88	0.88	0.88	
macro avg	0.88	0.88	0.88	302	
weighted avg	0.88	0.88	0.88	302	
Matthews_corrcoef					0.76

3.4 Probando el modelo

Creo un Dataset a partir del original con 202 filas, barajadas.

In [113…
```
path2=r'C:\Users\Nitropc\IT Academy\Data Science\Proyecto Data Science\Fuente de Datos\h
#Funciones
# Intento de Mecanizar acciones/herramientas para el analisis de cualquier heartSet
# Guardar información de los pasos EDA


#Cuerpo
#heart = pd.read_csv(path,sep=',',encoding='ISO-8859-1')
df = pd.read_csv(path2,sep=',',encoding='latin-1')
```

- Compruebo que no tenga duplicados.

In [114…
```
print(Style.BRIGHT +'DataSet Heart:\n'+Style.RESET_ALL)
print(df.duplicated().sum())
df=df.drop_duplicates()
print(df.duplicated().sum())
```

DataSet Heart:

```
1
0
```

- Estandarizamos el Dataset de Test.

In [115…
```
x_scaled = column_transformer.fit_transform(df)
df_prueba=pd.DataFrame(x_scaled,columns=columnas_Transformar)
dibujartabla('\n\nDatos de la estandarización:\n',round(df_prueba[0:5],6))
```

Datos de la estandarización:

age	se	cp	trtb	chol	fbs	res	thalac	ex	oldpea	slp	ca	tha	Sat_	tar
0.595	1	0	1	0.4888	0	0	-1.258	1	-0.414	1	1	1	0.476	0
0.638	1	3	2	0.6666	0	0	0.2580	0	-0.740	1	0	3	0.476	0
0.595	1	2	1	-1.733	1	1	0.7096	0	-0.740	2	1	3	0.476	1
0.574	0	0	0.2	2.4592	0	0	-0.032	1	0.6437	1	2	3	0.476	0
0.893	0	2	-1	0.3259	1	0	-0.677	0	-0.903	2	1	2	0.476	1

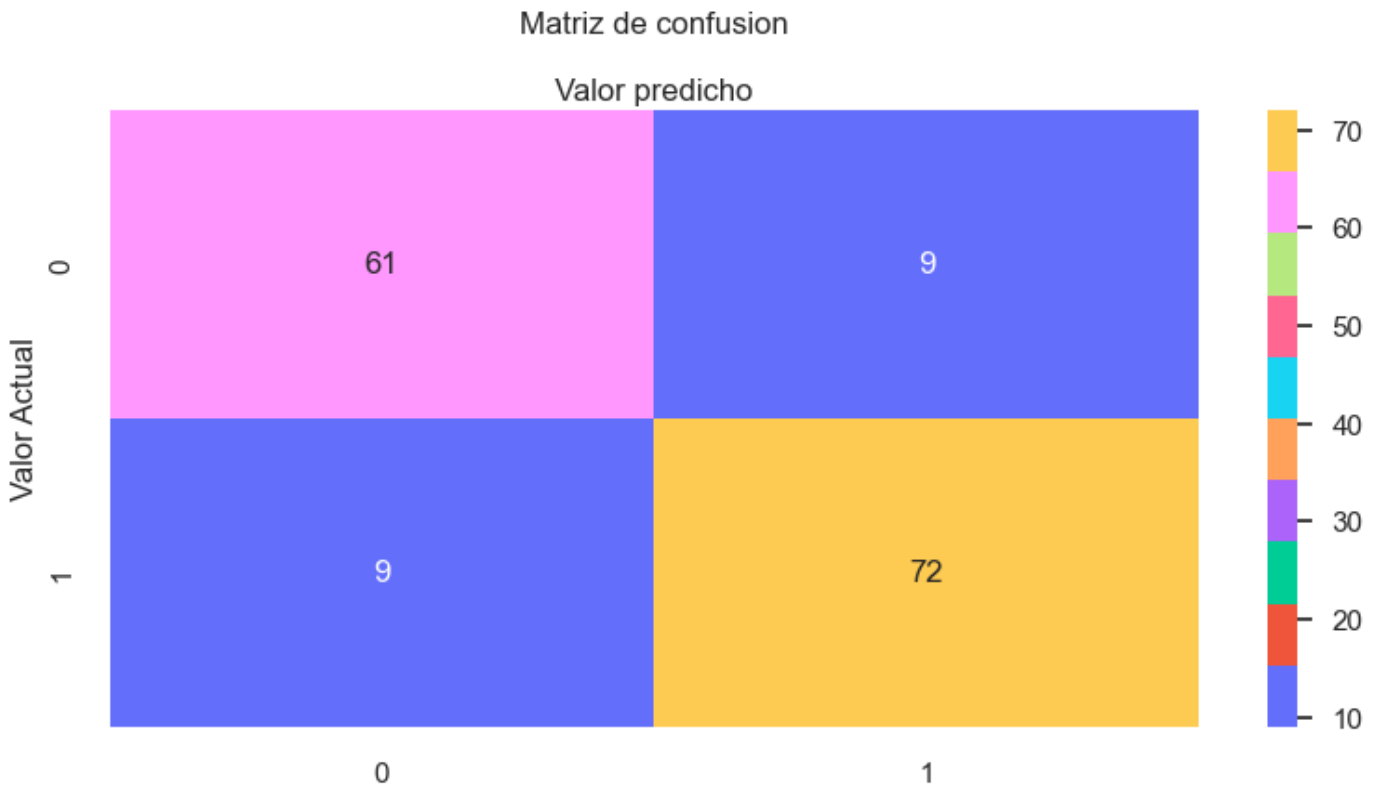
In [116…
```
# define predictors
X = df_prueba.drop(['target'], axis = 1)
# define target
y = df_prueba['target']

print('df_scaled.shape: ', df_prueba.shape)
print('y.shape: ', y.shape)
print('X.shape', X.shape)
```

```
df_scaled.shape:  (151, 15)
y.shape:  (151,)
X.shape (151, 14)
```

