Facultad de Ingeniería en Electricidad y Computación

Escuela Superior Politécnica del Litoral

Carrera:

Ingeniería en Computación

Primer avance de proyecto de segundo parcial

Materia:

Estructuras de Datos

Paralelo 3 - Grupo 9

Profesor:

MSc. Realpe Robalino Miguel Andres

Integrantes:

Lavayen Santana Stefany

Mateo Espinoza Willy

Montalvo Velasquez Rafael

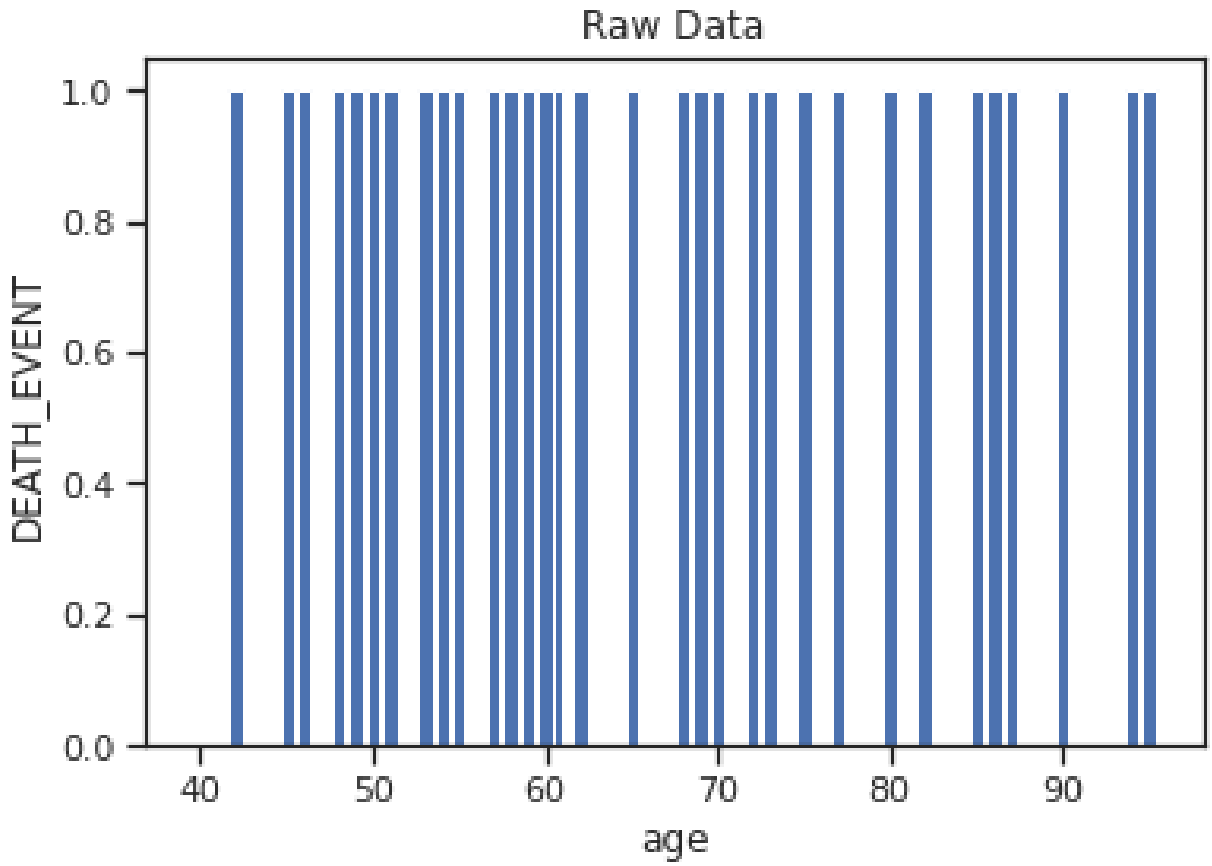
Contenido

Valores de umbral para los atributos no booleanos.

