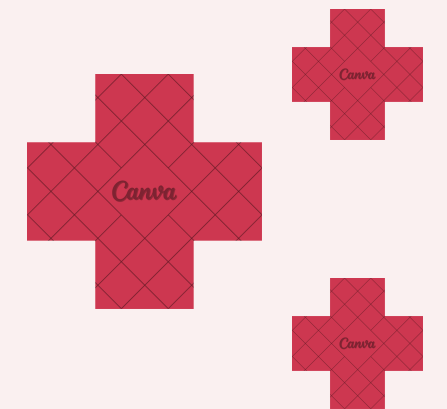
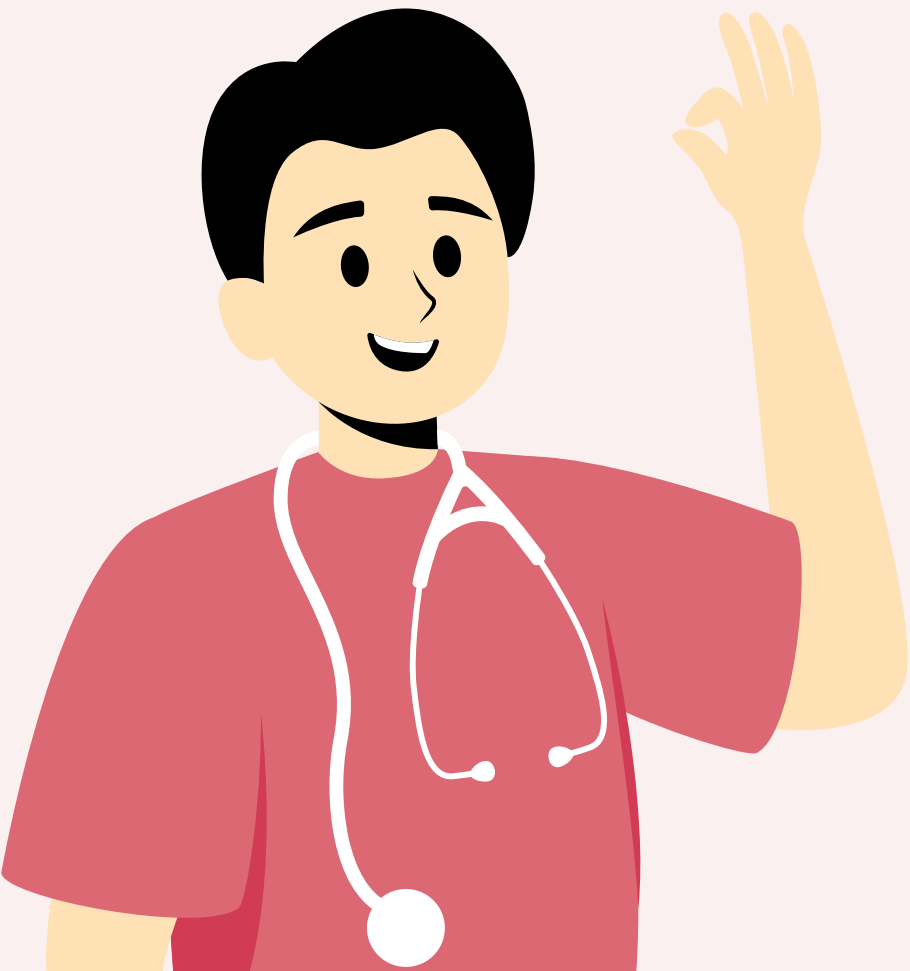


# Prediccion de **Insuficiencia cardiaca**

Juan Sebastian Alvarez Mesa-2220074  
Jhon Anderson Vargas Gomez-2220086





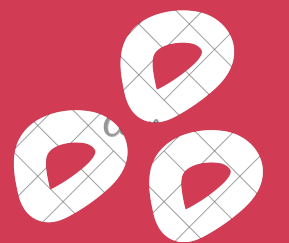
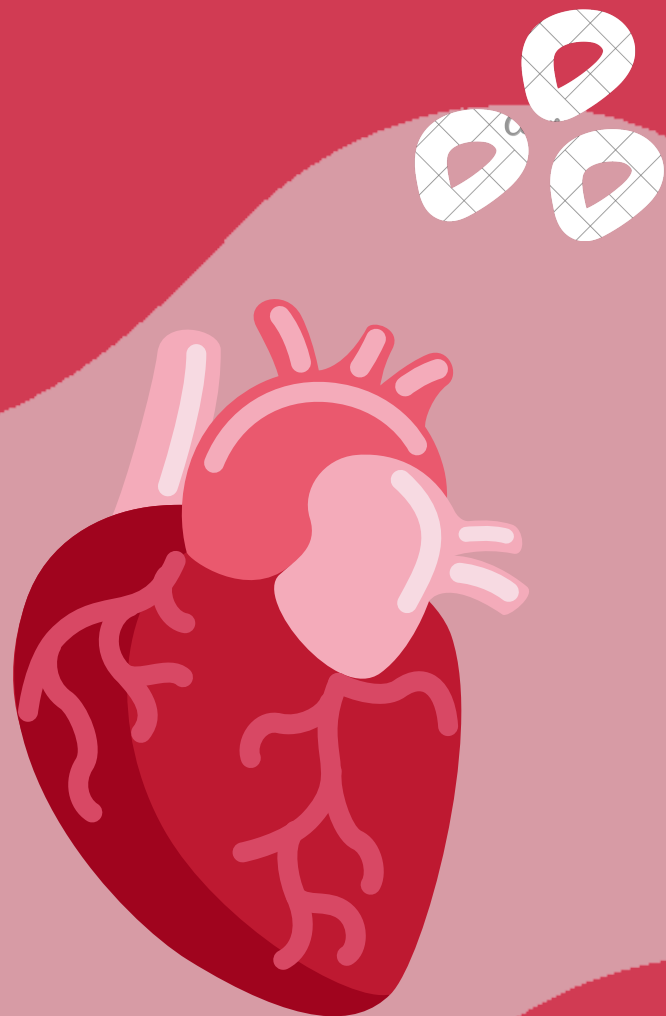
# Motivacion

El corazon es es uno de los organos vitales del ser humano y al saber que en colombia la causa principal de muertes son por enfermedades isquemicas cardiacas, nos vimos en la necesidad de predecir cuando una persona puede sufrir una falla cardiaca

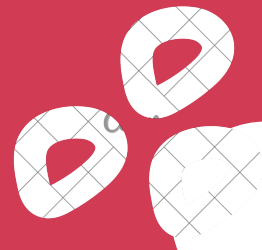


# Objetivo

Generar predicciones que nos muestre que si la persona puede sufrir una falla cardiaca evaluando diferentes características de la enfermedad para así prevenir situaciones de riesgo y mejorar la respuesta frente a las fallas cardiacas



# Informacion obtenida



Edad

Sexo

Presion Arterial en reposo

Tipo de dolor en el pecho

Colesterol en sangre

Azucar en sangre en ayunas



# Informacion obtenida

Electrocardiograma en reposo

Enfermedad cardíaca

Angina inducida por el  
ejercicio


Depresión del ST después del  
ejercicio(Oldpeak)

ST\_Slope (Pendiente del  
segmento ST después del  
ejercicio)

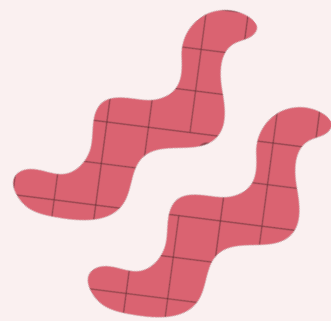
Frecuencia cardíaca máxima  
alcanzada

```
import kagglehub
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('ggplot')
import seaborn as sns
sns.set_style('darkgrid')
sns.set_palette('deep')

# Download latest version
df = pd.read_csv("/content/drive/MyDrive/Proyecto IA/heart.csv")
df.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 918 entries, 0 to 917
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  -
 0   Age                   918 non-null   int64  
 1   Sex                   918 non-null   object  
 2   ChestPainType         918 non-null   object  
 3   RestingBP             918 non-null   int64  
 4   Cholesterol           918 non-null   int64  
 5   FastingBS             918 non-null   int64  
 6   RestingECG           918 non-null   object  
 7   MaxHR                 918 non-null   int64  
 8   ExerciseAngina        918 non-null   object  
 9   Oldpeak               918 non-null   float64 
10   ST_Slope              918 non-null   object  
11   HeartDisease          918 non-null   int64  
dtypes: float64(1), int64(6), object(5)
memory usage: 86.2+ KB
```



# Procesamiento dataset

Para el procesamiento de datos, utilizamos Pandas para la manipulación y limpieza del dataset. Se verificó igualmente su estructura con `df.info()` para entender las variables y detectar valores faltantes.



```
from sklearn.preprocessing import LabelEncoder

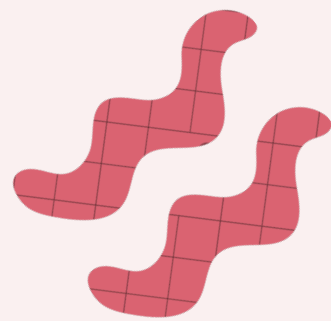
# Label Encoding para transformar las variables categóricas y así poder ejecutar los modelos
label_cols = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope']
le = LabelEncoder()

for col in label_cols:
    df[col] = le.fit_transform(df[col])
```

df.dtypes

	0
Age	int64
Sex	int64
ChestPainType	int64
RestingBP	int64
Cholesterol	int64
FastingBS	int64
RestingECG	int64
MaxHR	int64
ExerciseAngina	int64
Oldpeak	float64
ST_Slope	int64
HeartDisease	int64

dtype: object

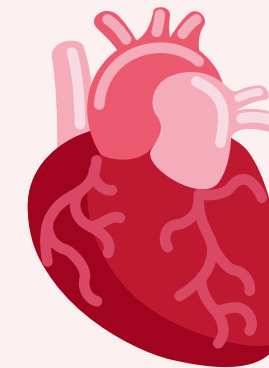


# Procesamiento dataset

Además se usó un label encoding para transformar las variables categóricas en valores numéricos, es necesario para que estos datos puedan ser usados en modelos de clasificación.

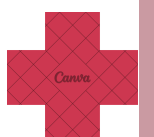
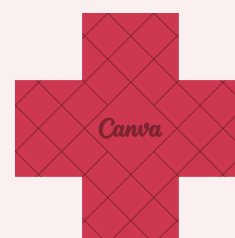