In [117…
```
# Predict heart attack risk using trained model
predict_value = modelo.predict(X)
#print('Predict Output : ', predict_value)
```

In [118…
```
grafica_matrix (y,predict_value)
```

```
Matriz de Confusión:
```



In [119…
```
#X3=df[df['target']==1]
#X3['Sat_level'].unique()
```

In [120…
```
# Calculamos métricas del módelo
df_final=GuardarMetricas(y,predict_value)

titulo = 'Métricas del módelo Final Test'
dibujartabla(titulo,df_final)
```

Métricas del módelo Final Test

Métrica	precisio	recall	f1-score	support	Valor
Less chance H.A	0.87	0.87	0.87	70	
More chance H.A	0.89	0.89	0.89	81	
accuracy	0.88	0.88	0.88	0.88	
macro avg	0.88	0.88	0.88	151	
weighted avg	0.88	0.88	0.88	151	
Matthews_corrcoef					0.76

In [121…
```
# Input values for prediction
#57,1,0,150,276,0,0,112,1,0.6,1,1,1,97.5,0
#59,1,3,170,288,0,0,159,0,0.2,1,0,3,97.5,0
#55,1,0,132,353,0,1,132,1,1.2,1,1,3,96,0
prueba ={
'age':[50,57,59,55],
'sex':[1,1,1,1],
'cp':[3,0,3,0],
'trtbps':[110,150,170,132],
'chol':[264,276,288,353],
'fbs':[1,0,0,0],
'restecg':[1,0,0,1],
'thalachh':[300,112,159,132],
'exng':[0,1,0,1],
'oldpeak':[1.2,0.6,0.2,1.2],
'slp':[1,1,1,1],
'caa':[0,1,0,1],
'thall':[3,1,3,1],
'Sat_level':[96.5,96.5,96.5,96.5],
'target':[1,0,0,0],
}
prueba = pd.DataFrame(prueba)
```

In [122…
```
x_scaled = column_transformer.fit_transform(prueba)
df_prueba=pd.DataFrame(x_scaled,columns=columnas_Transformar)
X2 = df_prueba.drop(['target'], axis = 1)
#dibujartabla('\n\nDatos de la estandarización:\n',round(df_prueba[0:5],6))
```

In [123…
```
predict_value = modelo.predict(X2)
print('Predict Output : ', predict_value)
```

```
Predict Output :  [1. 0. 0. 0.]
```

Conclusiones

En el sprint 7, realizamos un ejerccio muy completo con diferentes modelos de ML supervisado. En este proyecto hemos utilizado la librería **pycaret**, para que nos realizará la busqueda del algoritmo (módelo) que mejor se comportaba con este DataSet. La elección ha sido **RidgeClassifier**, cuyas métricas hemos podido mejorar un poco respecto al obtenido con la librería pycaret. Aunque la busqueda de hiper parámetros no tiene tantas posibilidades como otros modelos.

En la exploración de datos, hemos realizado la comparción siempre con el target dejando un poco de lado la relación que puedan tener algunos entre ellos. Al final de este proyecto he dejado un script que quiere subsanar ese punto.