Para hallar un posible umbral heurístico para convertir los valores de los atributos no booleanos se utilizó una comparativa de diagrama de barras, con la finalidad de encontrar sectores en los cuales la densidad de las barras es mayor que significan los casos donde el paciente ha fallecido en el periodo de seguimiento, es decir cuando el atributo DEATH_EVENT es 1, con respecto a los atributos analizados. En el dataset de pacientes analizado se manejan 6 atributos que poseen valores diferentes a los booleanos, a continuación, se detalla el proceso para determinar un posible valor de umbral:

Atributo age.

```
plt.bar(dfPacientes["age"], dfPacientes["DEATH_EVENT"])
plt.xlabel('age')
plt.ylabel('DEATH_EVENT')
plt.title("Raw Data")
plt.show()
```



Conclusión.

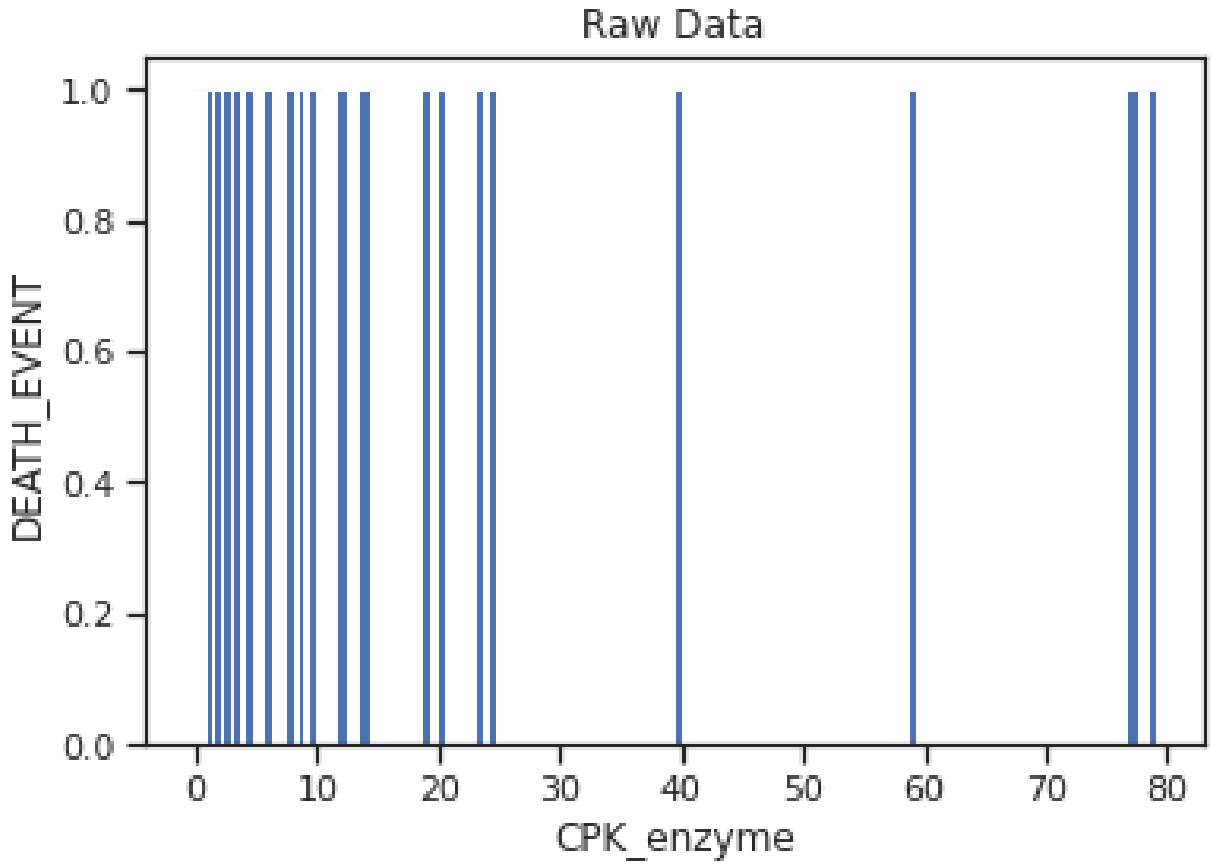
Posible valor intermedio: **62.**

Valores verdaderos: **<62.**

Valores falsos: **>62.**

Atributo CPK_enzyme.

```
plt.bar(dfPacientes["CPK_enzyme"]/100, dfPacientes["DEATH_EVENT"])
plt.xlabel('CPK_enzyme')
plt.ylabel('DEATH_EVENT')
plt.title("Raw Data")
plt.show()
```



Conclusión.

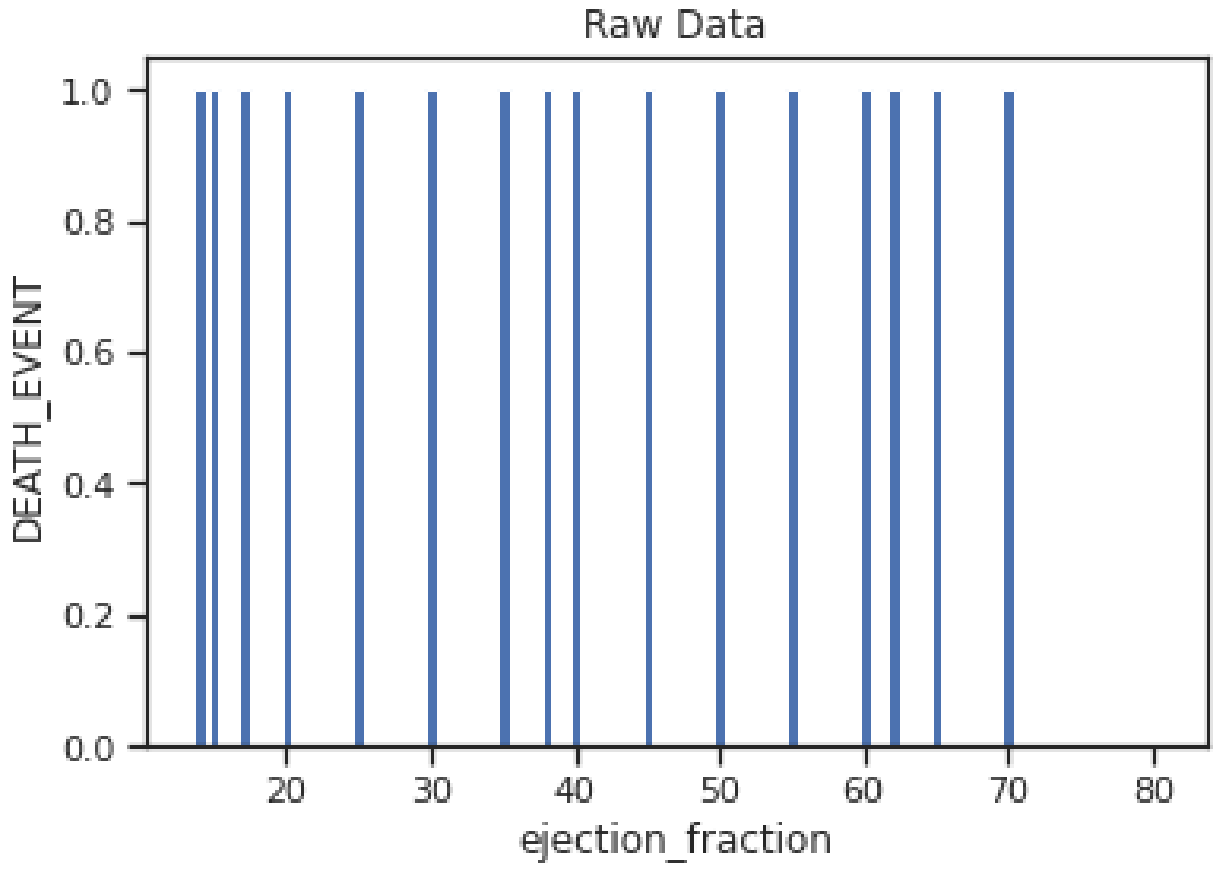
Posible valor intermedio: **25*100 =2500.**

Valores verdaderos: **<2500.**

Valores falsos: **>2500.**

Atributo ejection_fraction.

```
plt.bar(dfPacientes["ejection_fraction"], dfPacientes["DEATH_EVENT"])
plt.xlabel('ejection_fraction')
plt.ylabel('DEATH_EVENT')
plt.title("Raw Data")
plt.show()
```



Conclusión.

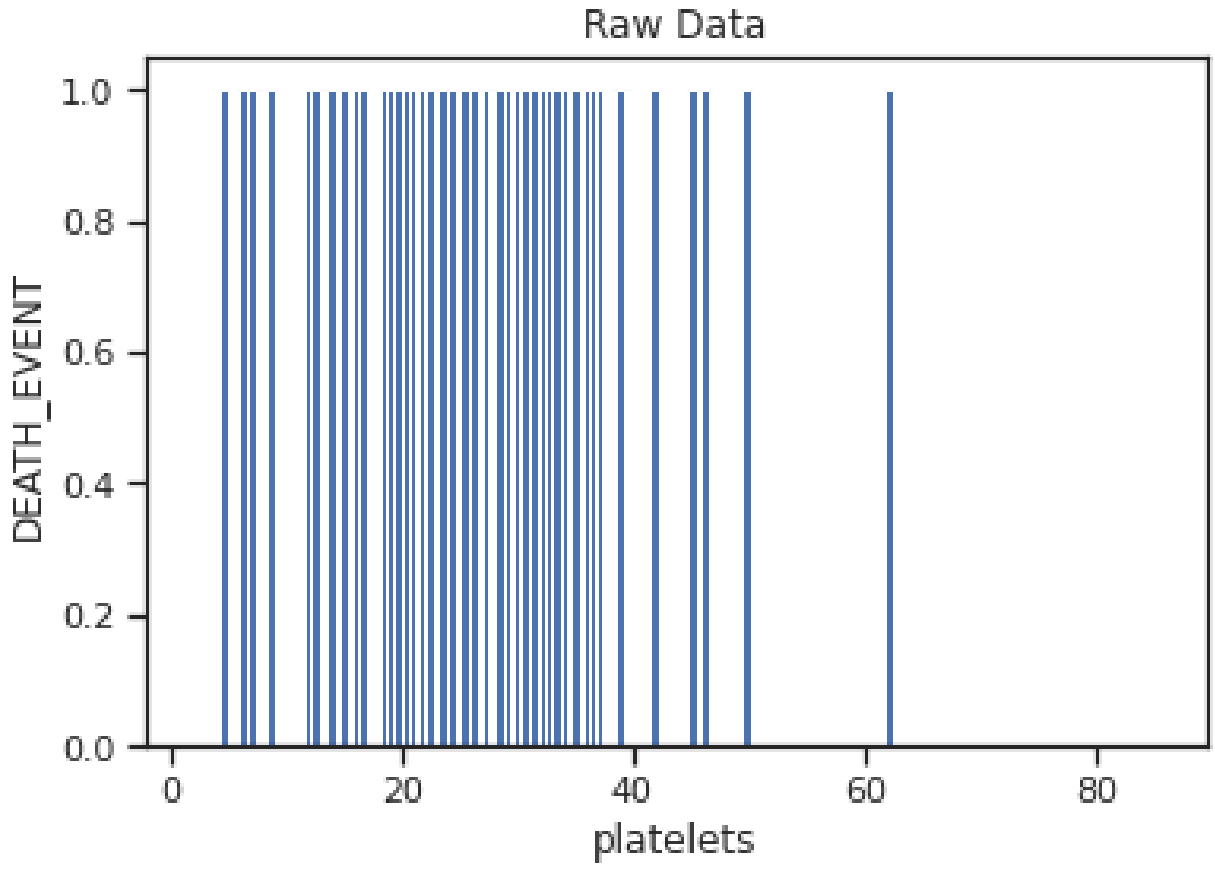
Posible valor intermedio: **43.**

Valores verdaderos: **<43.**

Valores falsos: **>43.**

Atributo platelets.

```
plt.bar(dfPacientes["platelets"]/10000, dfPacientes["DEATH_EVENT"])
plt.xlabel('platelets')
plt.ylabel('DEATH_EVENT')
plt.title("Raw Data")
plt.show()
```



Conclusión.

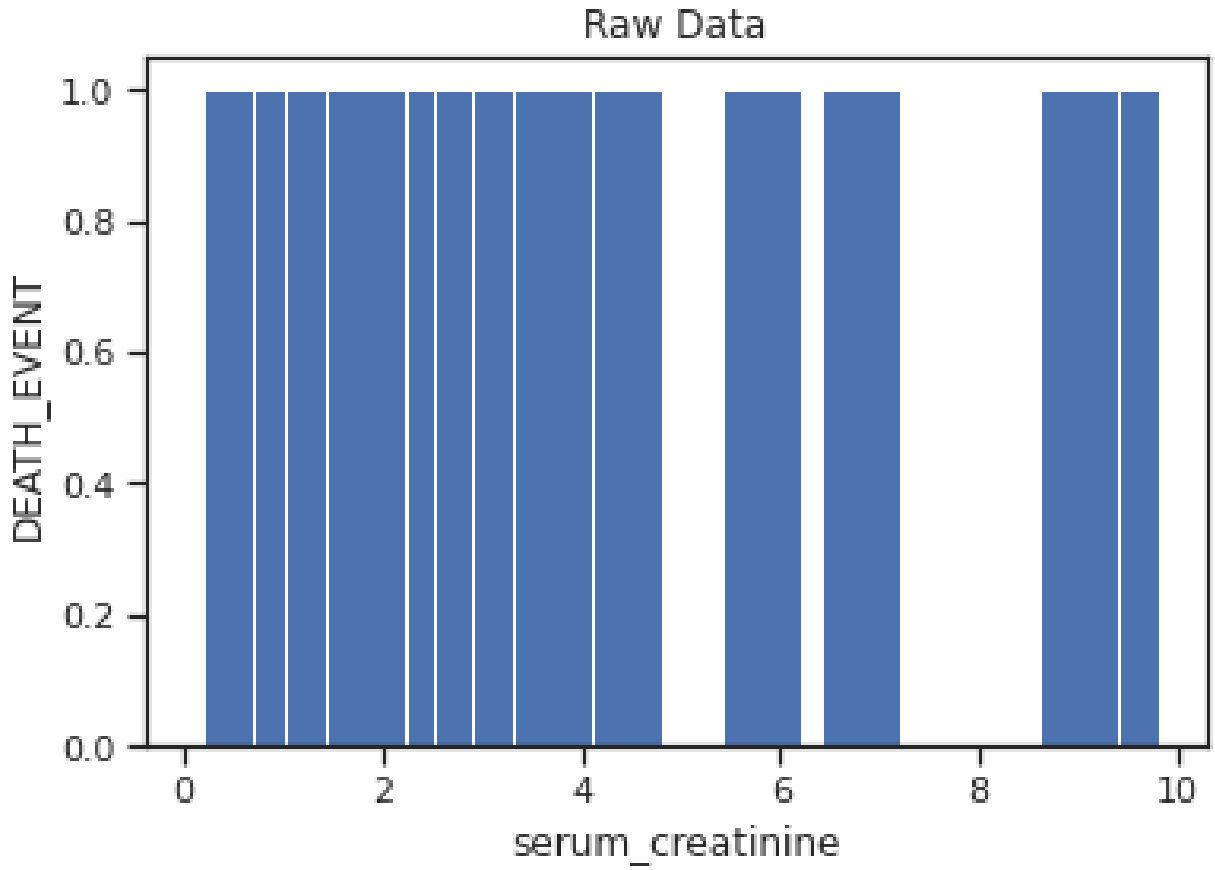
Posible valor intermedio: **50*1000= 50000.**

Valores verdaderos: **<50000.**

Valores falsos: **>50000.**

Atributo serum_creatinine.

```
plt.bar(dfPacientes["serum_creatinine"], dfPacientes["DEATH_EVENT"])
plt.xlabel('serum_creatinine')
plt.ylabel('DEATH_EVENT')
plt.title("Raw Data")
plt.show()
```



Conclusión.

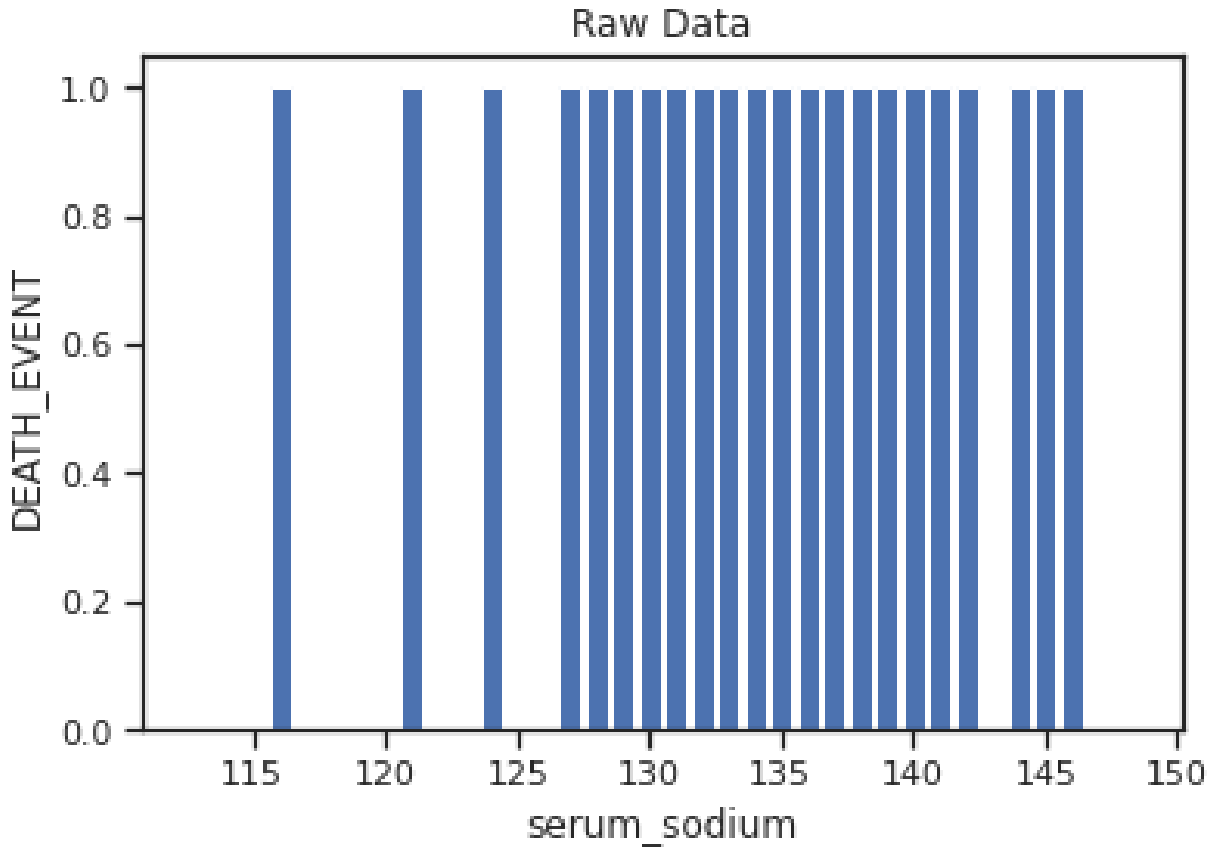
Posible valor intermedio: **5.**

Valores verdaderos: **<5.**

Valores falsos: **>5.**

Atributo serum_sodium.

```
plt.bar(dfPacientes["serum_sodium"], dfPacientes["DEATH_EVENT"])
plt.xlabel('serum_sodium')
plt.ylabel('DEATH_EVENT')
plt.title("Raw Data")
plt.show()
```



Conclusión.

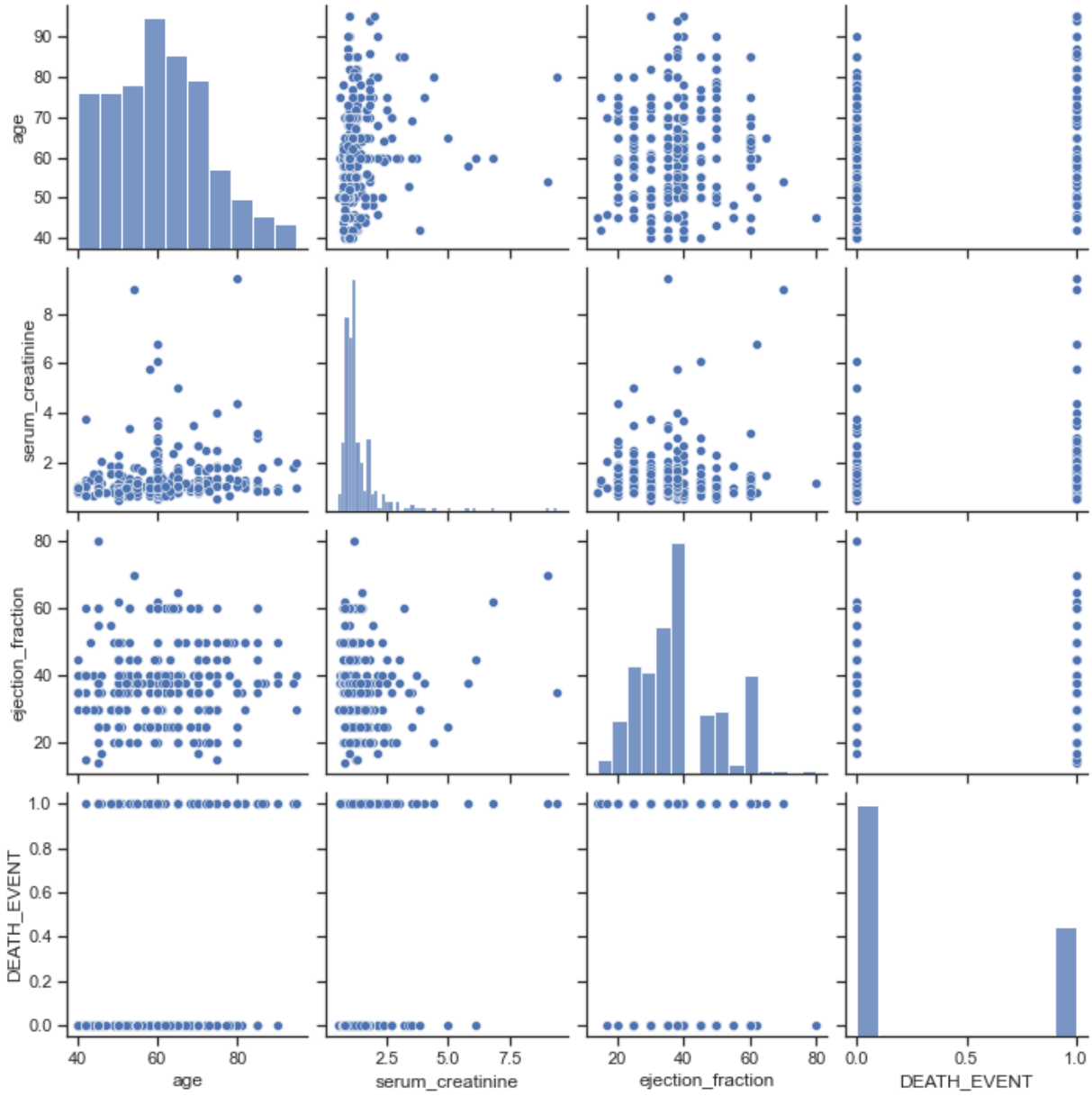
Posible valor intermedio: **127.**

Valores verdaderos: **>127.**

Valores falsos: **<127.**

Atributos con mayor relevancia para crear un árbol de decisión.

Fue necesario utilizar una gráfica de correlación de Pearson, la cual nos permite determinar que grado de relación poseen dos variables (ya sea directa o inversa), con el objetivo de lograr establecer cuáles son los atributos del dataset de mayor importancia.



Dado a que DEATH_EVENT es la variable que buscamos predecir, entonces es necesario analizar su correlación con las demás variables, dónde resaltan las que poseen valores correlativos muy altos o demasiado negativos. Los gráficos de barras y de dispersión reafirman el análisis para concluir que los 3 atributos de mayor relevancia y que serán primordiales para crear el árbol de decisión vienen dados en el siguiente orden:

1. serum_creatinine.
2. age.
3. ejection_fraction.

Representación del Dataset para un programa en Java.

Para incorporar el dataset a un software programado en lenguaje Java, se propone utilizar las siguientes estructuras que permitirán la óptima implementación, manipulación y almacenamiento de la información del dataset:

Matriz.

Es una excelente manera de representar para un dataset, pues ofrece una complejidad de tiempo mínima (O (1)) para acceder a sus elementos, por lo que el recorrerlo tiene el mismo tiempo computacional que el tamaño del array, considerando que es uno bastante bueno para manejar información muy extensa. Los atributos y la información relacionada a ellos se accederán mediante índices previamente asignados.

ArrayList.

Se plantea utilizar esta estructura para almacenar la información de cada una de las columnas en listas diferentes, pero conociendo que todos los elementos de una posición X pertenecen a la misma fila de información. Esto permitirá que el agregar los datos de manera secuencial al final uno del otro, lo que tendría una complejidad de tiempo mínima O (1), siendo útil para crearla. Asimismo, esta estructura maneja la misma complejidad de tiempo en caso de tener que realizar un get, por lo que podría ser una excelente forma de representar los datos.

Map.

Al utilizar esta estructura, se puede almacenar la información de las filas en los pares claves/valor a manera de FilaIndex/FilaData, lo que permitirá que la información pueda ser obtenida de manera rápida por medio de un forEach, teniendo una complejidad de tiempo equivalente a la cantidad de filas del archivo, solo en caso de recorrerlo y extraer su información.

Estructura escogida.

Se seleccionó la estructura *matriz o arreglo* debido a que contamos con un número de registros de datos fijos y presenta tiempos computaciones para acceso y almacenamiento de información inmediata representada en la notación de la gran O: O (1). Para recorrer los datos necesitaremos un tiempo O(N), sin embargo, el anterior análisis compensa este tiempo.