**ENTREGA FINAL**

**Problema y su contexto:** El problema que el modelo predictivo busca resolver es predecir la supervivencia de los pacientes con insuficiencia cardíaca en función de sus características clínicas, corporales y de estilo de vida. El objetivo es desarrollar un modelo que, con base en estos datos, pueda determinar si un paciente tiene mayor probabilidad de sobrevivir o no dentro del periodo de seguimiento.

Este tipo de modelo ayudaría a los médicos a identificar a los pacientes con mayor riesgo de mortalidad y a priorizar intervenciones o tratamientos específicos para mejorar su pronóstico. Además, busca identificar las características más influyentes en la predicción de la supervivencia, lo que podría guiar futuras investigaciones y mejorar el manejo clínico de estos pacientes.

**Pregunta de negocio:** ¿Cuál es la relación que existe entre ciertas características médicas (fumar, enfermedades prexistentes) de un paciente y sufrir muerte por insuficiencia cardíaca?

**Alcance del proyecto:** El objetivo de este proyecto es desarrollar un modelo de machine learning para predecir la supervivencia de pacientes con insuficiencia cardíaca, utilizando datos del dataset 'Heart Failure Clinical Records', que incluye características clínicas, corporales y de estilo de vida.

El proyecto comenzará con una exploración de datos y un preprocesamiento que incluirá la limpieza y normalización de la información. Luego, se implementarán diversos algoritmos de machine learning, como regresión logística y Random Forest, evaluando su rendimiento mediante validación cruzada para seleccionar el modelo más efectivo.

Se espera que el modelo permita clasificar a los pacientes según su probabilidad de supervivencia e identificar las características más influyentes en este resultado. Esto ayudará a los médicos a identificar pacientes en riesgo y priorizar intervenciones. Sin embargo, se reconoce que el modelo es una herramienta complementaria y no sustituye el juicio clínico.

**Descripción de conjuntos de datos a emplear:** El conjunto de datos contiene los registros médicos de 299 pacientes con insuficiencia cardíaca, recolectados en dos hospitales de Faisalabad, Pakistán, durante 2015. El dataset incluye 13 características que abarcan información clínica, corporal y de estilo de vida, como la presencia de anemia, hipertensión, diabetes, los niveles de creatinina y sodio en sangre, y la fracción de eyección del corazón.

El propósito principal del dataset es predecir, mediante machine learning, el riesgo de sufrir insuficiencia cardíaca de los pacientes y determinar cuáles son las características más importantes que influyen en este resultado. La variable objetivo es un evento de muerte (si el paciente murió o sobrevivió antes del final del período de seguimiento).

El conjunto de datos contiene diversas variables que son fundamentales para analizar y predecir la probabilidad de supervivencia de pacientes con insuficiencia cardíaca.

El data sheart\_failure\_clinical\_records\_dataset.csv es propocionado por Heart Failure Clinical Records, este conjunto de datos está licenciado bajo una licencia Creative Commons Attribution 4.0 International (CC BY 4.0). Esto permite compartir y adaptar los conjuntos de datos para cualquier propósito, siempre que se otorgue el crédito correspondiente.

**Primeros modelos:**

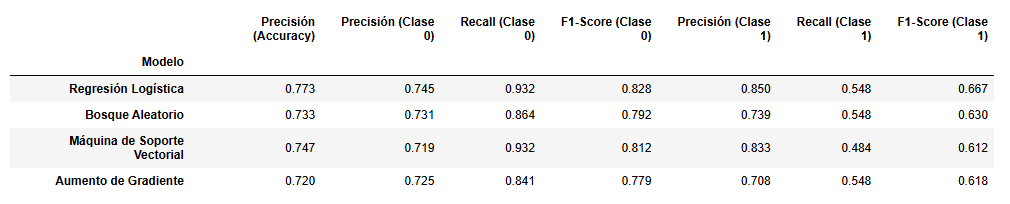
Se realiza un análisis de predicción para eventos de muerte utilizando varios modelos de clasificación: Regresión Logística, Random Forest, SVM y Gradient Boosting, .

**1. Regresión Logística:** Con los mejores hiperparámetros (C=1.0 y solver='liblinear'), obtuvo un accuracy de 77.33%. Su desempeño en términos de métricas muestra un f1-score de 0.83 para la clase 0 (no fallecimiento) y de 0.67 para la clase 1 (fallecimiento). La precisión para la clase 0 fue de 75%, mientras que para la clase 1 fue de 85%, lo que indica que es mejor identificando correctamente los casos de fallecimiento que los no fallecimientos, aunque con menor recall en la clase 1 (55%), sugiriendo que muchos casos positivos fueron clasificados incorrectamente como negativos.

**2. Random Forest:** Con hiperparámetros optimizados (max\_depth=10 y n\_estimators=100), alcanzó un accuracy de 73.33%. Para la clase 0, el modelo obtuvo un f1-score de 0.79, y para la clase 1 un f1-score de 0.63. A pesar de tener un buen recall para la clase 0 (86%), el recall para la clase 1 fue solo del 55%, similar a la regresión logística, lo que indica que también tiene dificultades en detectar correctamente los casos de fallecimiento.

**3. Support Vector Machine (SVM)**: Con C=0.1 y kernel='linear' tuvo un accuracy de 74.67%. En términos de precisión, logró un 83% para la clase 1 y 72% para la clase 0. Sin embargo, el recall para la clase 1 fue bajo (48%), lo que significa que, aunque es bastante preciso cuando predice un fallecimiento, no es muy eficaz en detectar todos los casos positivos (fallecimientos). El f1-score para la clase 1 fue de 0.61, lo que refleja un balance moderado entre precisión y recall.

**4. Gradient Boosting**: Con los parámetros óptimos (learning\_rate=0.1, max\_depth=3, y n\_estimators=100), logró un accuracy de 72%. Tuvo un f1-score de 0.78 para la clase 0 y 0.62 para la clase 1. La precisión para la clase 1 fue de 71%, con un recall de 55%, lo que indica un rendimiento comparable al Random Forest, aunque ligeramente inferior en términos generales.



**Comparación General:** El modelo de Regresión Logística tuvo el mejor desempeño general con un accuracy de 77.33%, destacando especialmente en su capacidad para identificar correctamente los casos de fallecimiento (clase 1) con una precisión de 85%.

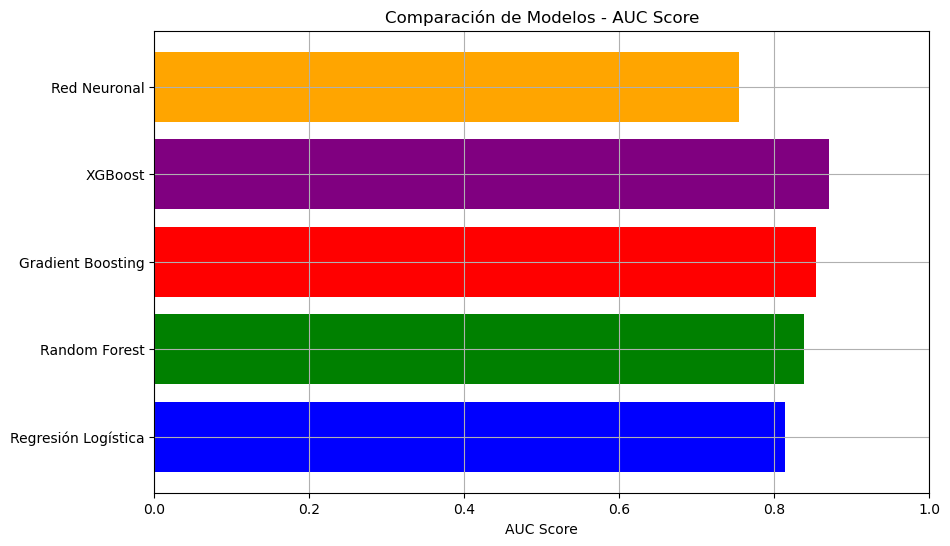
SVM tuvo un rendimiento ligeramente mejor que Random Forest y Gradient Boosting en términos de accuracy, con un 74.67%, pero su bajo recall en la clase 1 podría limitar su utilidad en un contexto donde es crítico identificar correctamente los casos positivos (fallecimientos). Random Forest y Gradient Boosting obtuvieron resultados similares, con accuracies alrededor del 73% y 72%, respectivamente. Estos modelos parecen ser más equilibrados, aunque no destacan en la detección de la clase minoritaria (fallecimientos).

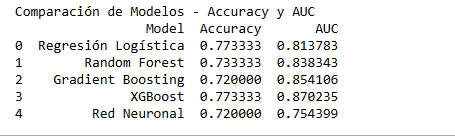
**Conclusión:** Dado que el objetivo es identificar correctamente los casos de fallecimiento, el modelo de Regresión Logística sería la mejor opción en este caso debido a su mayor precisión general y un mejor balance entre precisión y recall para la clase positiva. Sin embargo, se podría explorar un ajuste de hiperparámetros más fino en modelos como SVM o Gradient Boosting para mejorar la sensibilidad hacia la clase minoritaria.

**Segunda versión de modelos:**

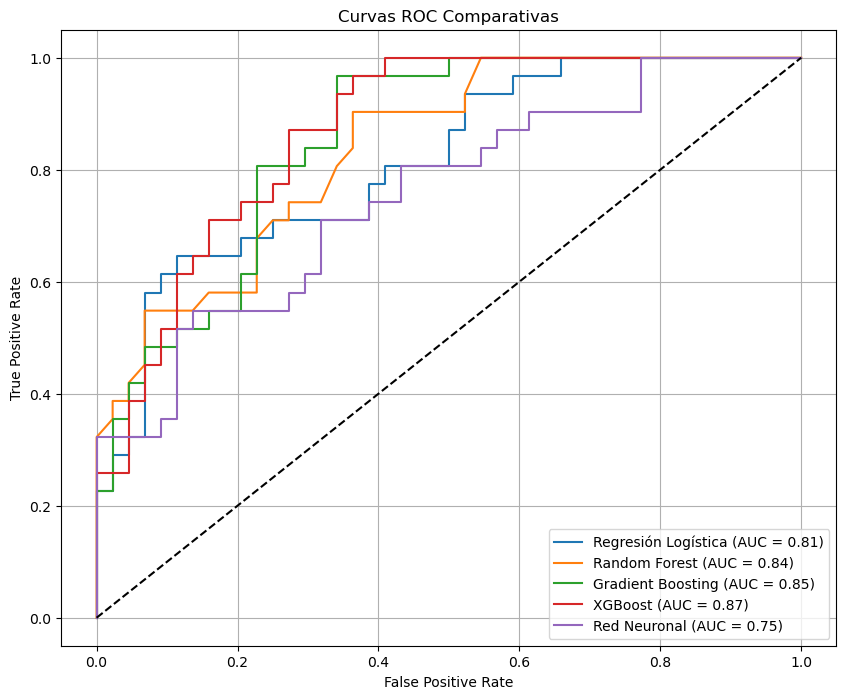
Se ajusta una red neuronal mejorada con 32 y 16 neuronas y se incluye XGBoostClassifier

Curvas ROC y AUC: El gráfico de barras compara el AUC de los cinco modelos:





Curvas ROC: Un gráfico adicional muestra las curvas ROC para cada modelo, permitiendo una evaluación visual de su rendimiento:



**Conclusiones:**

**XGBoost:** Es el modelo con mejor rendimiento global debido a su combinación de alta **accuracy** y el mayor AUC (0.8702), lo que lo convierte en la mejor opción para este problema. Este modelo es especialmente recomendable si tu objetivo es maximizar la capacidad de detección de eventos (clases) con alta sensibilidad.

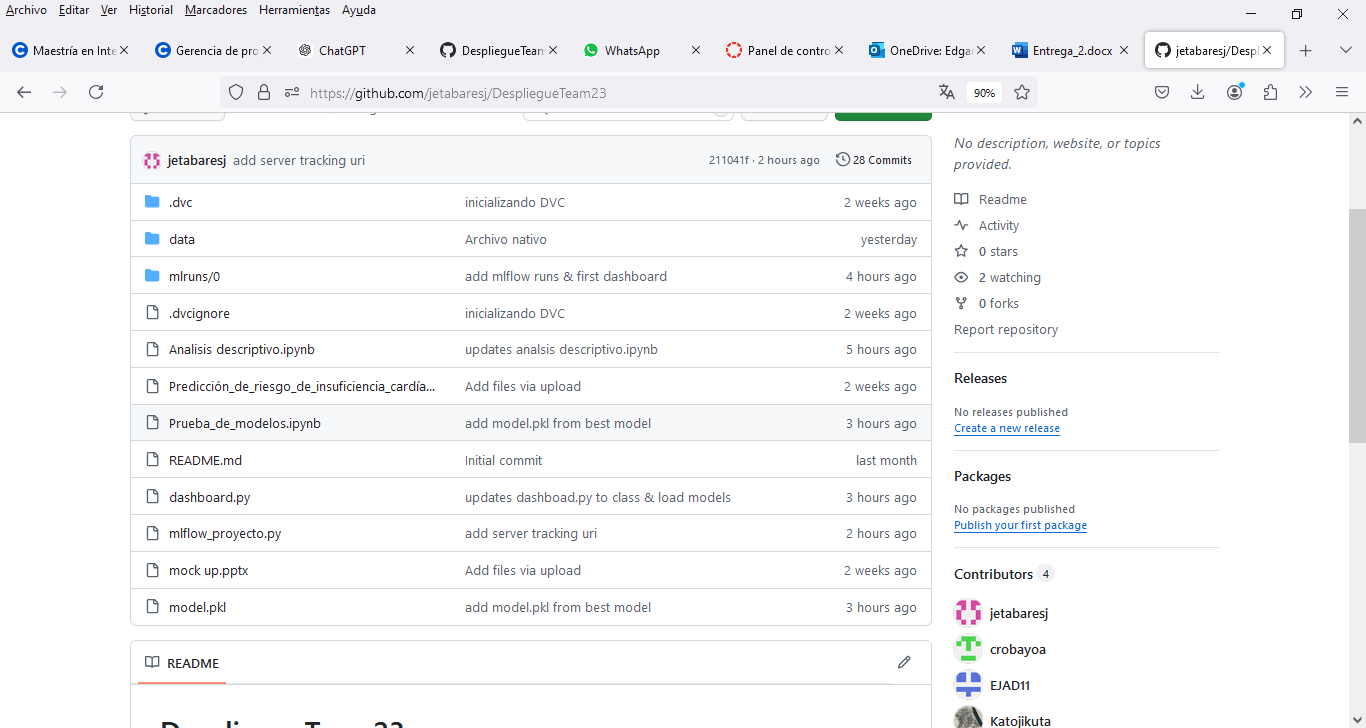
**Regresión Logística** también presenta un buen accuracy, igualando a **XGBoost**, pero con un AUC menor, lo que podría implicar un menor rendimiento en escenarios con un desbalance significativo de clases.

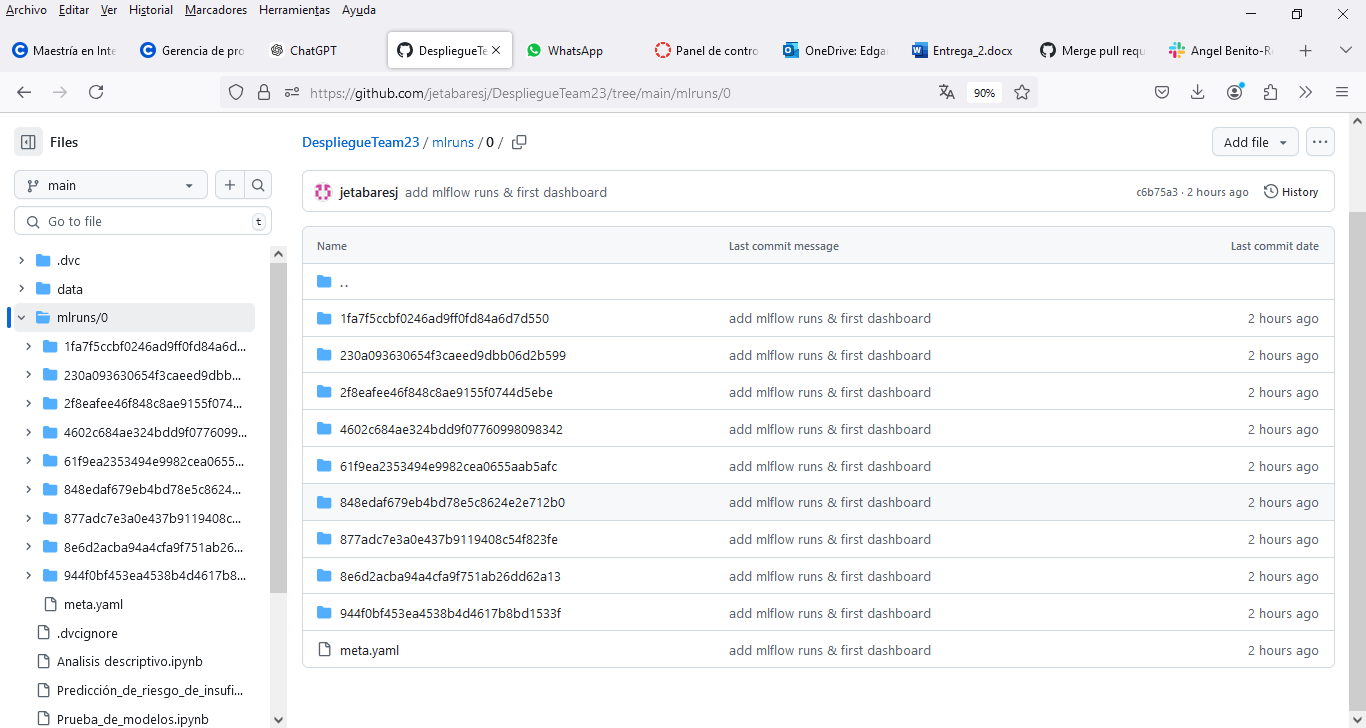
**Random Forest y Gradient Boosting** se desempeñan de forma intermedia. Aunque tienen AUC altos, su accuracy es menor que la de XGBoost y Regresión Logística, lo que sugiere que podrían ser útiles en aplicaciones donde la capacidad de discriminación es más crítica que la precisión general.

**Red Neuronal** mostró el rendimiento más bajo, indicando que este modelo podría requerir ajustes adicionales, como una arquitectura más compleja o más datos de entrenamiento para mejorar su capacidad predictiva.

**Repositorios con todo el código:** <https://github.com/jetabaresj/DespliegueTeam23>

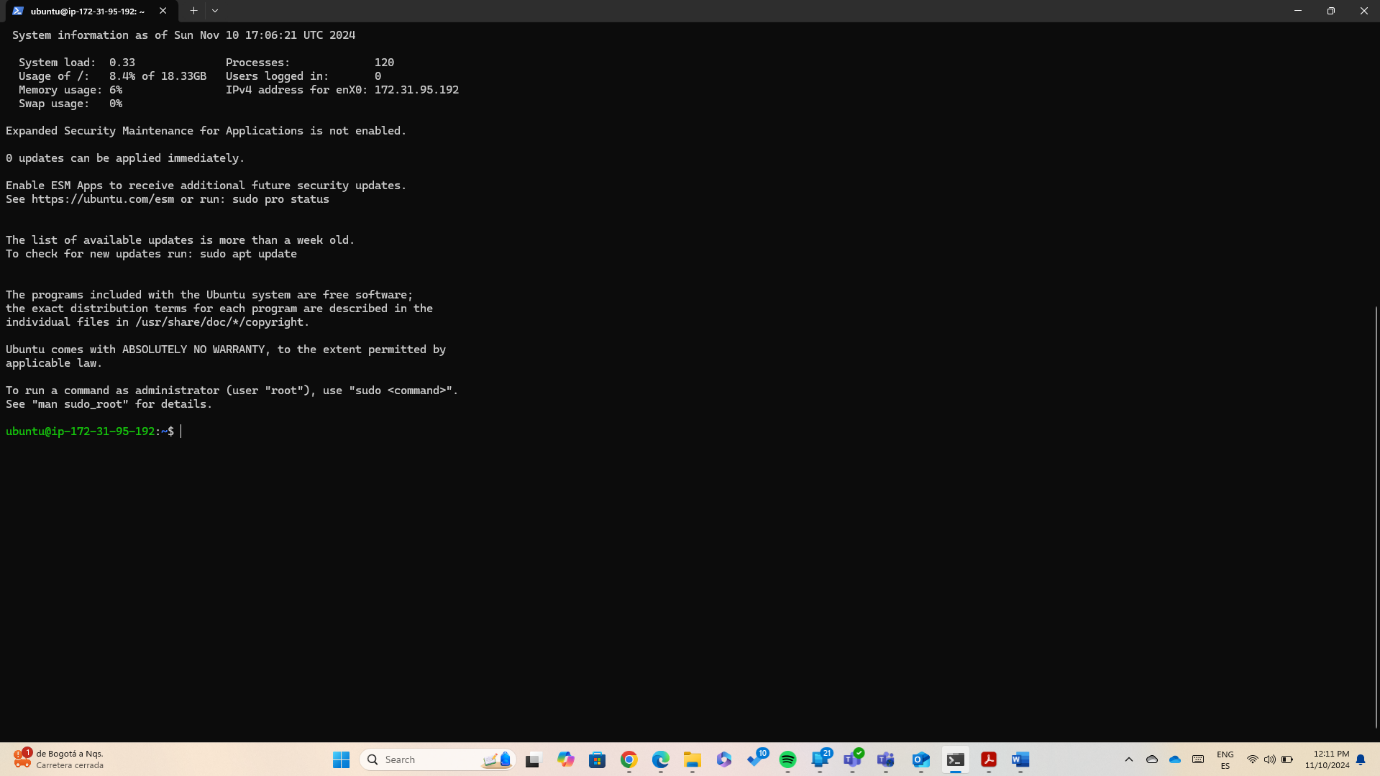
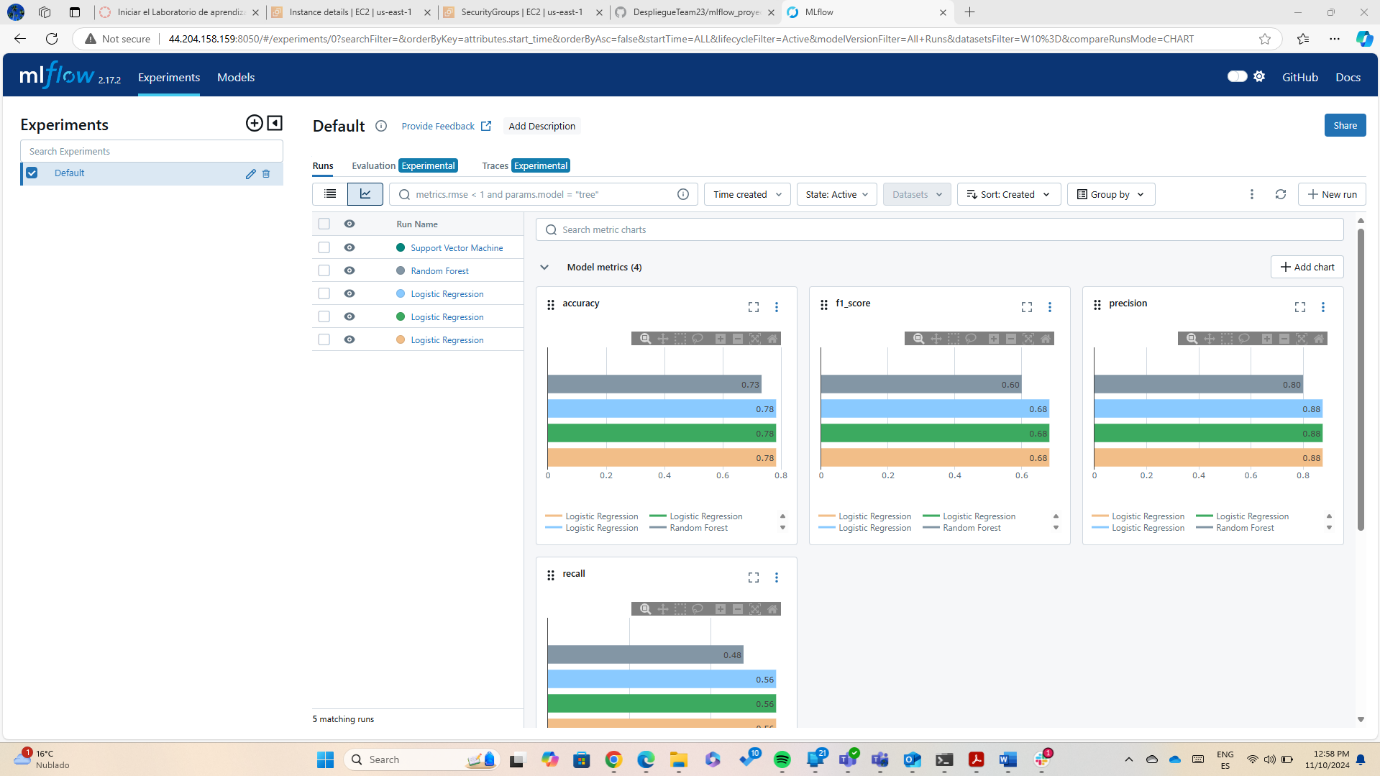
En la carpeta principal se subieron los archivos mlflow\_proyecto.py y dashboard.py los cuales fueron desplegados en la máquina EC2.

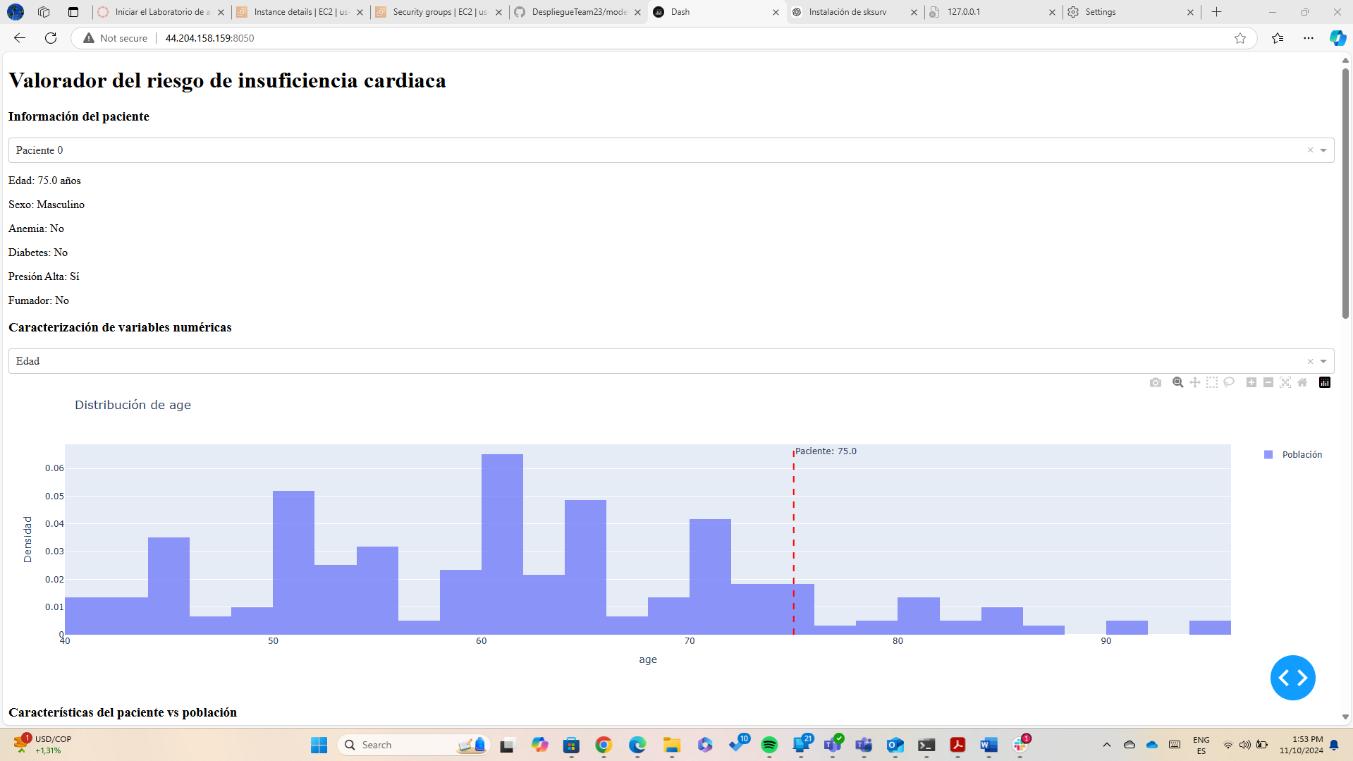


En la carpeta **mlruns** quedan los soportes del despliegue de los modelos y el tablero: 

**Conjuntos de datos a emplear:** <https://archive.ics.uci.edu/dataset/519/heart+failure+clinical+records>

**Despliegue mlflow**

**Conexión máquina virtual****Ejecución modelos****Dashboard**



**Fuente modelos desarrollados:**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

import mlflow

import mlflow.sklearn

mlflow.set\_tracking\_uri("http://localhost:5000")

url = "<https://raw.githubusercontent.com/jetabaresj/DespliegueTeam23/refs/heads/main/data/heart_failure_clinical_records_dataset.csv>"

data = pd.read\_csv(url)

X = data.drop('DEATH\_EVENT', axis=1)

y = data['DEATH\_EVENT']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Configurar modelos a probar y sus hiperparametros

models\_with\_params = {

"Logistic Regression": (LogisticRegression(max\_iter=1000), {

'C': [0.1, 1.0,],

'solver': ['liblinear',]

}),

"Random Forest": (RandomForestClassifier(), {

'n\_estimators': [50, 100,],

'max\_depth': [None, 10, ]

}),

"Support Vector Machine": (SVC(), {

'C': [0.1, 1.0, 10.0],

'kernel': ['linear', ]

}),

"Gradient Boosting": (GradientBoostingClassifier(), {

'n\_estimators': [50, 100,],

'learning\_rate': [0.01, 0.1,],

'max\_depth': [3, 5, 7]

})

}

# Definir función para entrenar y registrar modelos con MLflow

def train\_and\_log\_model(model\_name, model, params, X\_train, X\_test, y\_train, y\_test):

with mlflow.start\_run(run\_name=model\_name):

# Busqueda de hiperparámetros

grid\_search = GridSearchCV(estimator=model, param\_grid=params, scoring='f1', cv=5, n\_jobs=-1)

grid\_search.fit(X\_train, y\_train)

# Mejor modelo

best\_model = grid\_search.best\_estimator\_

# Hacer predicciones

y\_pred = best\_model.predict(X\_test)

# Calcular métricas

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred)

# Loggear hiperparámetros y métricas

mlflow.log\_param("model\_name", model\_name)

mlflow.log\_params(grid\_search.best\_params\_)

mlflow.log\_metric("accuracy", accuracy)

mlflow.log\_metric("precision", precision)

mlflow.log\_metric("recall", recall)

mlflow.log\_metric("f1\_score", f1)

# Registrar el modelo en MLflow

mlflow.sklearn.log\_model(best\_model, model\_name)

# Iterar sobre cada modelo, entrenarlo y registrarlo con MLflow

for model\_name, (model, params) in models\_with\_params.items():

print(model\_name, (params))

train\_and\_log\_model(model\_name, model, params, X\_train, X\_test, y\_train, y\_test)

# Cerrar el seguimiento de experimentos

mlflow.end\_run()

**Fuente Tablero desarrollado:**

from dash import Dash, html, dcc, callback, Output, Input, State

import plotly.express as px

import plotly.graph\_objects as go

import pandas as pd

import numpy as np

from pathlib import Path

import pickle

import logging

from sksurv.linear\_model import CoxPHSurvivalAnalysis

from sksurv.preprocessing import OneHotEncoder

from sksurv.metrics import concordance\_index\_censored

# Configurar logging

logging.basicConfig(level=logging.INFO)

logger = logging.getLogger(\_\_name\_\_)

class HeartFailurePredictor:

def \_\_init\_\_(self, data\_path: str, model\_path: str):

"""Inicializar el predictor con las rutas de datos y modelo."""

self.features = ['age', 'anaemia', 'creatinine\_phosphokinase', 'diabetes',

'ejection\_fraction', 'high\_blood\_pressure', 'platelets',

'serum\_creatinine', 'serum\_sodium', 'sex', 'smoking', 'time']

name=f'Paciente {patient\_id}',

line=dict(color='red', width=2)

))

fig.add\_trace(go.Scatter(

x=times,

y=base\_survival,

mode='lines',

name='Población base',

line=dict(color='blue', width=2, dash='dash')

))

fig.update\_layout(

title='Curva de Supervivencia Estimada',

xaxis\_title='Tiempo (días)',

yaxis\_title='Probabilidad de supervivencia',

yaxis\_range=[0, 1],

showlegend=True,

legend=dict(

yanchor="top",

y=0.99,

xanchor="left",

x=0.01

)

)

return fig

except Exception as e:

logger.error(f"Error en update\_survival\_curve: {str(e)}")

return go.Figure()

def run\_server(self, debug: bool = False):

"""Iniciar el servidor de la aplicación."""

try:

self.app.run(debug=debug)

except Exception as e:

logger.error(f"Error al iniciar el servidor: {str(e)}")

raise

if \_\_name\_\_ == '\_\_main\_\_':

try:

predictor = HeartFailurePredictor(

data\_path='data/heart\_failure\_clinical\_records\_dataset.csv',

model\_path='model.pkl'

)

dashboard = DashboardUI(predictor)

dashboard.run\_server(debug=True)

**Reporte de trabajo en equipo**

En la siguiente tabla se muestra el resumen de las actividades realizadas por cada integrante del equipo en pro de realizar la entrega correspondiente:

|  |  |
| --- | --- |
| Nombre | Actividades |
| Jhon Esteban Tabares Jiménez | -Generación versión final modelos -Generación versión final tablero |
| Camilo Robayo Abello | -Despliegue modelos  -Despliegue tablero  -Video presentación |
| Angel Benito Rebollo | -Ajustes presentación  -Ajustes tablero  -Video presentación |
| Javier Abril | -Estructura presentación -Video presentación  -Informe final y carga en Coursera |

Se adicionan también las contribuciones realizadas al repositorio por cada uno de los integrantes del grupo.