En cuanto a la correlación de los datos, podemos concluir que los atributos no estan demasiado correlacionados, por debajo +/- 0.7.

El algoritmo de análisis de Componente Principales (PCA), nos ha permitido observar que casi todas las características del dataset tienen mucha relevancia a la hora de entrenar el modelo y que ninguna de ellas puede aportar la información de otra.

Bibliografía

- Curso Data Science.
- Información del DataSet
- Ejemplos Heart Attack
- Thallium Stress Test Result.
- Componentes Principales (PCA)

Volver Índice general

In [124…
```
!pip install session_info
import session_info


session_info.show()
```

```
Requirement already satisfied: session_info in c:\users\nitropc\appdata\roaming\python\p
ython310\site-packages (1.0.0)
Requirement already satisfied: stdlib-list in c:\users\nitropc\appdata\roaming\python\py
thon310\site-packages (from session_info) (0.8.0)
```

Out[124]: ▶ Click to view session information

In [125…
```
!pip freeze > requirements.txt
```

1.6 Estudio dinámico de los Atributos

Falta depurarlo, el tiempo se ha tirado encima. La idea es que se comparen los atributos entre ellos, pero falta depurar.

In [126…
```
df = pd.DataFrame(heart)

# Definir diccionarios de conversión para los atributos
```

```
lista = ['age','trtbps', 'chol', 'thalachh', 'oldpeak', 'thall', 'Sat_level']

categoricos = ['sex','cp','fbs','restecg','exng','slp','caa','thall','target']


conversion_dict = {
'age': {
        'title': 'Age',
        'values': {'min': df['age'].min(), 'max': df['age'].max()}},
'sex': {'title': 'Gender',
        'values': {0: 'Female(0)', 1: 'Male(1)'}}, #genero
'exng': {'title': 'Exercise-induced Angina',
         'values': {0:'No(0)', 1:'Yes(1)'}}, #angina inducida por el ejercicio
'caa': {'title': 'Number of major vessels',
        'values': {0:'vessels(0)', 1:'vessels(1)',2:'vessels(2)', 3:'vessels(3)',4:'vess
'cp': {'title': 'Chest Pain type chest pain type',
       'values': {0:'Typical angina(0)', 1:'Atypical angina(1)', 2:'Non-anginal pain(2)'
'fbs': {'title': 'Fasting blood sugar > 120 mg/dl',
        'values': {0:'<121mg/dl(0)', 1:'>120mg/dl(1)'}}, #fasting blood sugar > 120 mg/d
'restecg': {'title': 'Resting electrocardiographic results',
            'values': {0:'Normal(0)', 1:'Having ST-T wave abnormality(1)',2:'Hypertrophy
    # Agrega otros diccionarios de conversión para los atributos adicionales
'trtbps': {
        'title': 'Resting blood pressure (in mm Hg)',
        'values': {'min': df['trtbps'].min(), 'max': df['trtbps'].max()}},
'chol': {
        'title': 'Cholestoral in mg/dl fetched via BMI sensor',
        'values': {'min': df['chol'].min(), 'max': df['chol'].max()}},
'thalachh': {
        'title': 'Maximum heart rate achieved',
        'values': {'min': df['thalachh'].min(), 'max': df['thalachh'].max()}},
'oldpeak': {
        'title': 'Previous peak',
        'values': {'min': df['oldpeak'].min(), 'max': df['oldpeak'].max()}},
'slp': {'title': 'Slope of the peak exercise',
        'values': {0:'Donwsloping(0)', 1:'Flat(1)', 2:'Upsloping(2)'}},
'thall': {
        'title': 'Thalium Stress Test result',
        'values': {'min': df['thall'].min(), 'max': df['thall'].max()}},
'Sat_level': {
        'title': 'O2 saturation',
        'values': {'min': df['Sat_level'].min(), 'max': df['Sat_level'].max()}},
'target': {'title': 'Heart attack chance',
        'values': {0:'less chance of heart attack(0)', 1:'More chance of heart attack(1)
}

# Crear la aplicación Dash con suppress_callback_exceptions=True
app = dash.Dash(__name__)

# Estilos CSS para los componentes HTML
attribute_name_style = {'color': 'blue'}
attribute_title_style = {'color': 'black'}
attribute_values_style = {'color': 'red'}

# Diseño de la aplicación
app.layout = html.Div([
    html.H1("Estudio de Atributos"),
    html.Div(id='attribute-values'),
    html.H2('Tabla de Valores'),
    html.Div(id='attribute-table'),
    html.Div([
        dcc.Dropdown(
            id='x-axis',
            options=[{'label': col, 'value': col} for col in df.columns],
            value=df.columns[0]
        )
```

```
    ],
        style={'width': '48%', 'display': 'inline-block'}),
    html.Div([
        dcc.Dropdown(
            id='y-axis',