# Estadísticas y graficas

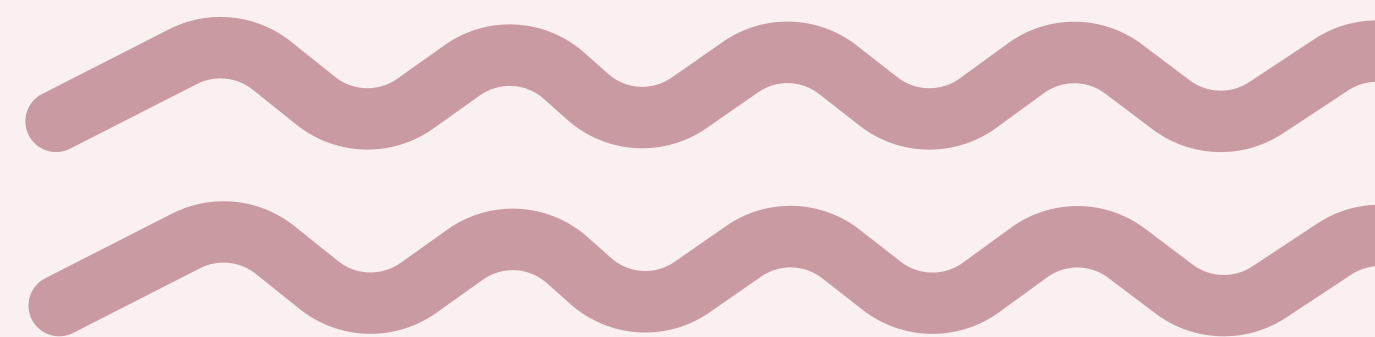
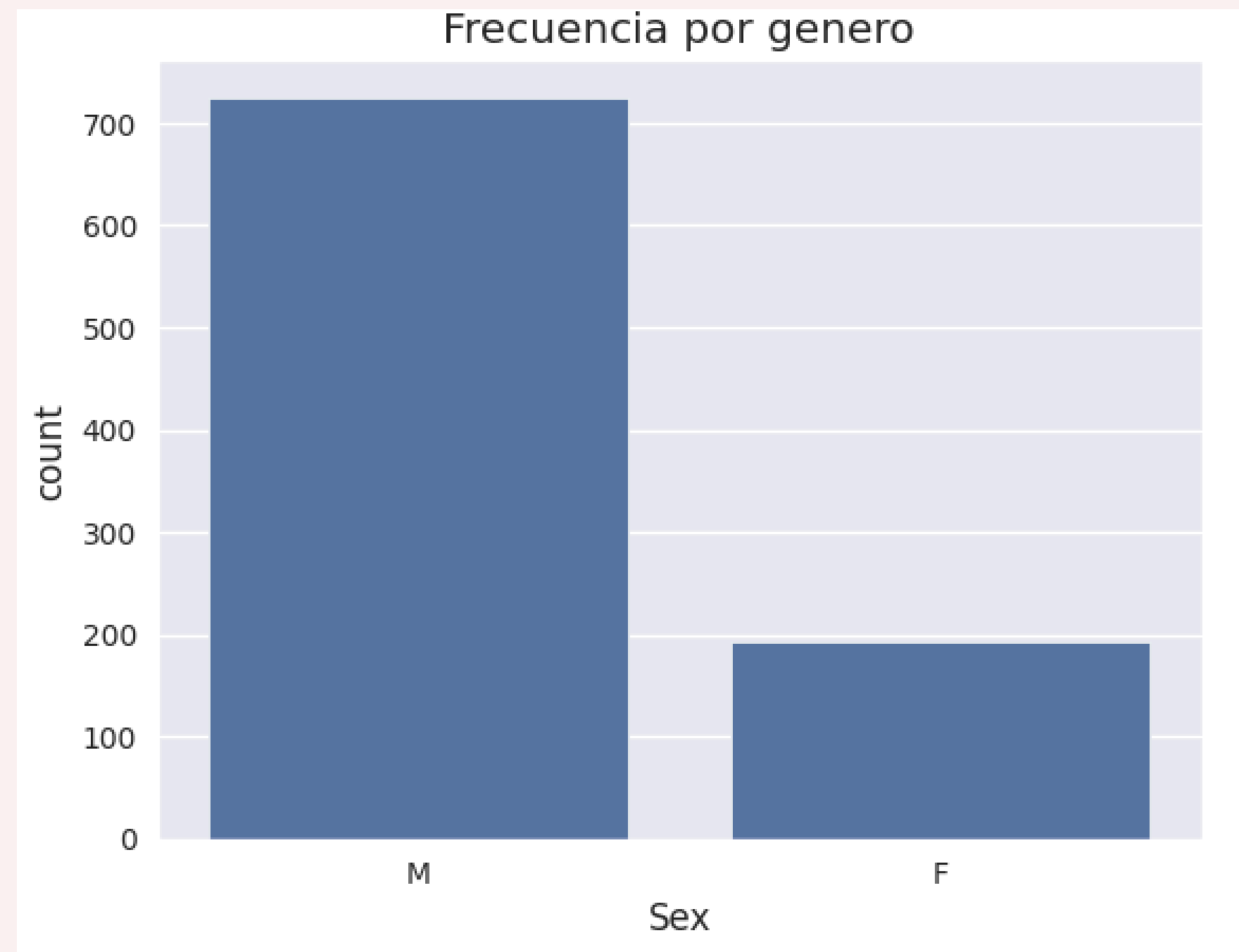
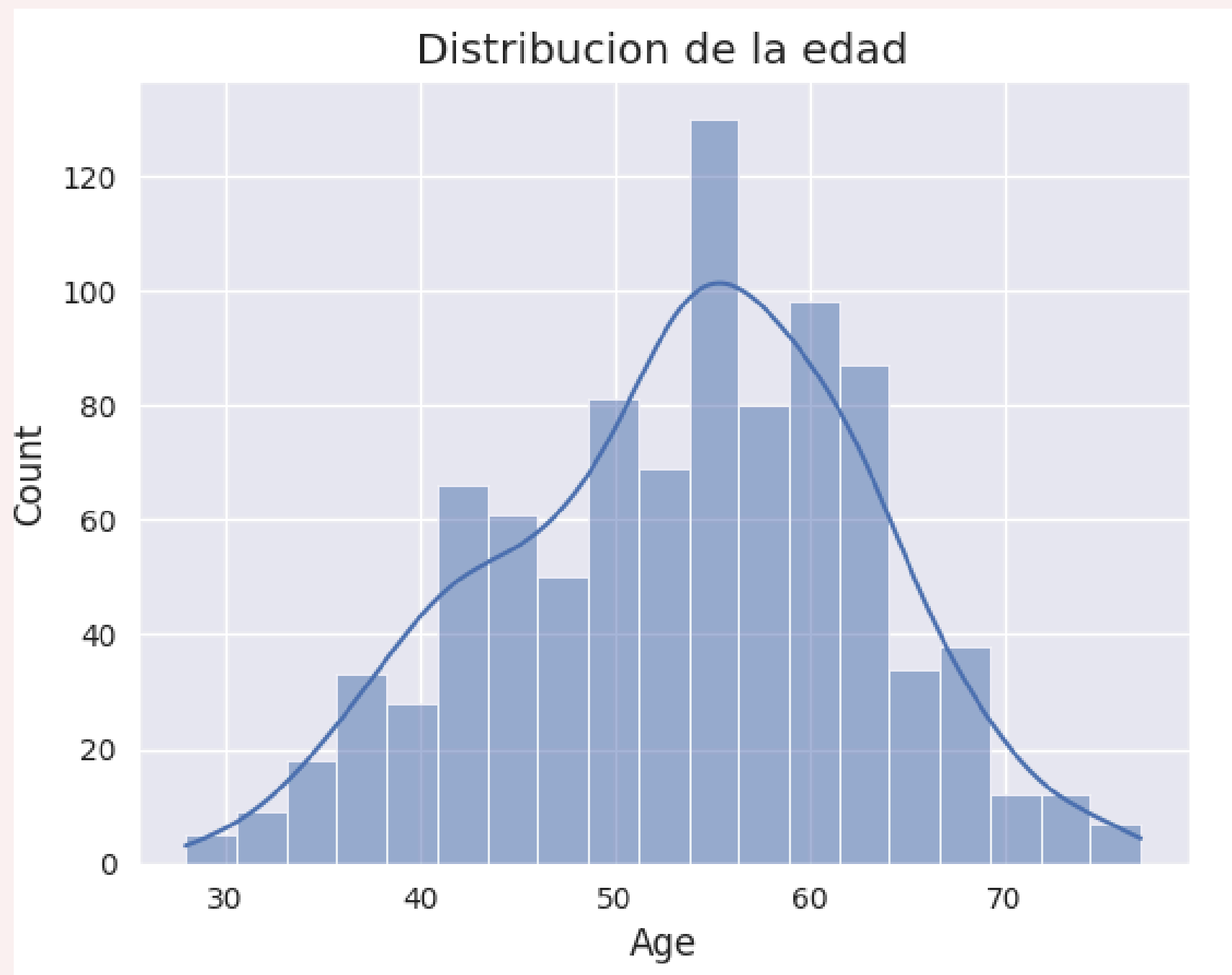


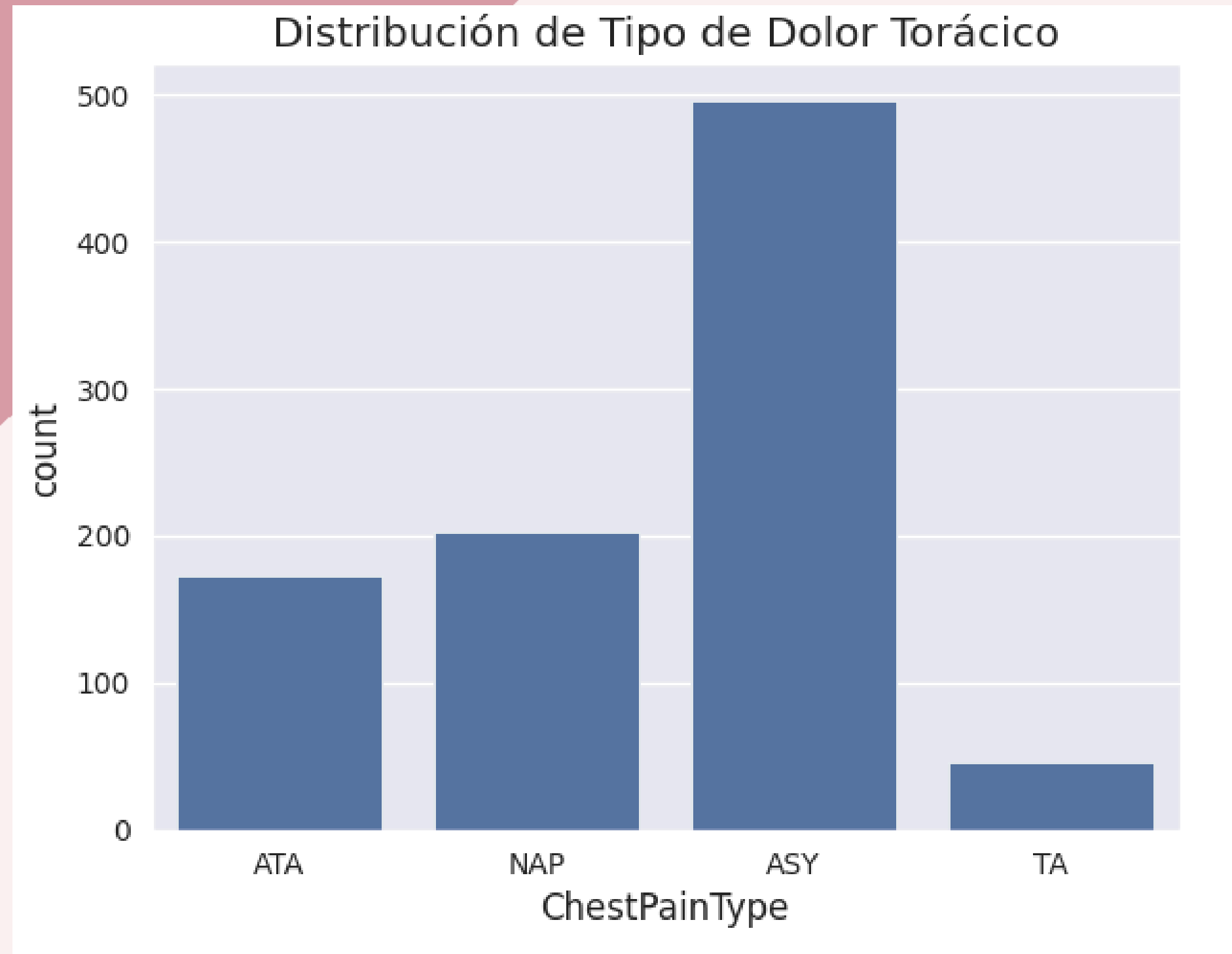
	count	mean	std	min	25%	50%	75%	max
Age	918.0	53.510893	9.432617	28.0	47.00	54.0	60.0	77.0
RestingBP	918.0	132.396514	18.514154	0.0	120.00	130.0	140.0	200.0
Cholesterol	918.0	198.799564	109.384145	0.0	173.25	223.0	267.0	603.0
FastingBS	918.0	0.233115	0.423046	0.0	0.00	0.0	0.0	1.0
MaxHR	918.0	136.809368	25.460334	60.0	120.00	138.0	156.0	202.0
Oldpeak	918.0	0.887364	1.066570	-2.6	0.00	0.6	1.5	6.2
HeartDisease	918.0	0.553377	0.497414	0.0	0.00	1.0	1.0	1.0

```
Sex
M      725
F      193
Name: count, dtype: int64
ChestPainType
ASY     496
NAP     203
ATA     173
TA       46
Name: count, dtype: int64
RestingECG
Normal   552
LVH     188
ST       178
Name: count, dtype: int64
```

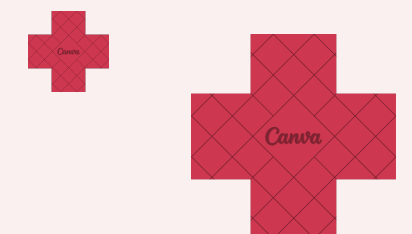


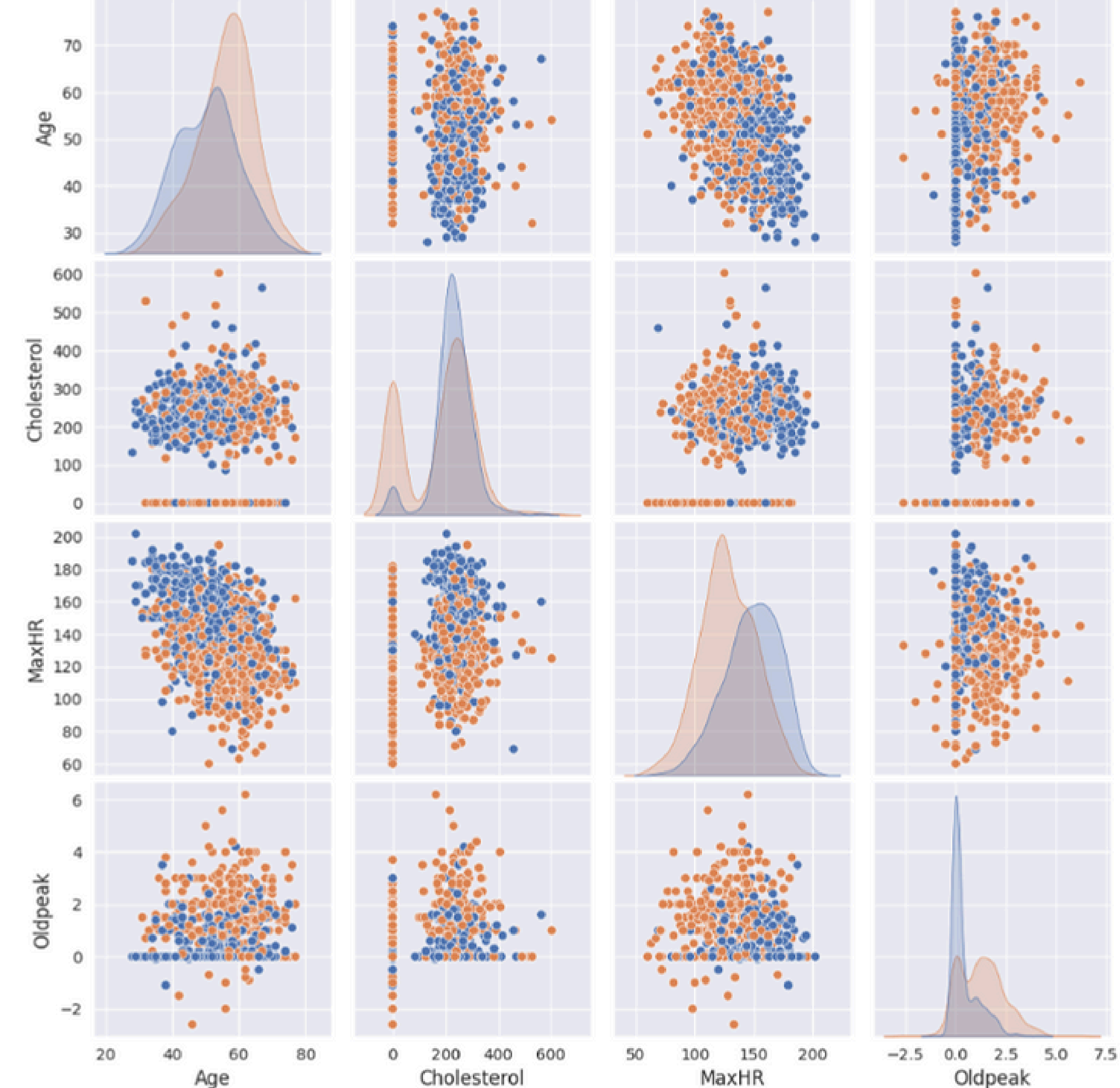




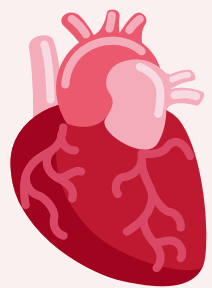


- TA (Típica angina): Dolor asociado con enfermedad cardíaca.
- ATA (Angina atípica): Dolor no relacionado con problemas cardíacos.
- NAP (Dolor no anginoso): Dolor en el pecho que no es angina.
- ASY (Asintomático): Sin dolor en el pecho.

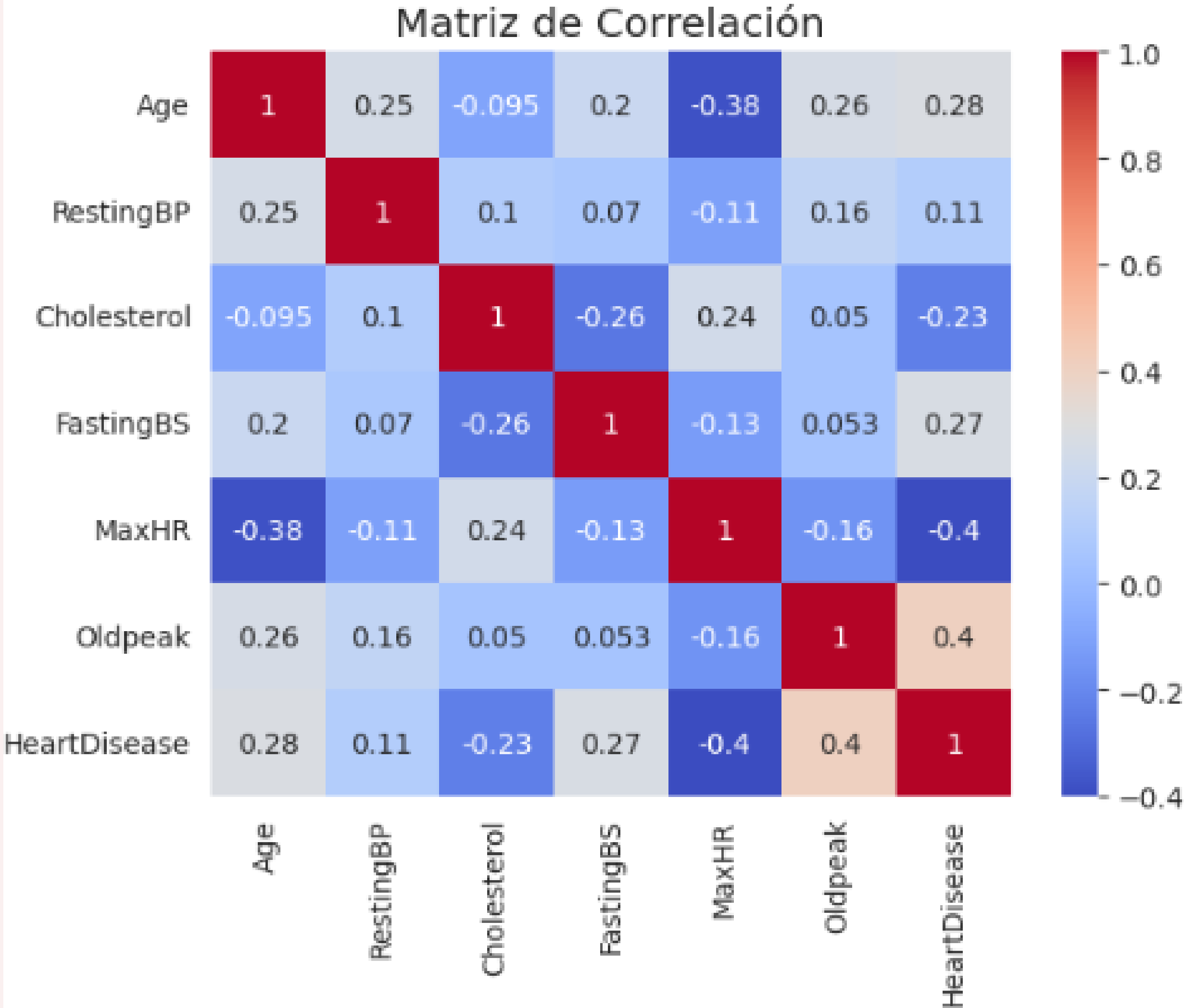


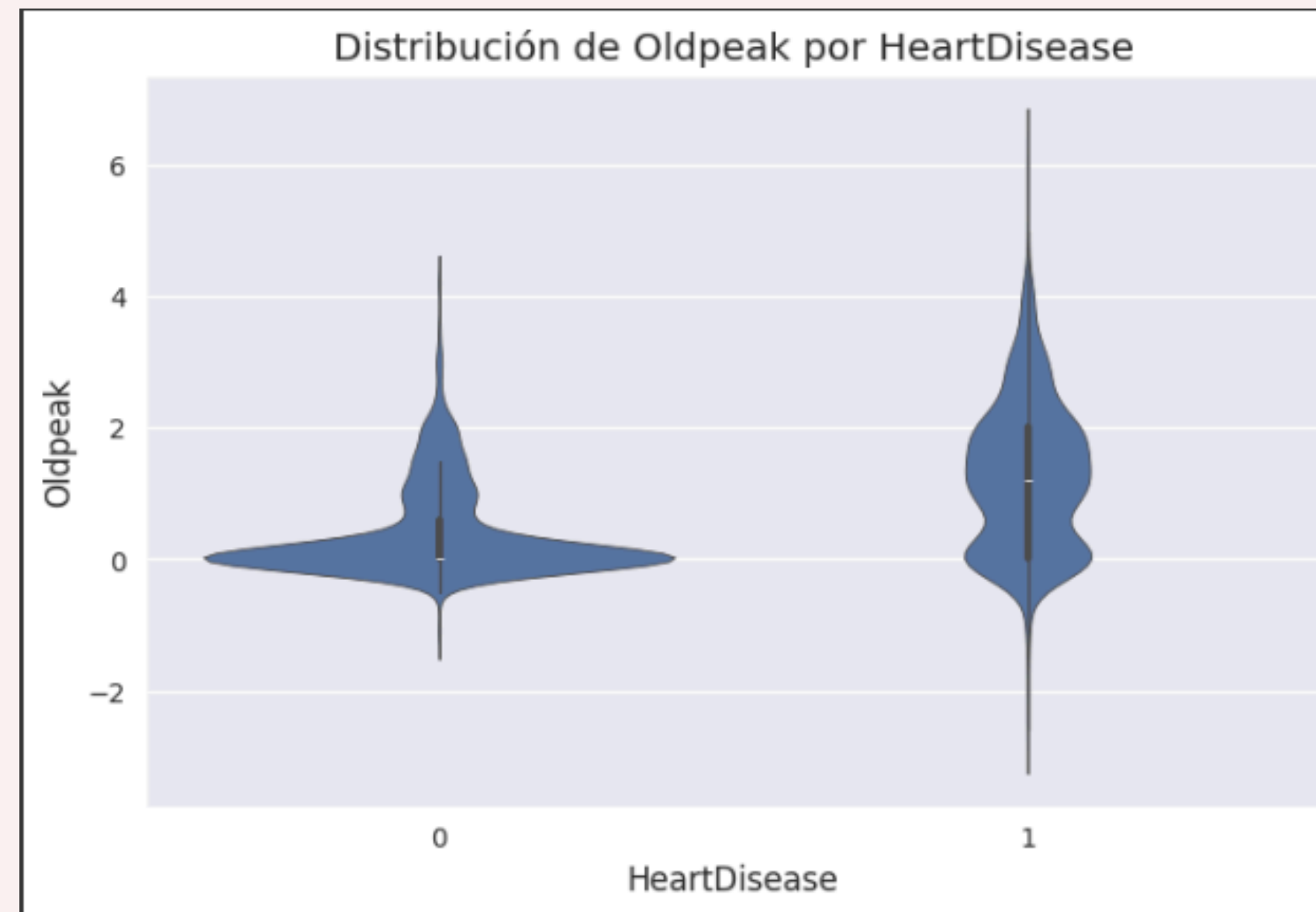
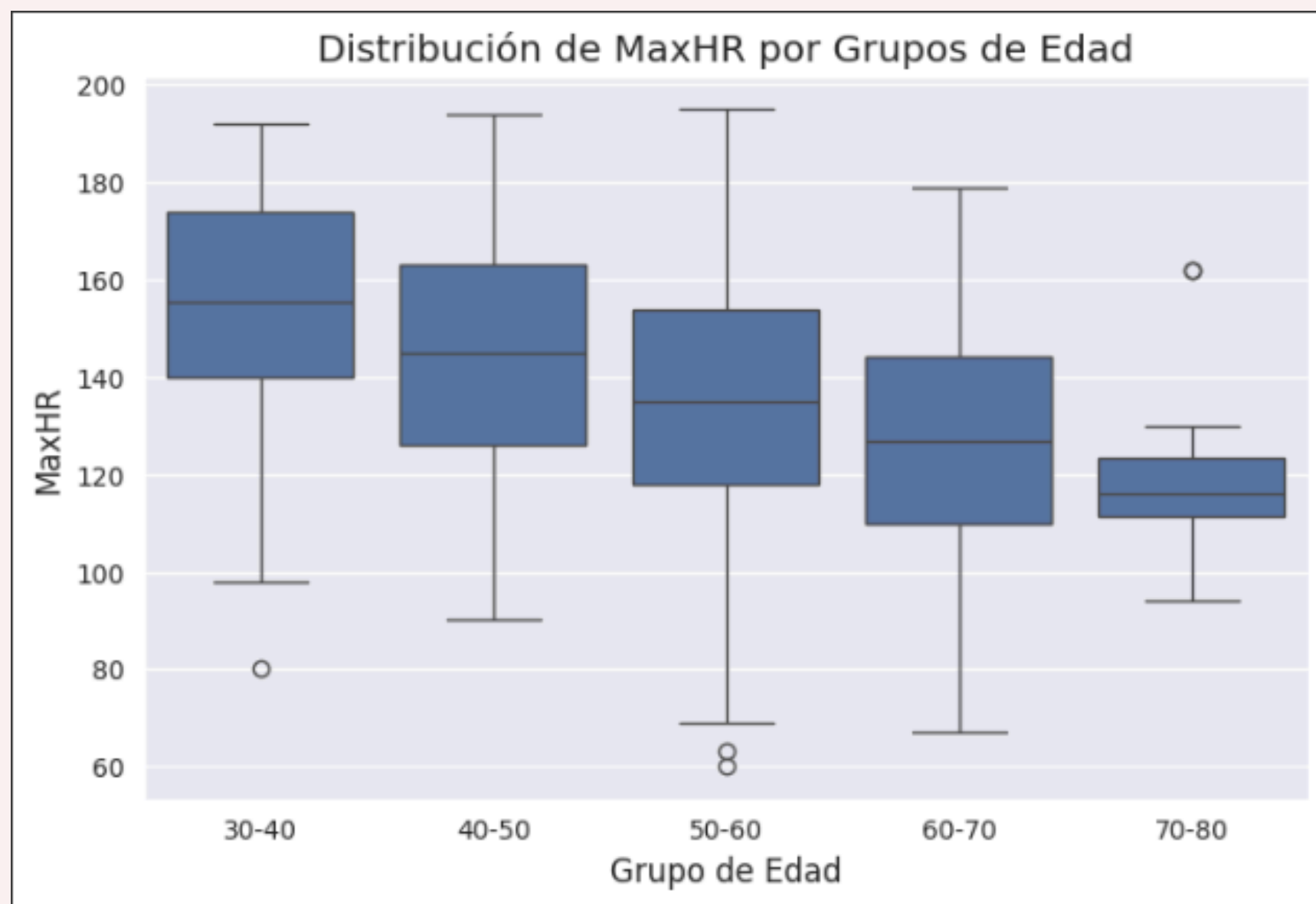


Este es un análisis multivariado, entre edad, colesterol, frecuencia cardíaca máxima alcanzada, Oldpeak (Depresión del ST o la recuperación del corazón después del ejercicio) teniendo en cuenta si la persona ha sufrido o sufre una enfermedad cardíaca o no.

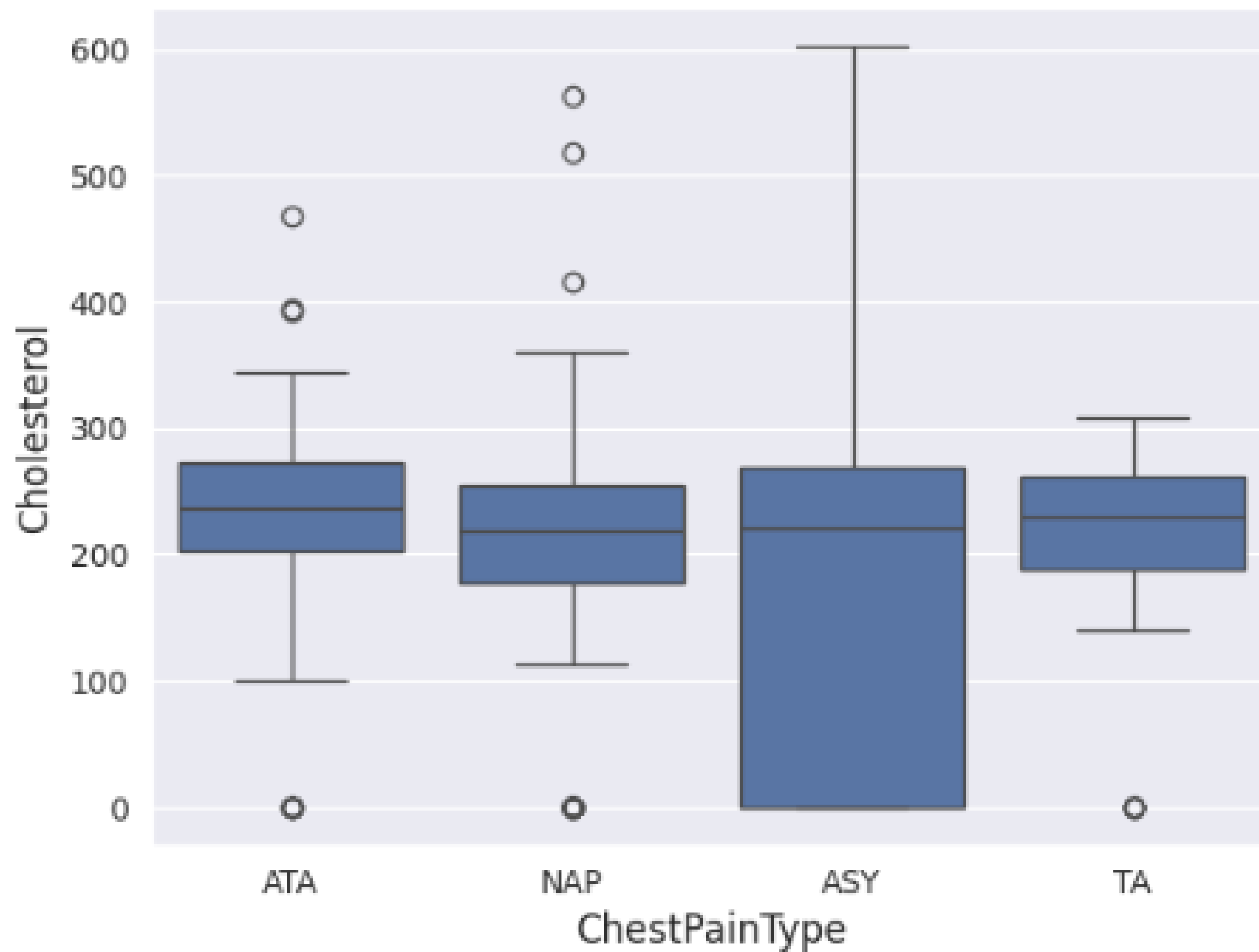


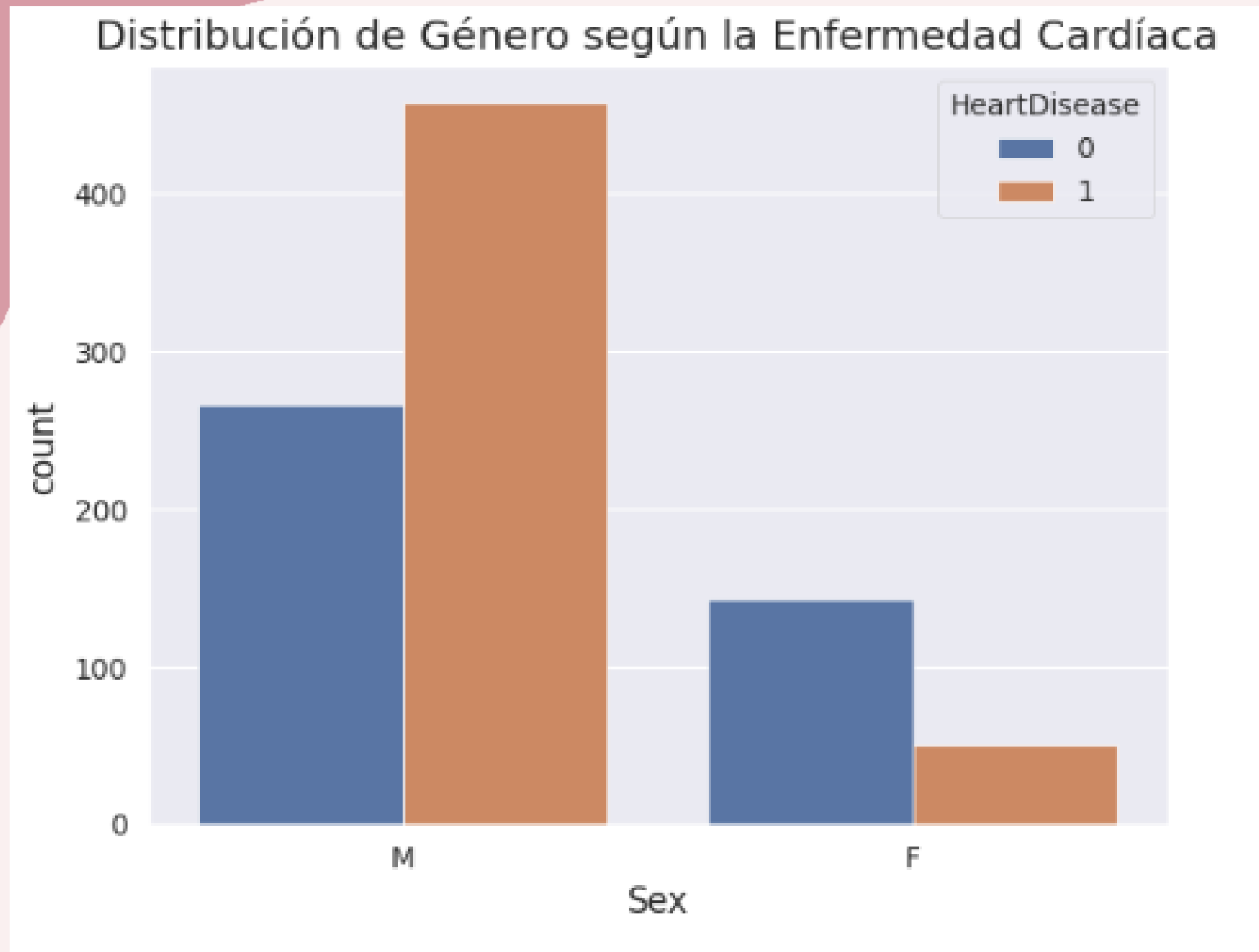
Este es una analisis en matriz de correlacion donde compara cada uno de los datos y muestra si estan directamente relacionados o no por lo que nos muestra que si un dato es relacionado su numero sera positivo si no ira al negativo , se compara entre -1 a 1





# Colesterol por Tipo de Dolor Torácico

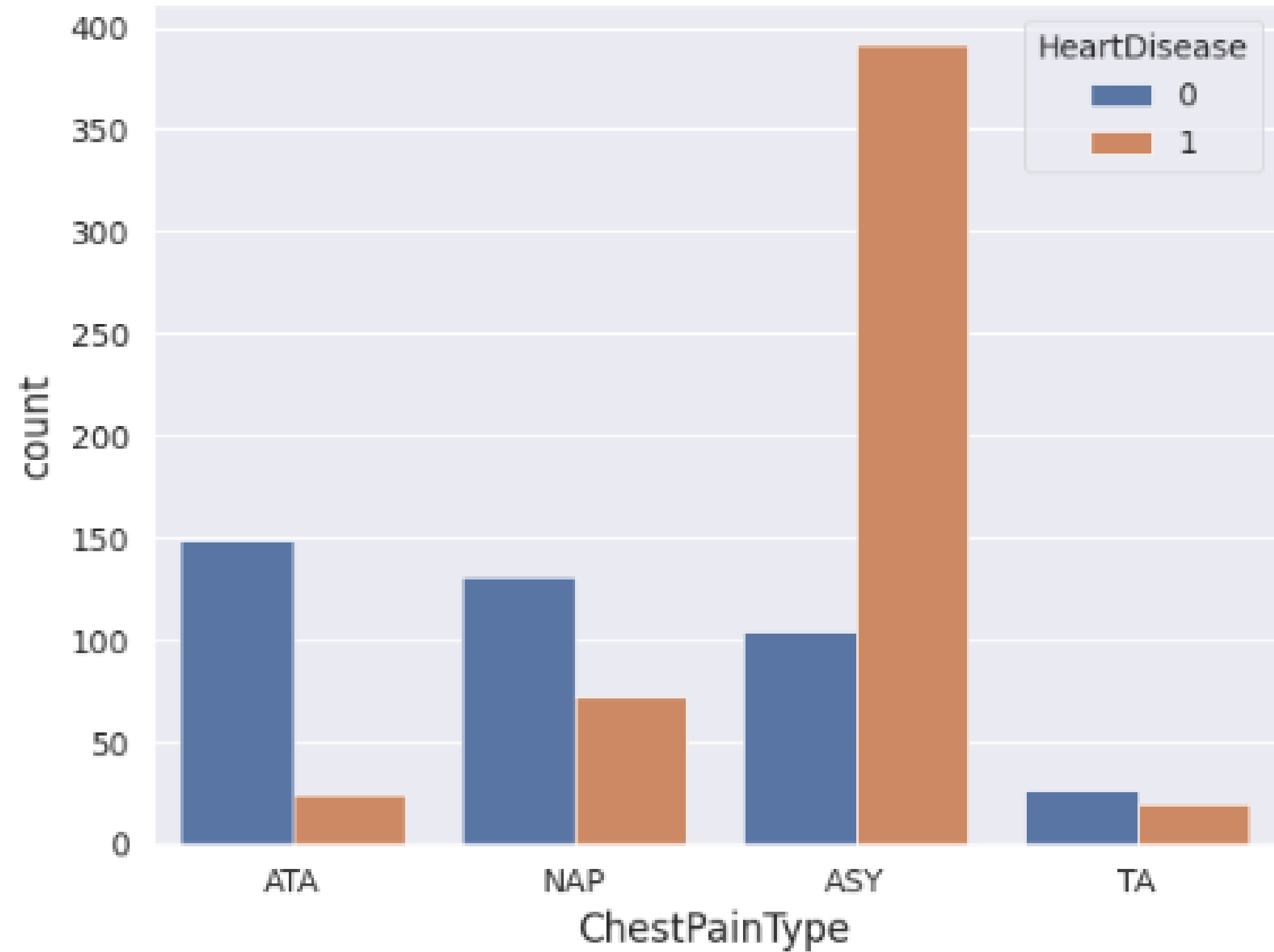




- Para esta población hay más hombres con enfermedades cardíacas que mujeres con enfermedades cardíacas y por el contrario hay más mujeres que no las tienen.

Que no se tenga dolor toracico no significa que no se tenga una enfermedad cardiaca

Distribución de Tipo de Dolor Torácico según la Enfermedad Cardíaca





# Características

**En la seleccion de características se tomaron todas, ya que a la hora de realizar la clasificacion salian mayores errores al realizar las pruebas del modelo cuando se eliminaban características que no se vieran relevantes. Por ello se dejaron todas ellas para unos mejores resultados.**

# Machine Learning

Se crearon pipelines para evaluar los hiperparametros mas optimos a evaluar para cada modelo.

```
#Eleccion del conjunto de entrenamiento y de testeo
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=42)

#Eleccion de modelos a evaluar
models = {
    "RandomForest": RandomForestClassifier(),
    "LogisticRegression": LogisticRegression(),
    "SVM": SVC(),
    "KNN": KNeighborsClassifier(),
    "XGBoost": XGBClassifier(eval_metric="logloss"),
    "NaiveBayes": GaussianNB()
}

# Pipeline para encontrar los mejores hiperparametros para los modelos
models_params = {
    "RandomForest": {
        "model": RandomForestClassifier(random_state=42),
        "params": {
            "classifier__n_estimators": [100, 200, 300],
            "classifier__max_depth": [None, 10, 20, 30]
        }
    },
    "LogisticRegression": {
        "model": LogisticRegression(max_iter=1000),
        "params": {
            "classifier__C": [0.01, 0.1, 1, 10],
            "classifier__penalty": ["l2"]
        }
    },
    "SVM": {
        "model": SVC(),
        "params": {
            "classifier__C": [0.1, 1, 10],
            "classifier__kernel": ["linear", "rbf"]
        }
    },
    "KNN": {
        "model": KNeighborsClassifier(),
        "params": {
            "classifier__n_neighbors": [3, 5, 7],
            "classifier__weights": ["uniform", "distance"]
        }
    },
}
```

```
"KNN": {
    "model": KNeighborsClassifier(),
    "params": {
        "classifier__n_neighbors": [3, 5, 7],
        "classifier__weights": ["uniform", "distance"]
    }
},
"XGBoost": {
    "model": XGBClassifier(use_label_encoder=False, eval_metric="logloss"),
    "params": {
        "classifier__n_estimators": [100, 200],
        "classifier__max_depth": [3, 5],
        "classifier__learning_rate": [0.01, 0.1]
    }
},
"NaiveBayes": {
    "model": GaussianNB(),
    "params": { "classifier__var_smoothing": [1e-9, 1e-8, 1e-7] }
}

= StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

er = []
name, mp in models_params.items():
    pipeline = Pipeline([
        ("scaler", StandardScaler()),
        ("classifier", mp["model"])
    ])

    grid = GridSearchCV(pipeline, mp["params"], cv=cv, scoring="accuracy", n_jobs=-1)
    grid.fit(X_train, y_train)

    hiper.append({
        "Model": name,
        "Best Params": grid.best_params_,
        "Best CV Accuracy": grid.best_score_
    })
})
```

# Resultados

```
SVM
classifier__C: 1
classifier__kernel: rbf
Best CV Accuracy: 0.8718

RandomForest
classifier__max_depth: None
classifier__n_estimators: 200
Best CV Accuracy: 0.8692

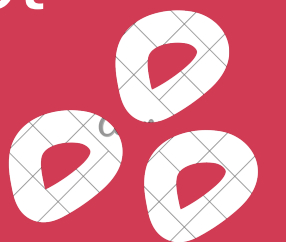
XGBoost
classifier__learning_rate: 0.01
classifier__max_depth: 3
classifier__n_estimators: 200
Best CV Accuracy: 0.8590

KNN
classifier__n_neighbors: 5
classifier__weights: uniform
Best CV Accuracy: 0.8577

NaiveBayes
classifier__var_smoothing: 1e-09
Best CV Accuracy: 0.8564

LogisticRegression
classifier__C: 0.1
classifier__penalty: l2
Best CV Accuracy: 0.8551
```

Se puede ver que el modelo SVM tiene los mejores hiperparametros para entrenamiento ya que tiene el accuracy mas alto de 87.18%, los que menor precision tuvieron fue el Naive Bayes y LogisticRegression, cabe recalcar que este resultado depende de los hiperparametros puestos para entrenamiento, y se obtienen los mejores gracias al pipeline y el modelo.



# Predicciones

```
#Creacion del diccionario que contiene los mejores hiperparametros encontrados con el g
best_models = {
    "RandomForest": RandomForestClassifier(n_estimators=200, max_depth=None),
    "LogisticRegression": LogisticRegression(C=0.1, penalty='l2', max_iter=1000),
    "SVM": SVC(C=0.1, kernel='linear'),
    "KNN": KNeighborsClassifier(n_neighbors=7, weights='distance'),
    "XGBoost": XGBClassifier(learning_rate=0.01, max_depth=3, n_estimators=200,
                           eval_metric='logloss'),
    "NaiveBayes": GaussianNB(var_smoothing=1e-9)
}
#Creacion de la pipeline que ejecutará primero el scaler y luego evaluará el modelo
pipelines = {
    name: Pipeline([
        ("scaler", StandardScaler()),
        ("classifier", model)
    ])
    for name, model in best_models.items()
}

# Ejecucion de los modelos y evaluacion de metricas para guardarlo en un dataframe de r
results = []
fig, axes = plt.subplots(2, 3, figsize=(18, 10))
axes = axes.flatten()

for idx, (name, pipeline) in enumerate(pipelines.items()):
    # Fit y predicción
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)

    # Métricas
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average='weighted', zero_division=0)
    recall = recall_score(y_test, y_pred, average='weighted', zero_division=0)

    # Cross-validation (accuracy)
    cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
    cv_scores = cross_val_score(pipeline, X_train, y_train, cv=cv, scoring='accuracy')
    cv_mean = cv_scores.mean()

    #Matriz de confusion
    cm = confusion_matrix(y_test, y_pred)

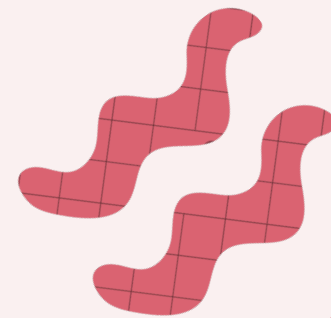
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", ax=axes[idx])
    axes[idx].set_title(f"Matriz de confusión\n{name}")
    axes[idx].set_xlabel("Predicho")
    axes[idx].set_ylabel("Actual")
```

```
# Guardar resultados
results.append({
    "Model": name,
    "Test Accuracy": acc,
    "Test Precision": prec,
    "Test Recall": recall,
    "CV Accuracy": cv_mean
})

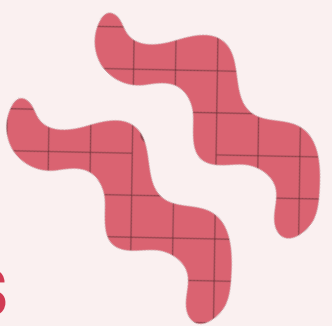
for j in range(idx + 1, len(axes)):
    axes[j].axis('off')

plt.subplots_adjust(hspace=0.4, wspace=0.3)

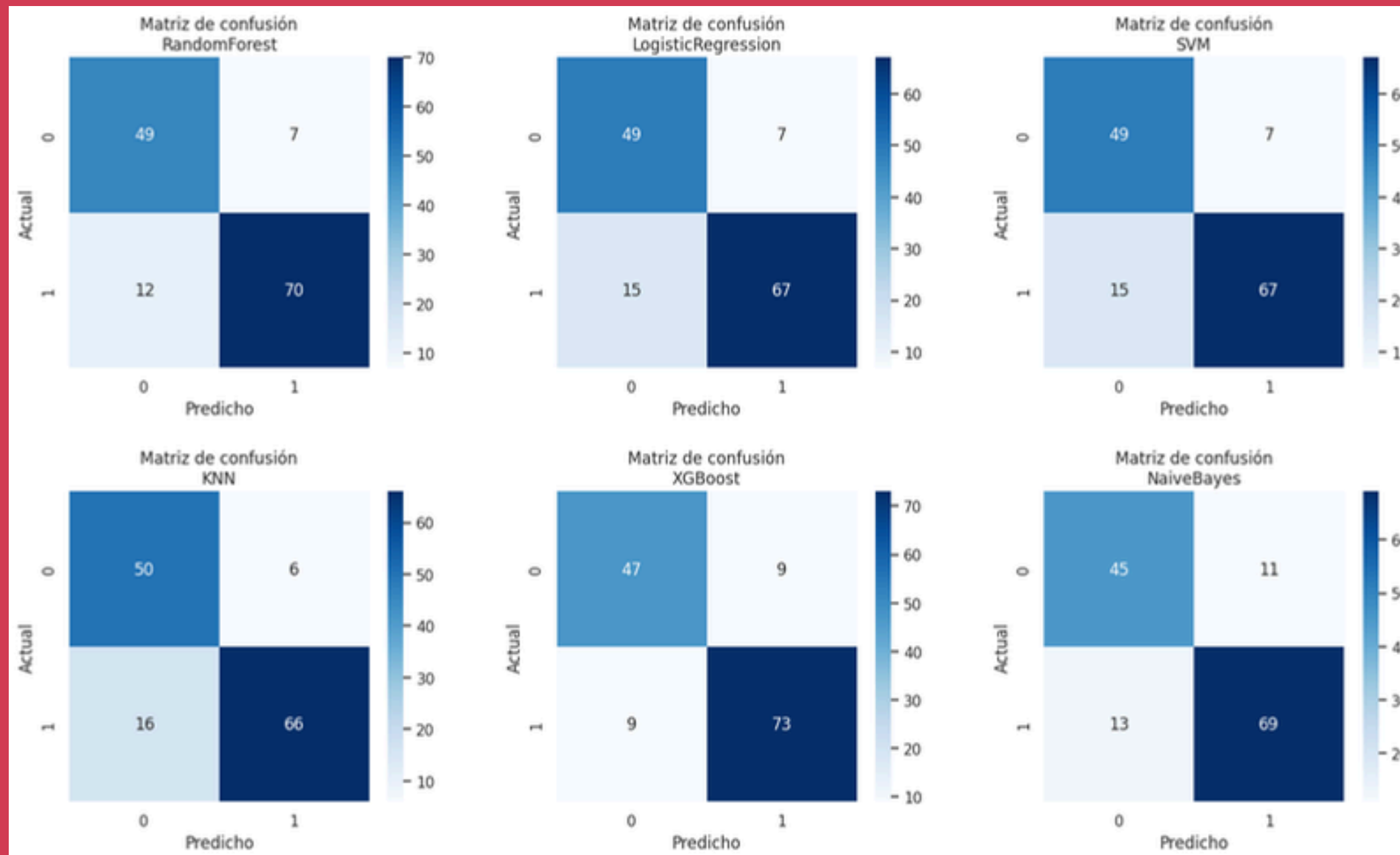
# Crear DataFrame de los resultados para una mejor visualizacion
results_df = pd.DataFrame(results).sort_values(by="Test Accuracy", ascending=False)
```



Ahora que se sabe los mejores hiperparametros se realizaron las pipelines y con estas las predicciones o testeos respectivos para cada modelo para obtener el accuracy, precision y recall de las pruebas. Ademas del Cross validation accuracy y la matriz de confusion de cada modelo.



# Resultados



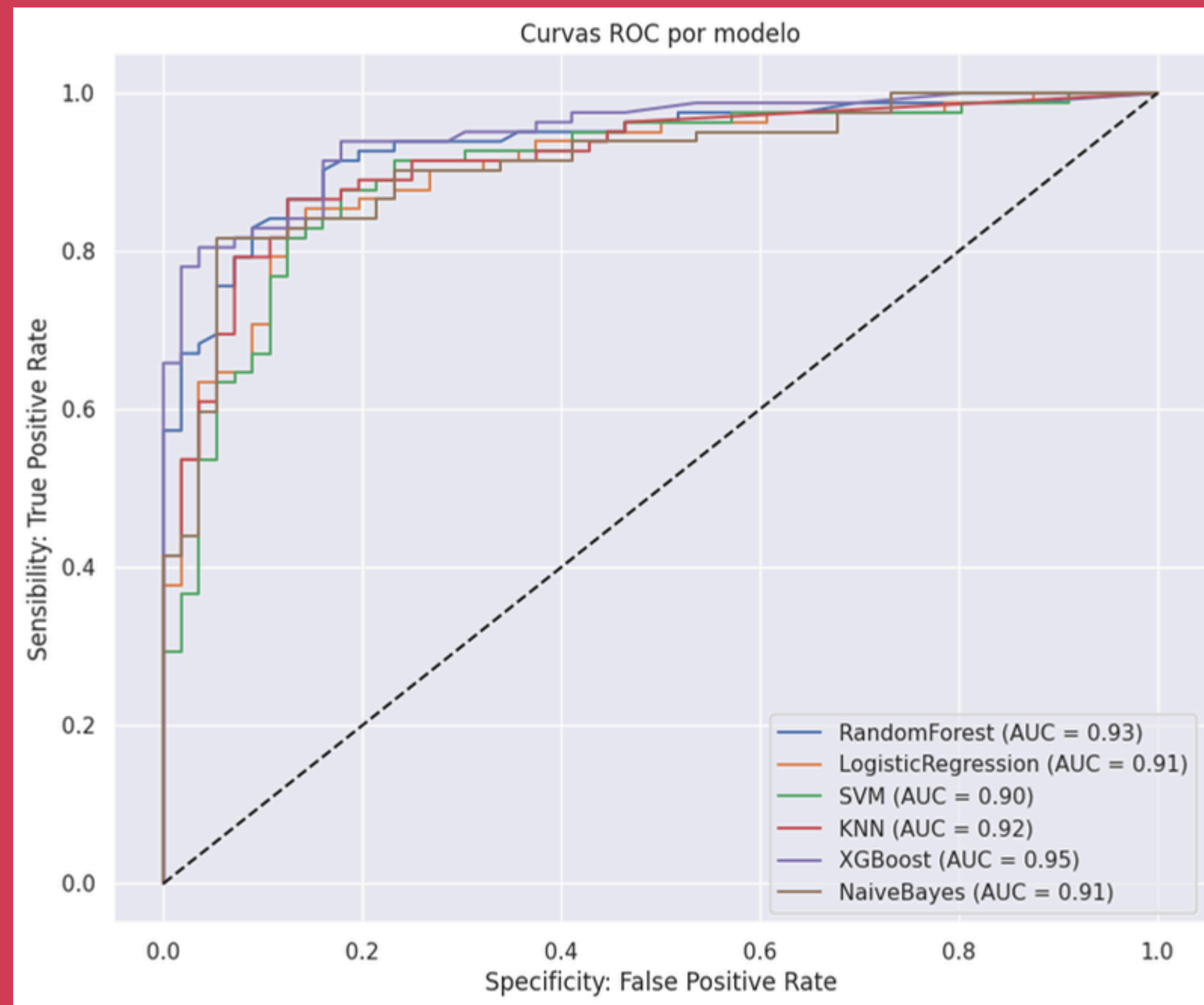
Se pudo ver con los resultados que el modelo XGBoost tiene la menor cantidad de errores (18) con el accuracy mas alto de 86.02%, Ademas se puede ver que el RandomForest es muy balanceado por lo que es bastante robusto y tiene buen rendimiento global.

	Model	Test Accuracy	Test Precision	Test Recall	CV Accuracy
4	XGBoost	0.869565	0.869565	0.869565	0.860256
0	RandomForest	0.862319	0.866153	0.862319	0.857692
1	LogisticRegression	0.840580	0.848683	0.840580	0.856410
2	SVM	0.840580	0.848683	0.840580	0.853846
3	KNN	0.840580	0.852108	0.840580	0.852564
5	NaiveBayes	0.826087	0.827343	0.826087	0.852564

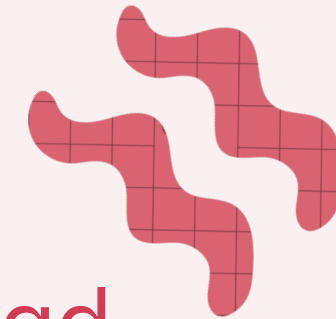
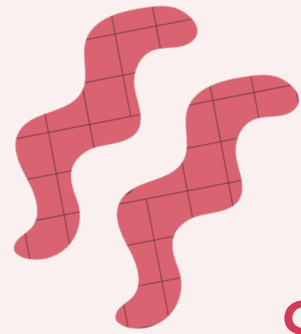


# Resultados

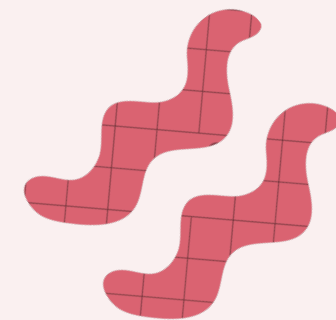
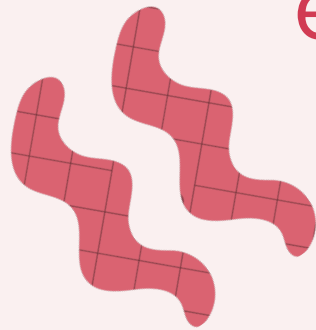
Se generaron estas curvas ROC, las cuales muestran cómo varía el rendimiento del modelo según el umbral de decisión, la sensibilidad y especificidad. Además, se presenta el área bajo la curva (AUC), donde un valor de 0.5 equivale a una predicción aleatoria. Por esto, los modelos más robustos son XGBoost y RandomForest, al presentar valores de AUC superiores a 0.93.



# Deep Learning



Se crearon 3 Modelos con distintas cantidad de capas para saber cual de estas es mejor a la hora de detectar pacientes que sufran de enfermedades o insuficiencias cardiacas.



# Modelo 1

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
from tensorflow.keras.layers import BatchNormalization

model = Sequential([
    Dense(256, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(512, activation='relu'),
    Dropout(0.05),
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(128, activation='relu'),
    Dropout(0.1),
    Dense(2, activation='softmax')
])
model.summary()
model.compile(optimizer=Adam(learning_rate = 0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

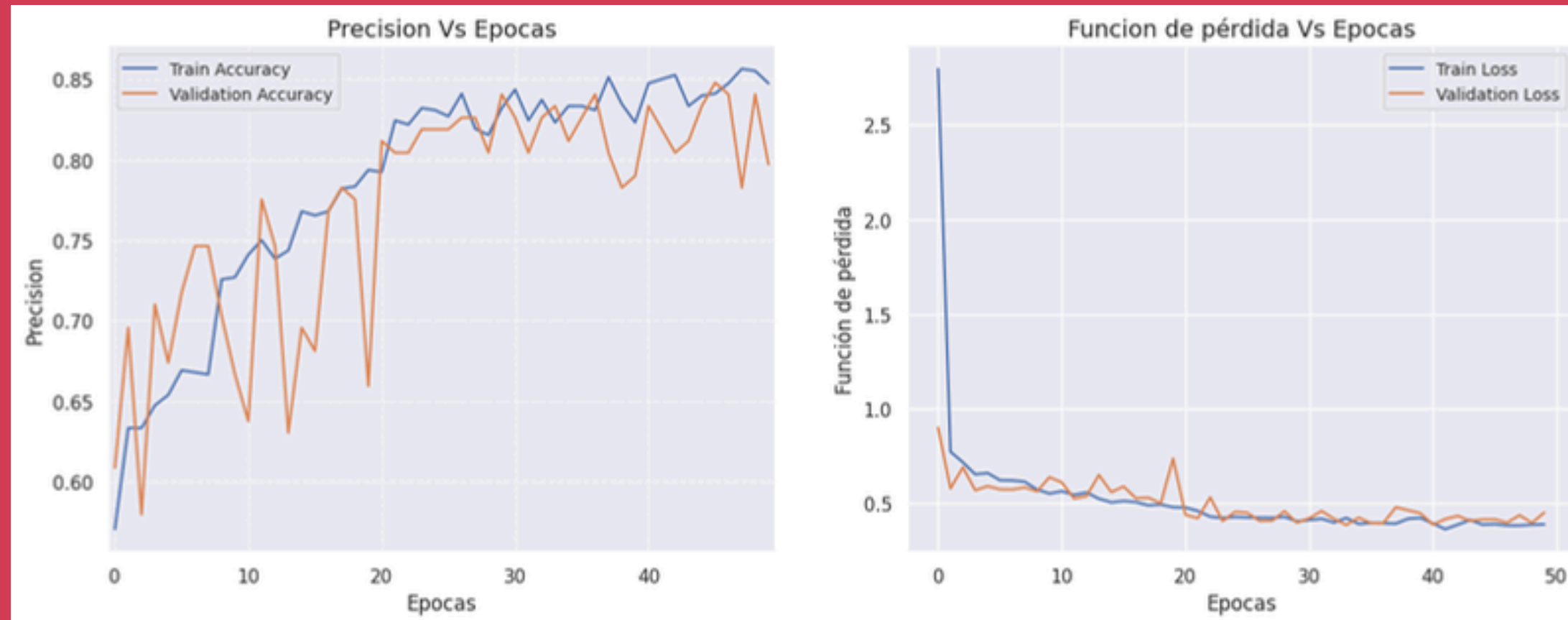
Layer (type)	Output Shape	Param #
dense_27 (Dense)	(None, 256)	3,072
dense_28 (Dense)	(None, 512)	131,584
dropout_19 (Dropout)	(None, 512)	0
dense_29 (Dense)	(None, 256)	131,328
dropout_20 (Dropout)	(None, 256)	0
dense_30 (Dense)	(None, 128)	32,896
dropout_21 (Dropout)	(None, 128)	0
dense_31 (Dense)	(None, 2)	258

Total params: 299,138 (1.14 MB)  
Trainable params: 299,138 (1.14 MB)  
Non-trainable params: 0 (0.00 B)

Este modelo posee 5 capas densas las cuales van de esta manera 256→512→256→128 (unidades). Además realiza dropouts para que no halla overfitting a la hora de validar nuevos datos y usa como funcion de activacion la relu.



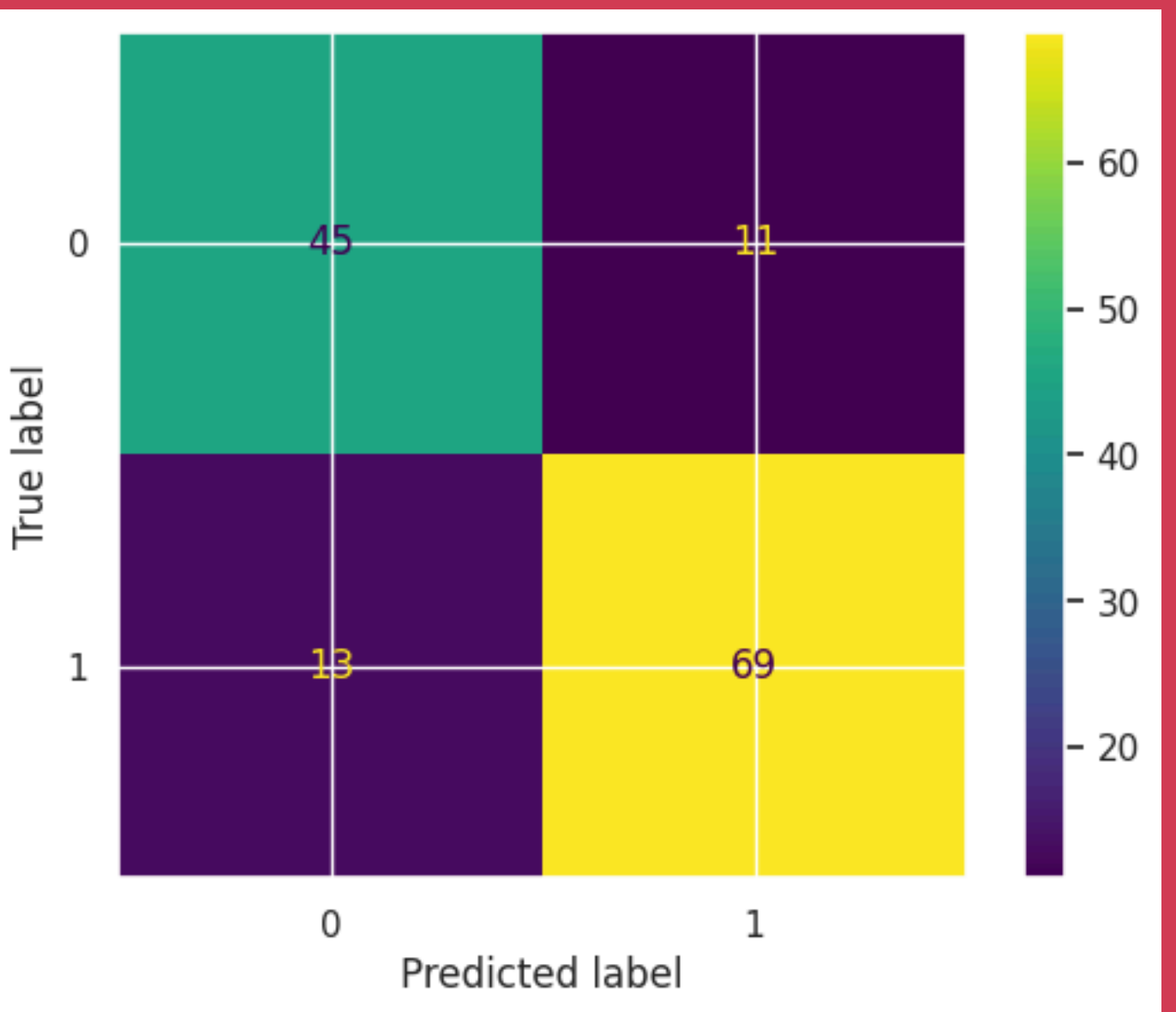
# Resultados



- En la Grafica de precision vs Epocas se observa una tendencia general creciente entre el train accuracy y el validation accuracy, esto nos muestra que el modelo esta aprendiendo a clasificar correctamente con ciertos errores.
- En la Grafica de perdida vs Epocas, la perdida de entrenamiento y validacion sigue un comportamiento similar, donde disminuyen con pequeñas oscilaciones. Esto nos muestra que en las oscilaciones hubo equivocaciones leves. Por lo tanto el modelo funciona correctamente

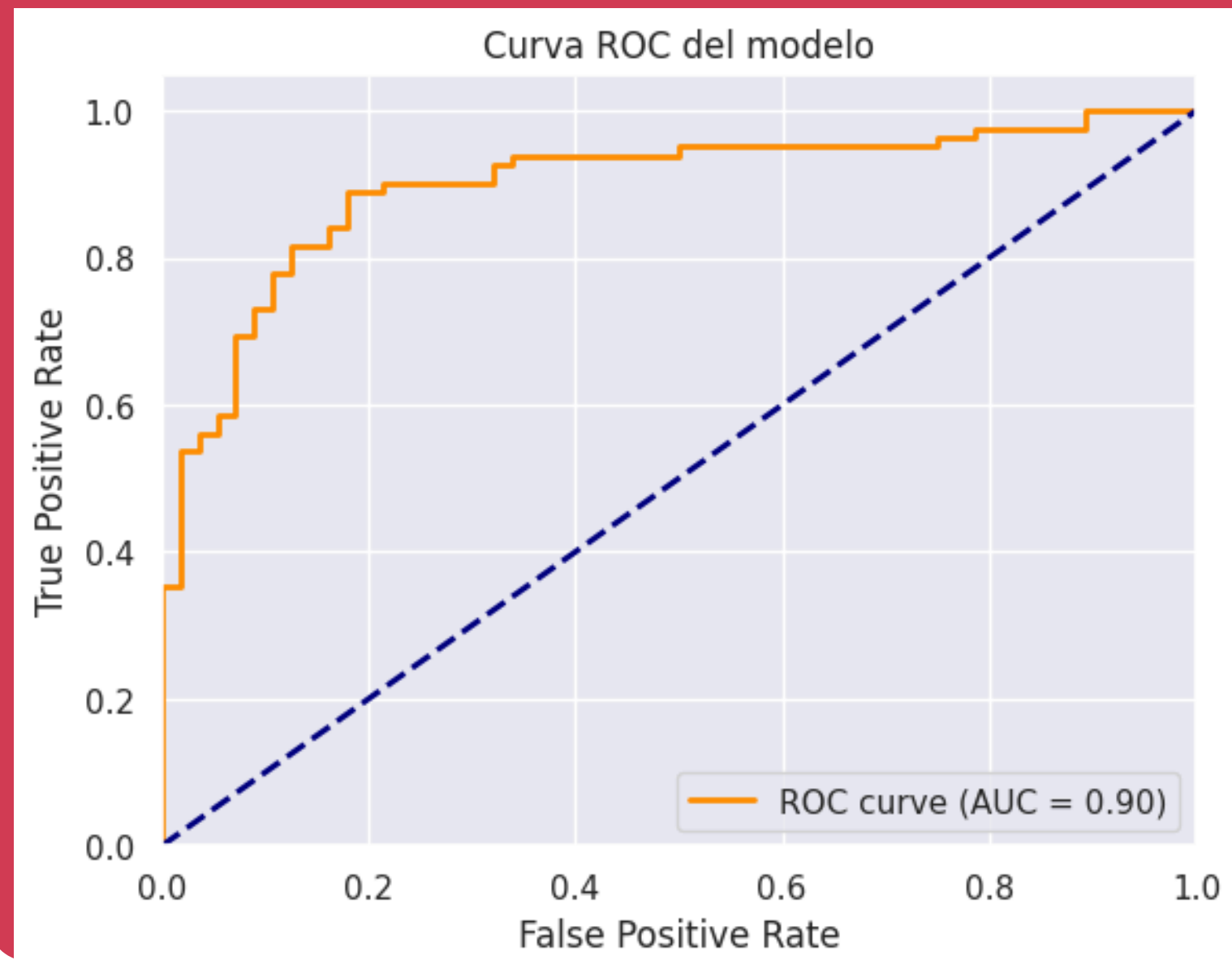
# Resultados

	precision	recall	f1-score	support
0.0	0.78	0.80	0.79	56
1.0	0.86	0.84	0.85	82
accuracy			0.83	138
macro avg	0.82	0.82	0.82	138
weighted avg	0.83	0.83	0.83	138



- Observando las gráficas, se clasificaron correctamente 111 datos(personas), tanto en verdaderos positivos como en verdaderos negativos. Sin embargo, se presentaron 24 errores de clasificación: 13 correspondieron a falsos negativos y 11 a falsos positivos. Esto significa que esas 13 personas fueron clasificadas como sanas cuando en realidad no lo estaban.
- Viendo el Classification report se ve que tiene mejor precision la clase 1 con 0,86 , pero recall muy parecidos tanto en la clase 1 como en la 0.

# Resultados



Vemos que el area bajo la curva del modelo es de 0,90 lo que indica que tiene una buena prediccion a la hora de detectar las clases, en otro sentido muestra en gran medida verdaderos positivos y falsos positivos, pero al ser de 0,90 nos muestra que hay errores en las predicciones, pero se entiende que el modelo posee una alta capacidad de discriminacion de clases.

# Modelo 2

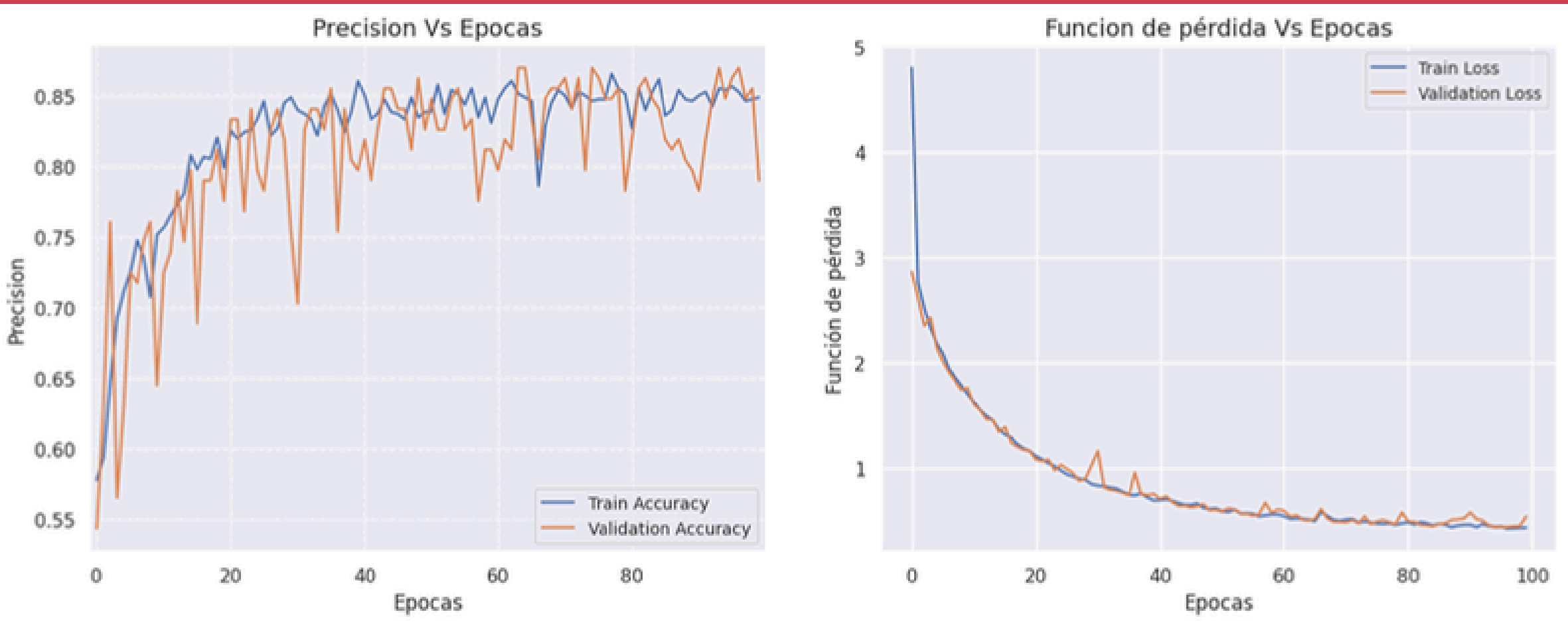
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.regularizers import l2

model = Sequential([
    Dense(1024, activation='relu', input_shape=X_train.shape[1:], kernel_regularizer=l2(0.001)),
    Dense(2048, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.1),
    Dense(1024, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.2),
    Dense(512, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.15),
    Dense(256, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.1),
    Dense(128, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.1),
    Dense(64, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.1),
    Dense(2, activation='softmax')
])
model.summary()
model.compile(optimizer=Adam(learning_rate = 0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Layer (type)	Output Shape	Param #
dense_55 (Dense)	(None, 1024)	10,240
dense_56 (Dense)	(None, 2048)	2,099,200
dropout_37 (Dropout)	(None, 2048)	0
dense_57 (Dense)	(None, 1024)	2,098,176
dropout_38 (Dropout)	(None, 1024)	0
dense_58 (Dense)	(None, 512)	524,800
dropout_39 (Dropout)	(None, 512)	0
dense_59 (Dense)	(None, 256)	131,328
dropout_40 (Dropout)	(None, 256)	0
dense_60 (Dense)	(None, 128)	32,896
dropout_41 (Dropout)	(None, 128)	0
dense_61 (Dense)	(None, 64)	8,256
dropout_42 (Dropout)	(None, 64)	0
dense_62 (Dense)	(None, 2)	130

Este modelo posee 7 capas densas las cuales van de esta manera 1024→2048→1024→512→256→128→64 (unidades). Además realiza dropouts y utiliza un Kernel l2 = 0.001 el cual penaliza el valor de los pesos grandes en la función pérdida, esto es para que no haya overfitting a la hora de validar nuevos datos y usa como función de activación la relu.

# Resultados

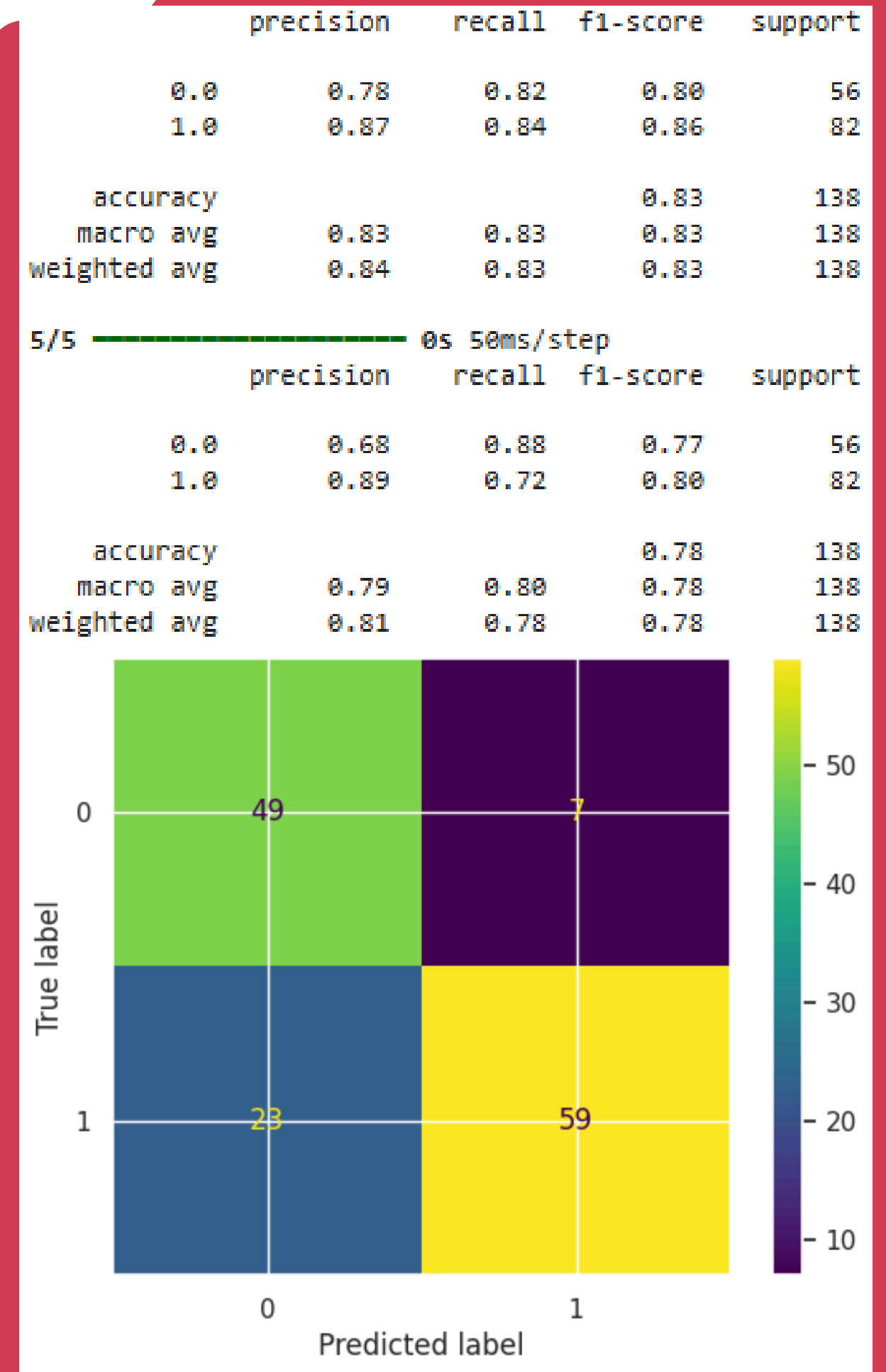


- En la Grafica de precision vs Epocas se observa una tendencia general creciente entre el train accuracy y el validation accuracy, cabe recalcar la variabilidad en la precision, pero despues de la epoca 30 se ven mas parecidas.
- En la Grafica de perdida vs Epocas, la perdida de entrenamiento y validacion disminuyen progresivamente sin muchas fluctuaciones. y se asemejan mucho el train loss y validation loss.



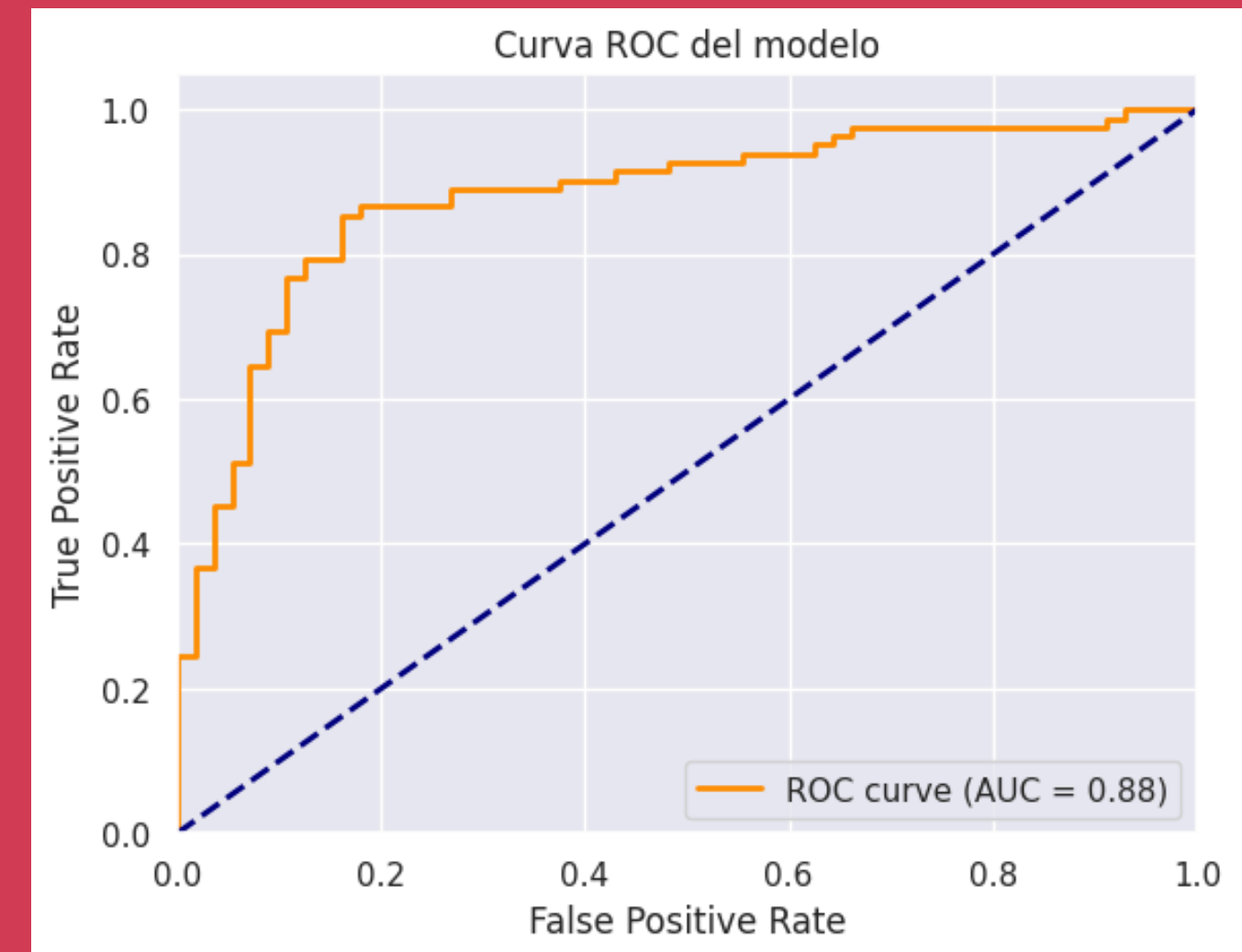
# Resultados

- Observando las gráficas, se clasificaron correctamente 108 datos(personas), tanto en verdaderos positivos como en verdaderos negativos. Sin embargo, se presentaron 30 errores de clasificación: 23 correspondieron a falsos negativos y 7 a falsos positivos. Esto significa que esas 23 personas fueron clasificadas como sanas cuando en realidad no lo estaban. El modelo tiende a ser más conservador con las predicciones positivas.
- Viendo el Classification report las metricas podemos ver como la clase 0 tiene un recall de 0,88 pero una baja precision de 0,68 y en la clase 1 una precision de 0,89 con un recall de 0,72 por lo que se comprueba lo que aparece en la matriz.



# Resultados

Vemos que el area bajo la curva del modelo es de 0,88 lo que indica que tiene una prediccion buena a la hora de detectar las clases, en otro sentido muestra en gran medida verdaderos positivos y falsos positivos, pero el modelo puede tener errores ya que se aleja un poco del 1,0 y ya es por debajo de 0.90 que seria menos de lo ideal



# Modelo 3

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
from tensorflow.keras.layers import BatchNormalization
model = Sequential([
    Dense(512, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(1024, activation='relu'),
    Dropout(0.05),
    Dense(512, activation='relu'),
    Dropout(0.2),
    Dense(256, activation='relu'),
    Dropout(0.1),
    Dense(512, activation='relu'),
    Dropout(0.1),
    Dense(128, activation='relu'),
    Dropout(0.1),
    Dense(2, activation='softmax')
])
model.summary()
model.compile(optimizer=Adam(learning_rate = 0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

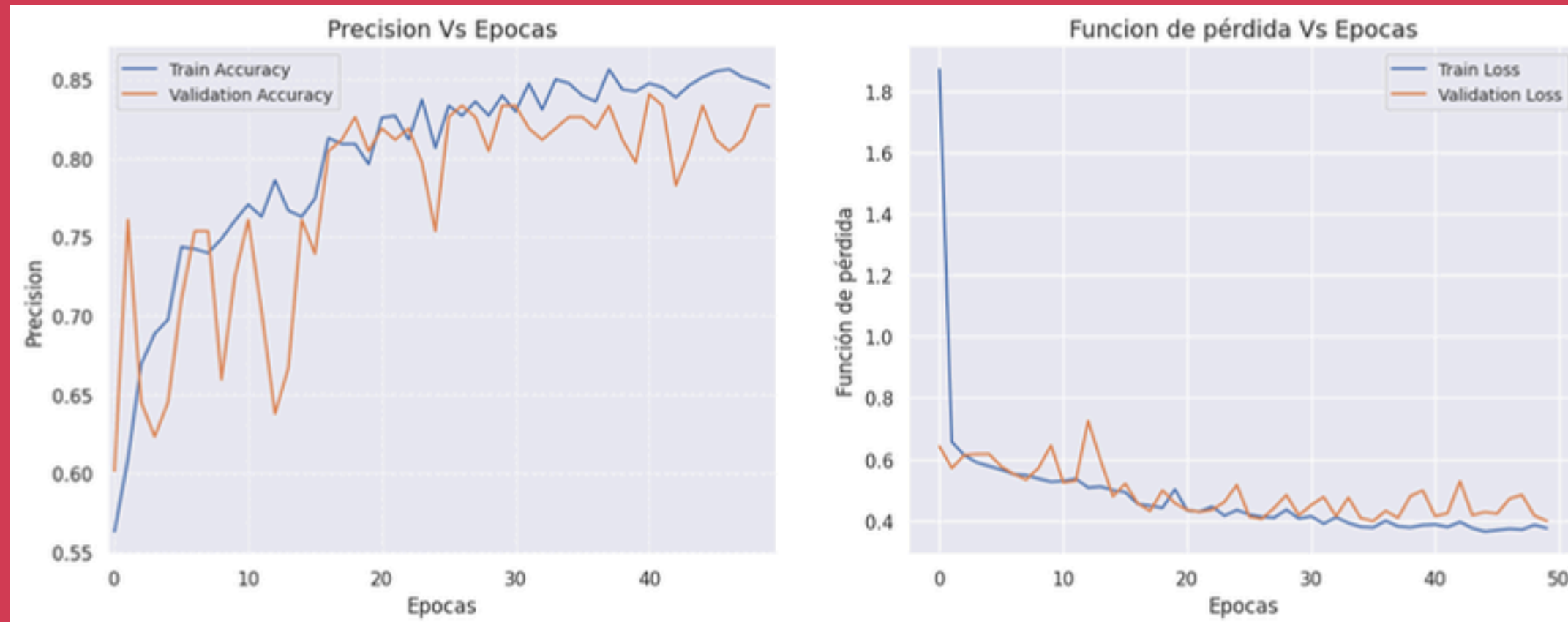
Layer (type)	Output Shape	Param #
dense_107 (Dense)	(None, 512)	6,144
dense_108 (Dense)	(None, 1024)	525,312
dropout_73 (Dropout)	(None, 1024)	0
dense_109 (Dense)	(None, 512)	524,800
dropout_74 (Dropout)	(None, 512)	0
dense_110 (Dense)	(None, 256)	131,328
dropout_75 (Dropout)	(None, 256)	0
dense_111 (Dense)	(None, 512)	131,584
dropout_76 (Dropout)	(None, 512)	0
dense_112 (Dense)	(None, 128)	65,664
dropout_77 (Dropout)	(None, 128)	0
dense_113 (Dense)	(None, 2)	258

Total params: 1,385,090 (5.28 MB)  
Trainable params: 1,385,090 (5.28 MB)  
Non-trainable params: 0 (0.00 B)

Este modelo posee 6 capas densas las cuales van de esta manera 512→1024→512→256→512→128 (unidades). Además realiza dropouts el cual penaliza el valor de los pesos grandes en la función pérdida, esto es para que no haya overfitting a la hora de validar nuevos datos y usa como función de activación la relu. El problema es su estructura que puede generar cuello de botella



# Resultados

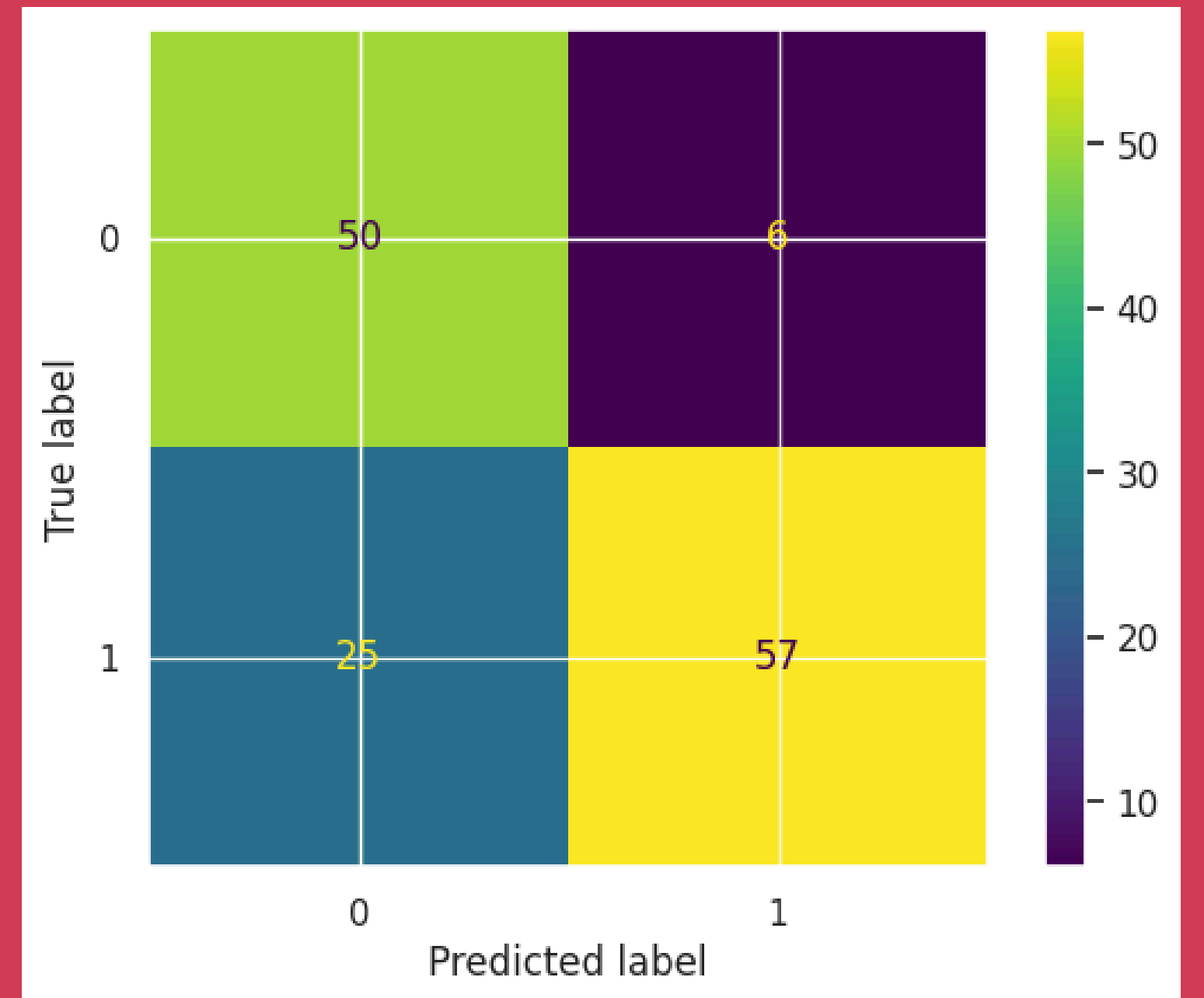


- En la Grafica de precision vs Epocas se observa una tendencia general creciente entre el train accuracy y el validation accuracy, cabe recalcar que hay mucha variabilidad antes de la epoca 20 despues se regula pero aun pueden haber errores.
- En la Grafica de perdida vs Epocas, la perdida de entrenamiento y validacion disminuyen progresivamente y hay fluctuaciones pero no tan grandes. asi que pueden haber equivocaciones

# Resultados

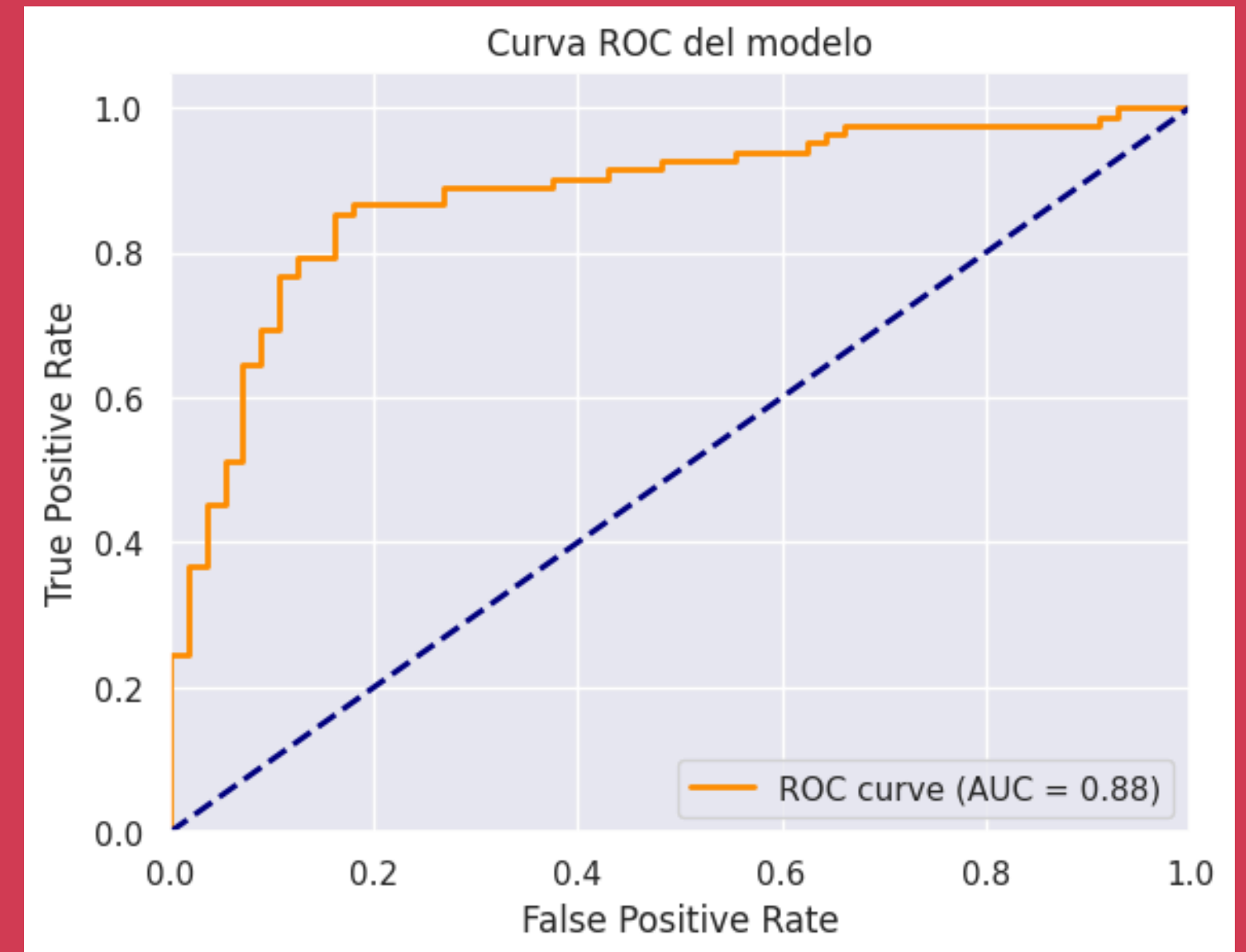
- Observando las gráficas, se clasificaron correctamente 107 datos(personas), tanto en verdaderos positivos como en verdaderos negativos. Sin embargo, se presentaron 31 errores de clasificación: 25 correspondieron a falsos negativos y 6 a falsos positivos. Esto significa que esas 25 personas fueron clasificadas como sanas cuando en realidad no lo estaban.
- Viendo el Classification report ,las metricas, podemos ver como la clase 0 tiene un recall de 0,89 pero una baja precision de 0,67 y en la clase 1 una precision de 0,90 con un recall de 0,70 por lo que se comprueba lo que aparece en la matriz.

	precision	recall	f1-score	support
0.0	0.67	0.89	0.76	56
1.0	0.90	0.70	0.79	82
accuracy			0.78	138
macro avg	0.79	0.79	0.77	138
weighted avg	0.81	0.78	0.78	138



# Resultados

Vemos que el area bajo la curva del modelo es de 0,88 lo que indica que tiene una prediccion buena pero no la mejor, en otro sentido muestra en gran medida verdaderos positivos y falsos positivos, el problema es si se genera un cuello de botella el cual causaria muchos errores graves.





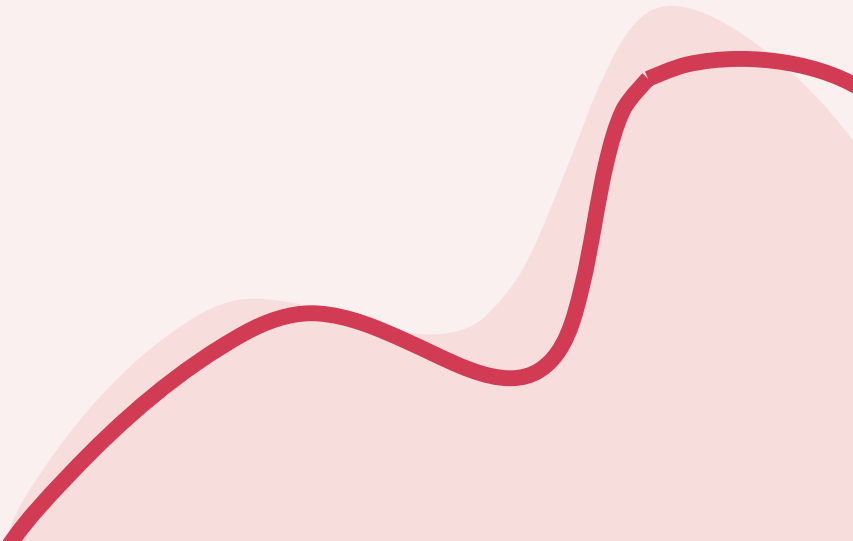
# Conclusiones



- El primer modelo ofrece un buen equilibrio entre precisión y recall, lo cual es crucial si los falsos negativos son costosos o pueden causar la muerte. La curva ROC con  $AUC = 0.90$  confirma una buena capacidad de discriminación. Su predicción de los valores tiende a tener una mayor std al igual que su función de pérdida

- El segundo modelo presenta resultados ligeramente mejores que el primer modelo respecto a la precisión, recall, pérdida y predicción. Mostrando que se equivoca un poco menos en la clasificación. Esto podría ser útil si se quiere evitar falsos negativos a toda costa.

Finalmente los 2 primeros modelos son buenos y realizan de manera satisfactoria la predicción de una enfermedad cardíaca

- El tercer modelo no mejora respecto a los anteriores y contiene un cuello de botella, debido a la arquitectura, que posiblemente no es compatible con la clasificación binaria. Lo cual es crítico pues se puede estar perdiendo información vital al hacer el cuello de botella.
- 



**Muchas Gracias**