            options=[{'label': col, 'value': col} for col in df.columns],
            value=df.columns[1]
        )
    ],
    style={'width': '48%', 'float': 'right', 'display': 'inline-block'}),
    html.Div([
        dcc.Dropdown(
            id='chart-type',
            options=[
                {'label': 'Gráfico de Barras', 'value': 'bar'},
                #{'label': 'Gráfico de Curve', 'value': 'Curve'},
                {'label': 'Gráfico de Dispersión', 'value': 'scatter'},
                {'label': 'Gráfico de Pastel', 'value': 'pie'}
            ],
            value='scatter'
        )
    ],
        style={'width': '48%', 'display': 'inline-block'}),
    dcc.Graph(id='graph')
])


# Callback para mostrar los valores únicos en texto y el diccionario correspondiente
@app.callback(
    [Output('attribute-table', 'children'),
     Output('graph', 'figure')],
    [Input('x-axis', 'value'),
     Input('y-axis', 'value'),
     Input('chart-type', 'value')])

def show_attribute_values(x_axis, y_axis, chart_type):
    attribute_values = pd.DataFrame(columns=['Atributo', 'Nombre', 'Valores'])

    for attribute, conversion in conversion_dict.items():
        title = conversion['title']
        values = conversion['values']

        if attribute in lista:
            formatted_values = 'Min: {}, Max: {}'.format(values['min'], values['max'])
        else:
            unique_values = df[attribute].unique()
            formatted_values = [str(values[val]) if val in values else str(val) for val
            formatted_values = ', '.join(formatted_values)

        attribute_values = attribute_values.append({'Atributo': attribute, 'Nombre': tit

    if chart_type == 'scatter':

        # Evitamos que los valores categóricos esten el eje de las x, no se representan
        if y_axis in categoricos:
            fig = px.scatter(df, x=x_axis, y=y_axis,color=y_axis, marginal_x="box")
        elif x_axis in categoricos:
            fig = px.scatter(df, x=y_axis, y=x_axis, color=x_axis, marginal_x="box")
        else:
            fig = px.scatter(df, x=x_axis, y=y_axis, color=y_axis, marginal_x="box", mar



    elif chart_type == 'bar':

        # Evitamos que los valores categóricos esten el eje de las x, no se representan
```

```
        if x_axis == y_axis:
            fig = px.histogram(df, x=x_axis, color=x_axis, histfunc = "count")
        elif y_axis in categoricos:
            fig = px.histogram(df, x=x_axis, color=y_axis, barmode='group')
        elif x_axis in categoricos:
            fig = px.histogram(df, x=y_axis, color=x_axis, barmode='group')
        else:
            fig = px.bar(df, x=x_axis, y=y_axis)

    elif chart_type == 'pie':
        if y_axis in categoricos:
            fig = px.pie(df, names=df[y_axis], title=f"Distribución de {y_axis}")
        else:
            fig = px.pie(df, names=y_axis, title=f"Distribución de {y_axis}")

    fig.update_layout(title='Atributo {} & Atributo {}'.format(x_axis, y_axis)
                      )

    return generate_attribute_table(attribute_values), fig


# Función para generar la tabla de atributos
def generate_attribute_table(df):
    table = html.Table([
        html.Thead(html.Tr([
            html.Th('Atributo', style=attribute_name_style),
            html.Th('Nombre', style=attribute_name_style),
            html.Th('Valores', style=attribute_name_style)
        ])),
        html.Tbody([
            html.Tr([
                html.Td(attribute, style=attribute_title_style),
                html.Td(nombre, style=attribute_title_style),
                html.Td(values, style=attribute_values_style)
            ]) for attribute, nombre, values in zip(df['Atributo'], df['Nombre'], df['Va
        ])
    ])

    return table


# Función para detener la ejecución de la aplicación Dash al capturar una señal de tecla
def stop_execution(signal, frame):
    os._exit(0)


# Capturar la señal SIGINT (Ctrl+C) para detener la ejecución
signal.signal(signal.SIGINT, stop_execution)


def EjecutarApp(empezar):
 # Ejecutar la aplicación Dash
    if __name__ == empezar:
        app.run_server(debug=True, use_reloader=False)
    #app.run_server(debug=False)
```

In [127…

```
#Ejecutar la aplicación Dash
#EjecutarApp( '__main__')
```

In [ ]: