### **INTRODUCCIÓN**

Desde el inicio del proyecto, adoptamos un enfoque de trabajo en equipo que hemos

mantenido a lo largo de todo el curso. Nuestra dinámica consistió en compartir la pantalla y

tomar decisiones de manera conjunta, basándonos en el debate y el consenso de todos los

integrantes. Esta metodología nos ha permitido alcanzar los mejores resultados y fortalecer

nuestros conocimientos al complementar las habilidades de cada participante.

Es importante resaltar que, debido a nuestra forma de trabajo colaborativo, sin asignar

tareas específicas a cada integrante, no podemos identificar decisiones individuales de cada

miembro del equipo una vez finalizado el proyecto. Desde el inicio, cada integrante aportó

de manera igualitaria y trabajamos de forma conjunta en todas las actividades y decisiones.

Además es importante destacar que todas las decisiones tomadas para llevar a cabo este

proyecto fueron explicadas y justificadas a lo largo del notebook.

En resumen, elegimos una aproximación colaborativa en la que todos los miembros del

equipo participaron conjuntamente y se tomaron decisiones de manera colectiva en todo

momento. Esta forma de trabajo en equipo ha sido fundamental para el éxito del proyecto y

demuestra nuestra capacidad de trabajar en equipo de manera efectiva.

Somos conscientes de que uno de los requisitos es presentar un seguimiento detallado de

las contribuciones individuales de cada miembro del equipo. Sin embargo, dada la

metodología que hemos explicado previamente, hemos decidido otorgar prioridad al éxito

del proyecto en lugar de realizar una anotación continua de cada idea o decisión individual.

Antes de comenzar describimos las variables del dataset:

"""

Nombres de las variables!

amount\_tsh = carga estática total (cantidad de agua disponible, para el punto de agua).

date\_recorded = fecha en la que se incluyó el registro en los datos.

funder = quién financió el pozo.

gps\_height = altitud del pozo.

installer = organización que lo instaló.

longitude = coordenada GPS.

latitude = coordenada GPS.

wpt\_name = nombre del punto de agua, si lo tiene.

num\_private =

basin = cuenca hidrográfica.

subvillage = localización geográfica.

region = localización geográfica.

region\_code = código localización geográfica.

district\_code = código localización geográfica.

lga = ubicación geográfica.

ward = ubicación geográfica.

population = población alrededor del pozo.

public\_meeting = True/False si es punto de reunión.

recorded\_by = grupo que introdujo este registro en los datos.

scheme\_management = quién opera el punto de agua.

scheme\_name = quién opera el punto de agua.

permit = si el punto de agua está permitido.

construction\_year = año de construcción.

extraction\_type = el tipo de extracción que utiliza el punto de agua.

extraction\_type\_group = el tipo de extracción que utiliza el punto de agua.

extraction\_type\_class = el tipo de extracción que utiliza el punto de agua.

management = cómo se gestiona el pozo.

management\_group = cómo se gestiona el pozo.

payment = coste del agua.

payment\_type = coste del agua.

water\_quality = calidad del agua.

quality\_group = calidad del agua.

quantity = cantidad de agua que aporta el pozo.

quantity\_group = cantidad de agua que aporta el pozo.

source = la fuente del agua.

source\_type = la fuente del agua.

source\_class = la fuente del agua.

waterpoint\_type = el tipo de punto de agua.

waterpoint\_type\_group = el tipo de punto de agua.

"""

### **2.1. Ejercicio 1**

**Dado un fichero reto\_agua.csv con los datos, realizad los siguientes puntos:**

* **Cargad el csv**
* **Mostrad los primeros 5 datos**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.linear\_model import LogisticRegression

from sklearn.linear\_model import LinearRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import ExtraTreesClassifier

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

from sklearn import preprocessing

from sklearn.metrics import confusion\_matrix

import statsmodels.api as sm

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

from sklearn.model\_selection import GridSearchCV

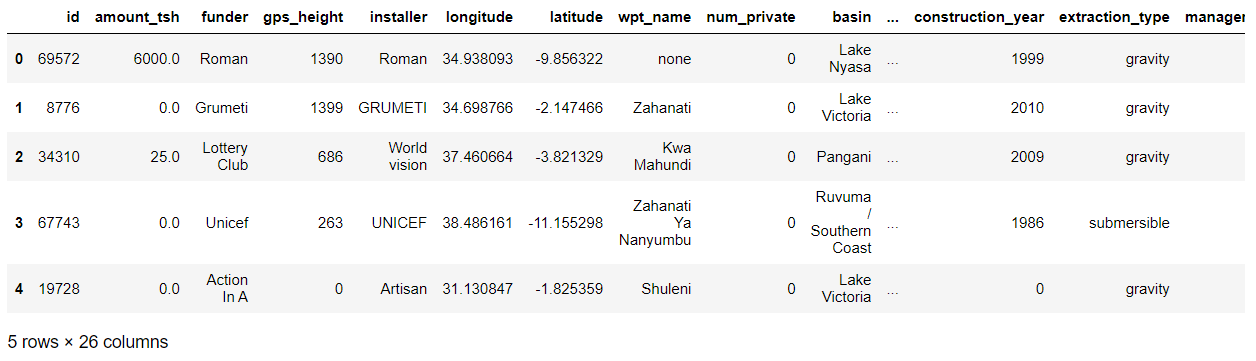
from sklearn.model\_selection import RandomizedSearchCV

file=r'reto\_agua.csv'

data=pd.read\_csv(file,sep=',')

df=data.copy()

df.head()



* **Realizad un análisis exploratorio de la estructura y los datos**
* **Extraed la información de la estructura del dataset para responder a las siguientes preguntas:**

Decidimos eliminar la variable "id" porque es un valor único para cada registro

df.drop(columns='id',inplace=True)

Decidimos unificar todos los valores object a minúsculas, antes de buscar duplicados y evitar categorías que pudiesen ser iguales escritas con mayúsculas o minúsculas, para ello tenemos que pasar los datos a str y antes decidimos tratar los valores nulos ya que luego no serían detectables.

def deteccion\_nulos(df):

null\_to\_drop=[]

for i in df.columns:

null=df[i].isnull().sum()

if null >0:

percentage\_of\_null=(null\*100)/len(df)

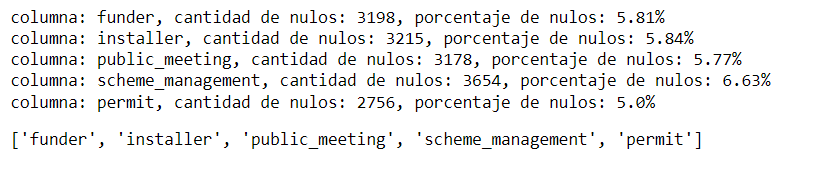
print(f'columna: {i}, cantidad de nulos: {null}, porcentaje de nulos: {round(percentage\_of\_null,2)}%')

null\_to\_drop.append(i)

return(null\_to\_drop)

nulos = deteccion\_nulos(df)

nulos



Como las variables donde hay nulos son categóricas, decidimos poner una "?" a los nulos para que tomen su propia categoría:

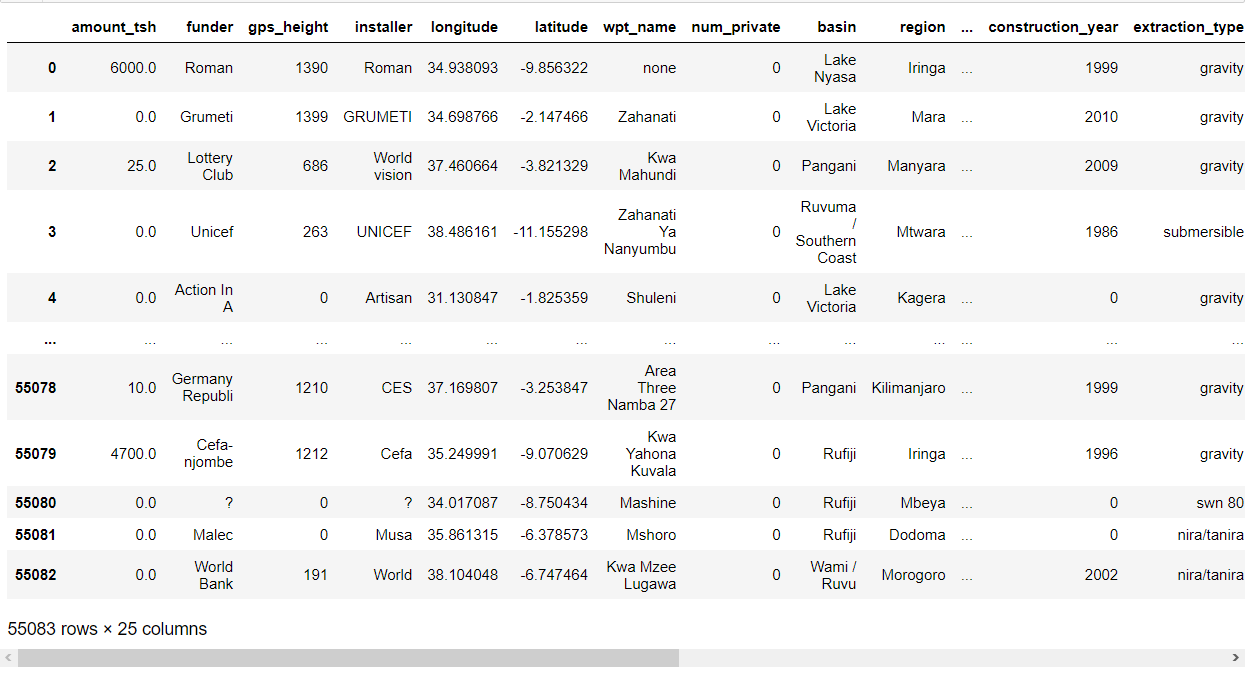
def cambio\_a\_interrogacion(df, nulos):

for i in nulos:

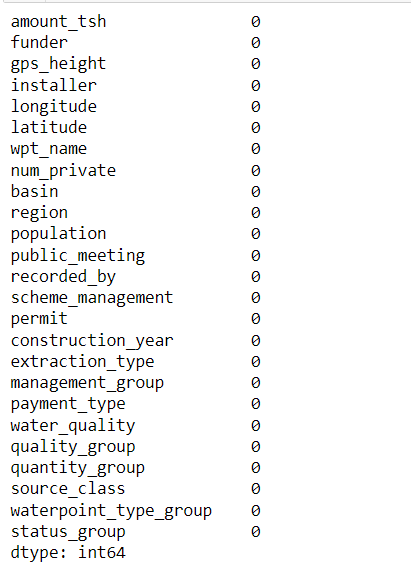
df[i] = df[i].fillna("?")

return df

cambio\_a\_interrogacion(df, nulos)



df.isnull().sum()



Transformación a minúculas

columnas\_a\_transformar = ["funder", "installer", "wpt\_name", "basin", "region", "public\_meeting", "recorded\_by",

"scheme\_management", "permit", "extraction\_type", "management\_group", "payment\_type", "water\_quality",

"quality\_group", "quantity\_group", "source\_class", "waterpoint\_type\_group", "status\_group"]

def minusculas(df, columnas):

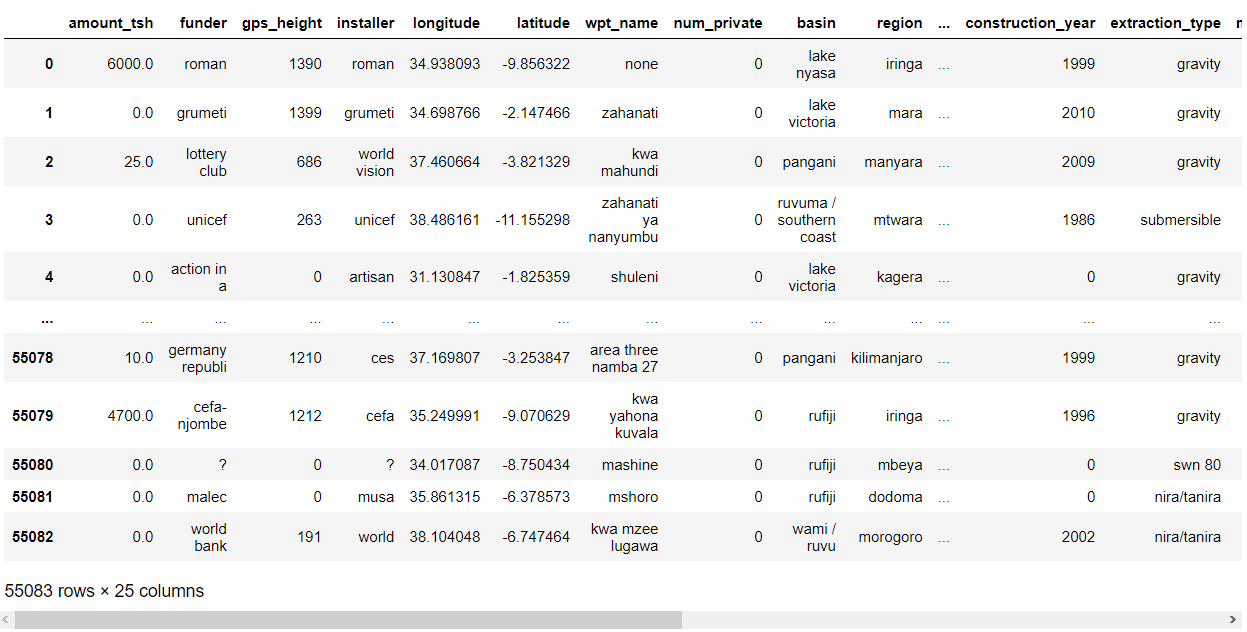
for i in columnas:

df[i] = df[i].astype(str).str.lower()

return df

df = minusculas(df, columnas\_a\_transformar)

df



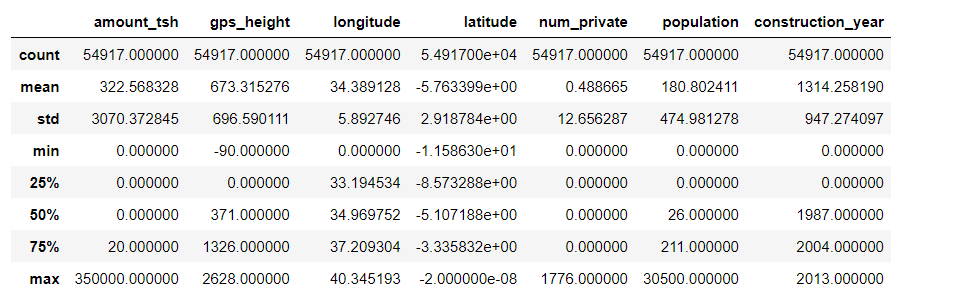
Tratamiento de duplicados

df.duplicated().sum()

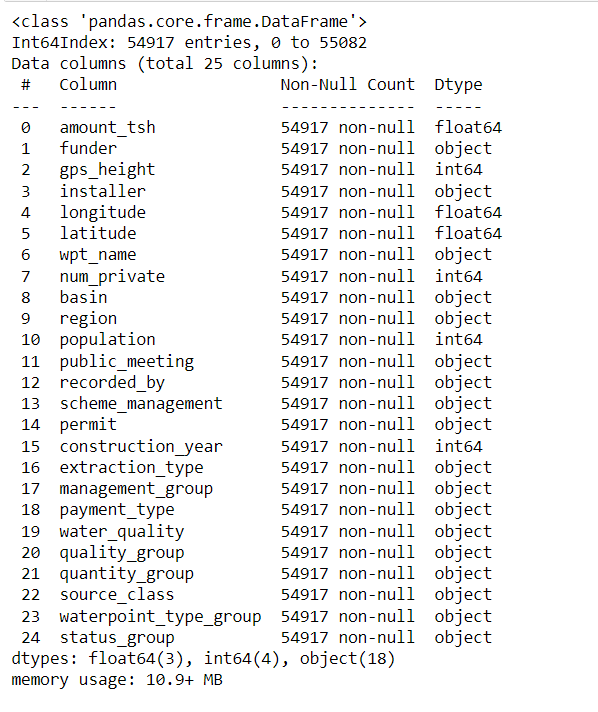
Entrega de resultado: 166

df=df.drop\_duplicates()

df.describe()

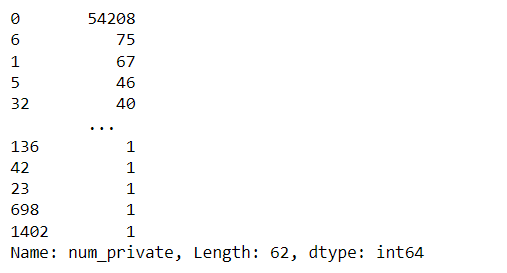


df.info()

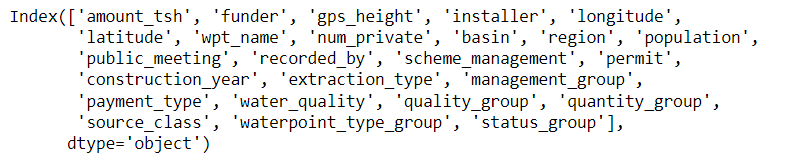


Decidimos observar algunas de las variables categóricas para intentar entenderlas mejor

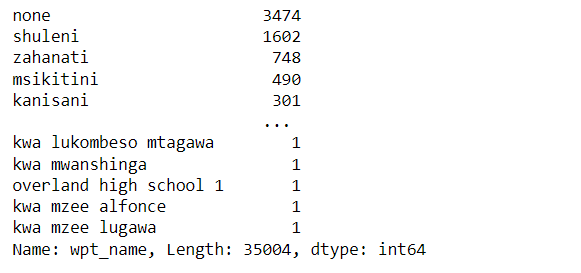
df.num\_private.value\_counts()



df.columns



df.wpt\_name.value\_counts()



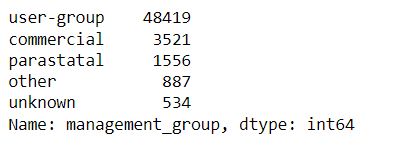
df.recorded\_by.value\_counts()



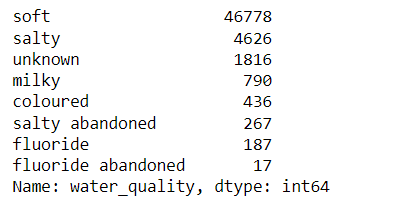
Decidimos eliminar la variable "recorded\_by" ya que tiene el mismo valor para todos los registros:

df.drop(columns='recorded\_by',inplace=True)

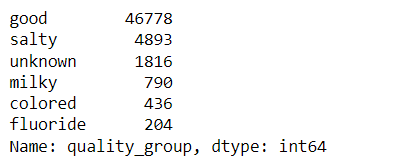
df.management\_group.value\_counts()



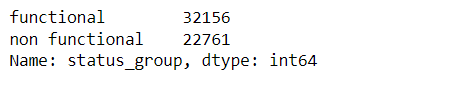
df.water\_quality.value\_counts()



df.quality\_group.value\_counts()



df.status\_group.value\_counts()



Encontramos el target como "status\_group"

**¿Veis alguna columna que no consideréis necesaria para el modelo?**

En nuestra opinion la columna ID no proporciona ninguna informacion relevante para la confeccion del modelo predictivo, ya que es simplemente un identificador y no afecta en nada al hecho de que la bomba funcione o no.

También decidimos eliminar la variable recorder\_by porque tiene el mismo valor en todos los registros

**¿Cuántos datos totales hay en dataset?**

Originalmente había 55083 registros en el Dataset. En el proceso exploratorio de datos identificamos 166 registros duplicados, los cuales decidimos eliminar, quedando de esta forma 54917 registros.

**¿Hay valores nulos? En ese caso, ¿qué columnas los tienen?**

Las columnas con elementos nulos son:

columna: funder, cantidad de nulos: 3198, porcentaje de nulos: 5.82%

columna: installer, cantidad de nulos: 3215, porcentaje de nulos: 5.85%

columna: public\_meeting, cantidad de nulos: 3134, porcentaje de nulos: 5.71%

columna: scheme\_management, cantidad de nulos: 3650, porcentaje de nulos: 6.65%

columna: permit, cantidad de nulos: 2756, porcentaje de nulos: 5.02%

Decidimos pasar los nulos a un símbolo "?" para después poder hacer un label encoder y que tengan su propia categoría

**¿Detectáis alguna columna que tenga datos anómalos? En ese caso, ¿cuáles?**

numeric\_columns = df.select\_dtypes(include=['float64', 'int64']).columns

fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(12, 6))

for i, column in enumerate(numeric\_columns):

row = i // 4

col = i % 4

*# Crea el boxplot en el subplot correspondiente*

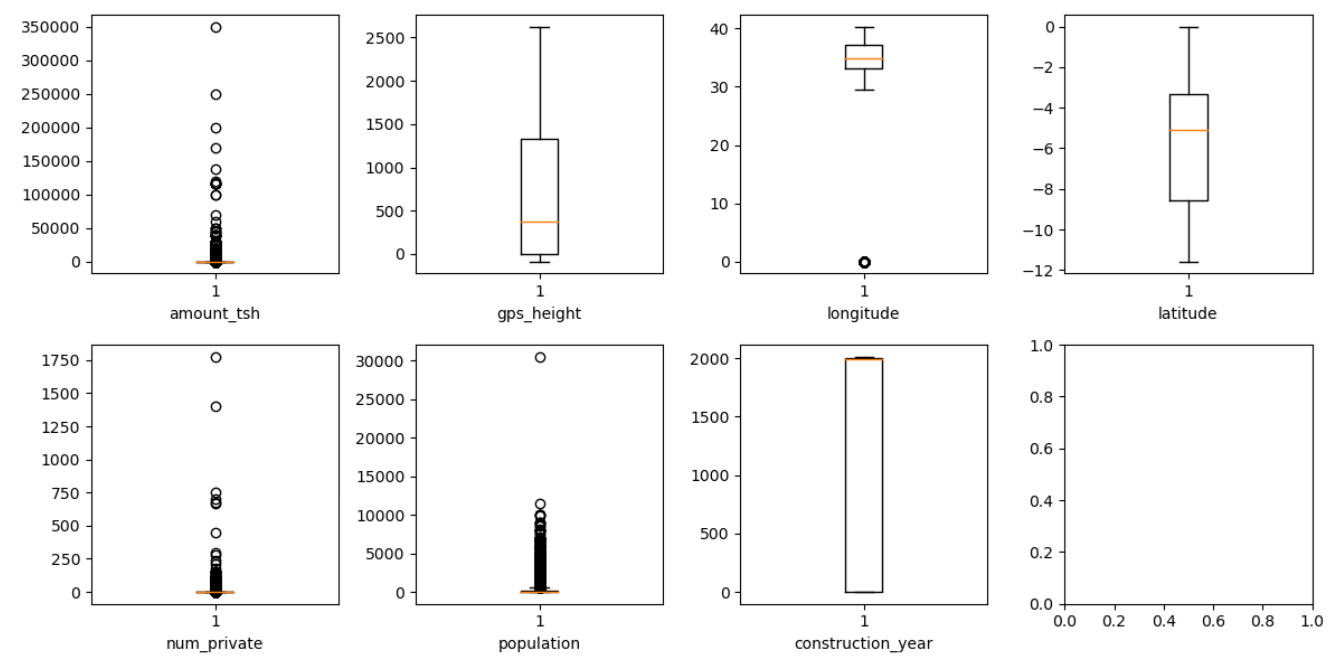
axes[row, col].boxplot(df[column], vert=True)

axes[row, col].set\_xlabel(column)

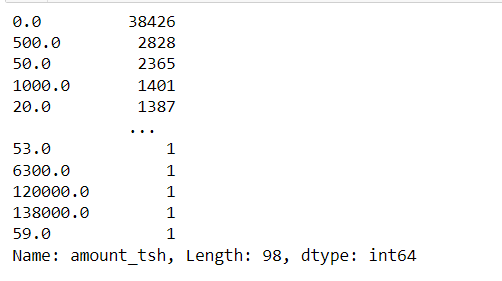
axes[row, col].set\_ylabel('')

plt.tight\_layout()

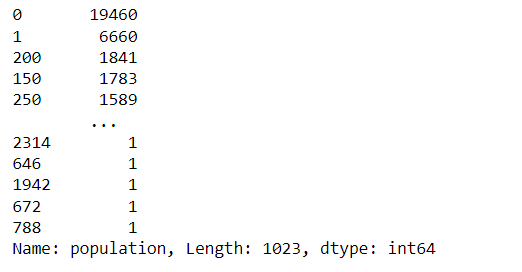
plt.show()



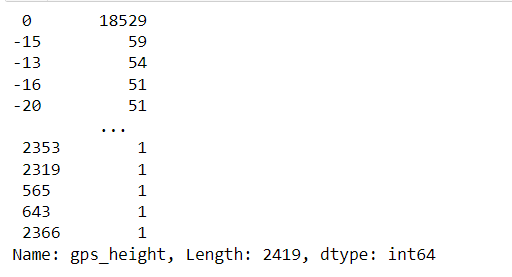
df.amount\_tsh.value\_counts()



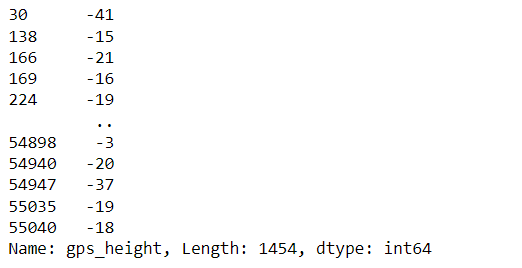
df.population.value\_counts()



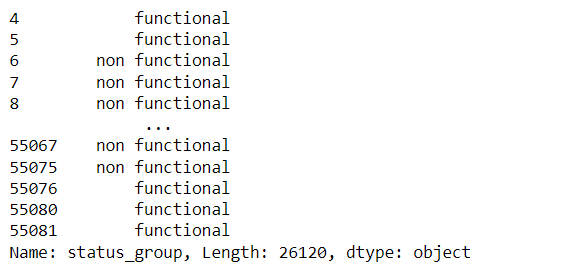
df.gps\_height.value\_counts()



df.loc[(df.gps\_height<0),'gps\_height']



df.loc[((df.population==0)|(df.population==1)),'status\_group']



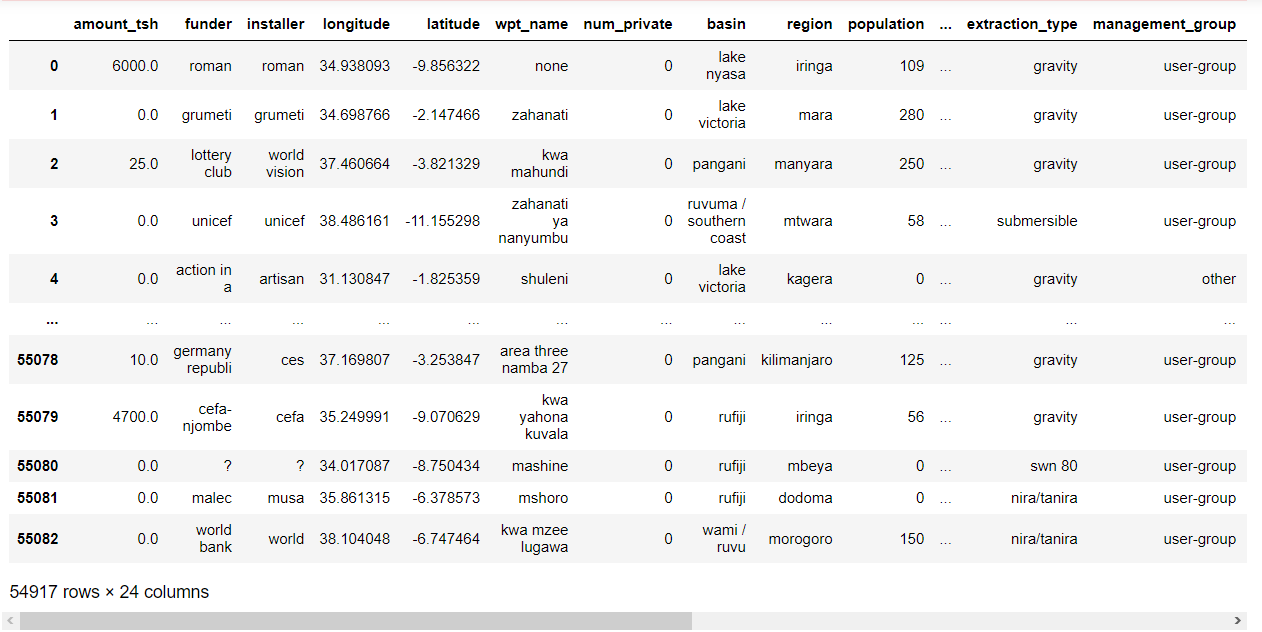
* amount\_tsh: Detectamos la presencia de muchos valores iguales a "0". De todas formas consideramos que este dato puede no representar una anomalía y ser un dato correcto ya que es la cantidad de agua que hay en el pozo y por diversas circunstancias este valor puede ser igual a "0".
* gps\_height: Nos llamó poderosamente la atención la presencia de valores negativos en esta variable, ya que representa los metros sobre el nivel del mar de cada pozo, es decir su altura. Imputamos los valores negativos a "0".

df["gps\_height\_imp"] = df["gps\_height"].apply(lambda x: x if x >= 0 else 0)

*#Eliminamos la columna original para dejar más limpio el df*

df = df.drop(columns="gps\_height")

df



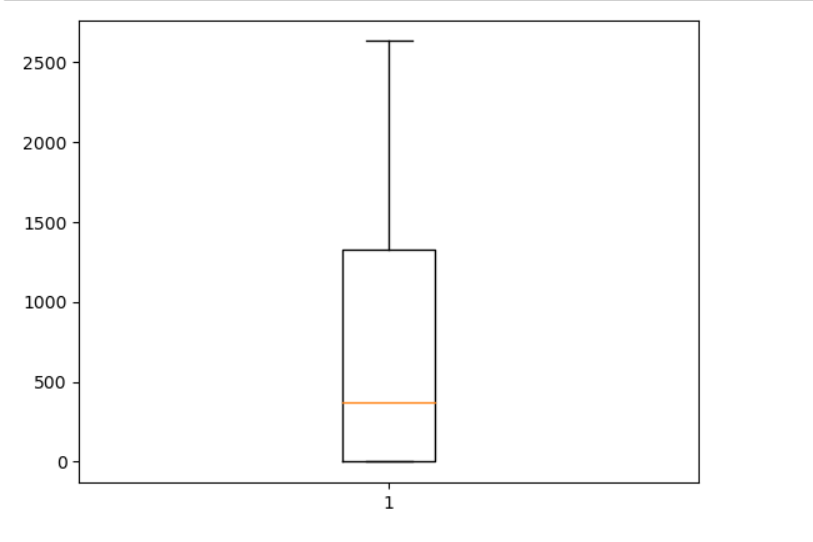
df.gps\_height\_imp.min()

Entrega el valor: 0

**Según pide el ejercicio 4, buscamos los outliers en "gps\_height"**

plt.boxplot(df['gps\_height\_imp'])

plt.show()



No detectamos ningún valor outlier probablemente porque previamente hemos corregido los valores que no tenían sentido (alturas de pozo negativas que estarían por debajo del nivel del mar)

* population: Detectamos muchos valores "0" que pueden ser reales porque correspondan con sitios que se han quedado desiertos, pero los valores "1" nos parecen muy extraños. Sacamos la mediana de la población excluyendo los valores "0" y "1", y se la imputamos a los valores "1"

valores = df.loc[(df.population>1),'population']

mediana\_population = valores.median()

df["population\_imp"] = df["population"].apply(lambda x: x if x != 1 else mediana\_population)

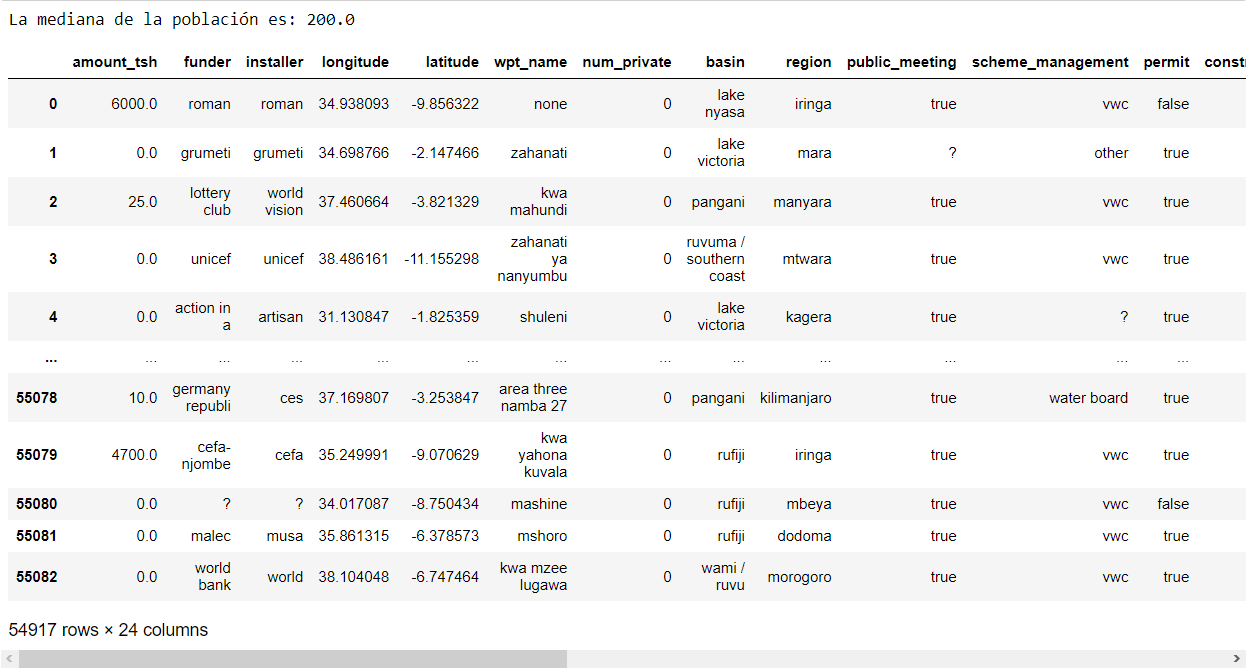
*#Eliminamos la columna original para dejar más limpio el df*

df = df.drop(columns="population")

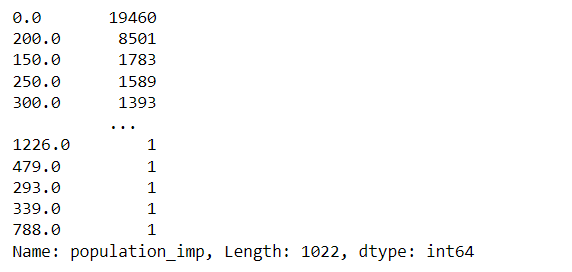
print("La mediana de la población es:", mediana\_population)

pd.set\_option('display.max\_columns', None)

df



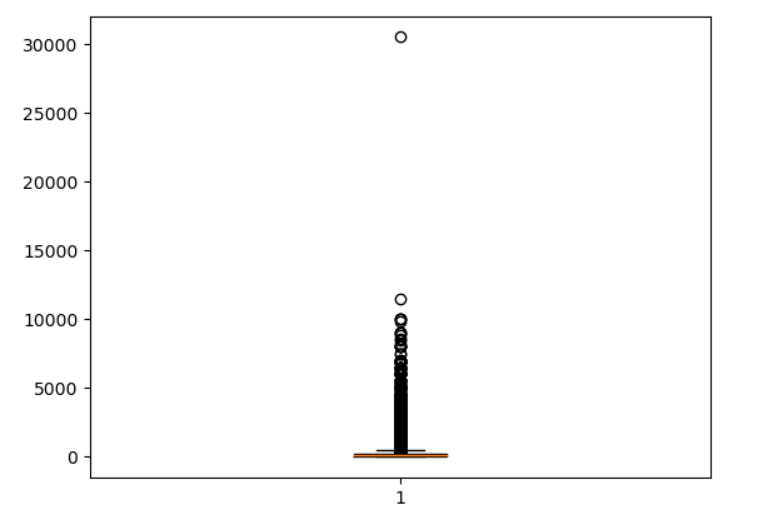
df["population\_imp"].value\_counts()



**Según pide el ejercicio 4, buscamos los outliers en "population"**

plt.boxplot(df['population\_imp'])

plt.show()



*# Cálculo de los percentiles*

q1 = np.percentile(df["population\_imp"], 25)

q3 = np.percentile(df["population\_imp"], 75)

per95 = df['population\_imp'].quantile(0.95)

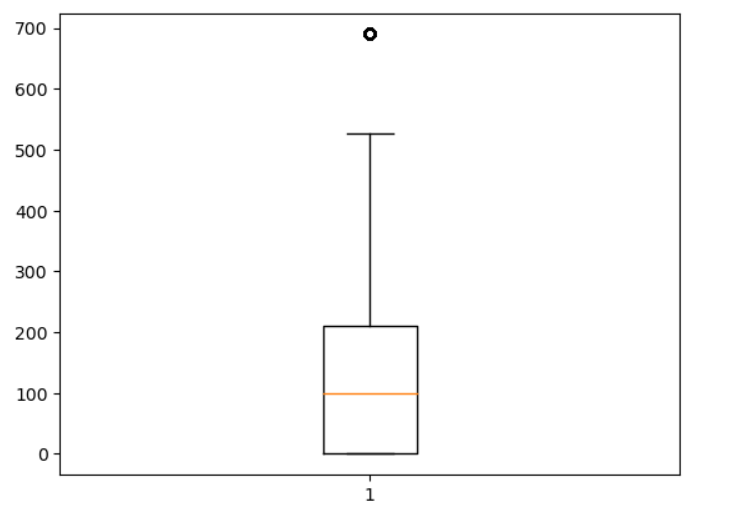
RI = q3 - q1

*#Imputamos el percentil 95 a los valores outliers que están por encima*

df.loc[df['population\_imp'] > q3 + 1.5 \* RI , 'population\_imp'] = per95

plt.boxplot(df['population\_imp'])

plt.show()



* num\_private: Nos parece raro que hay una gran cantidad de valores "0" pero no podemos tomar ninguna decisión ya que en el diccionario de variables no tiene descripción.
* construction\_year: Vemos que tiene valores igual a "0" y obviamente el año de construcción tiene que tener un dato. Imputamos la moda.

valores = df.loc[(df.construction\_year>0),'construction\_year']

moda\_construction = valores.mode().values[0]

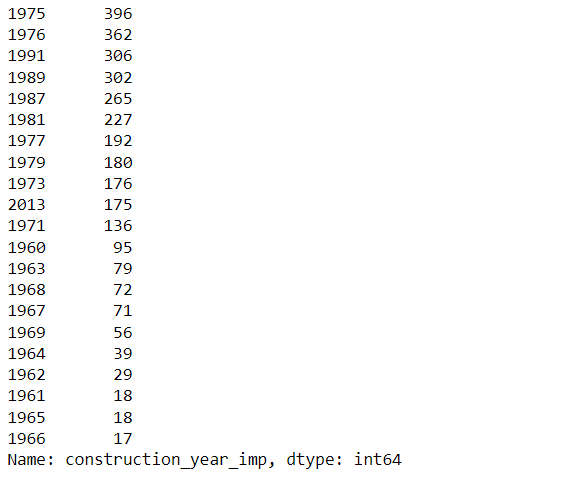
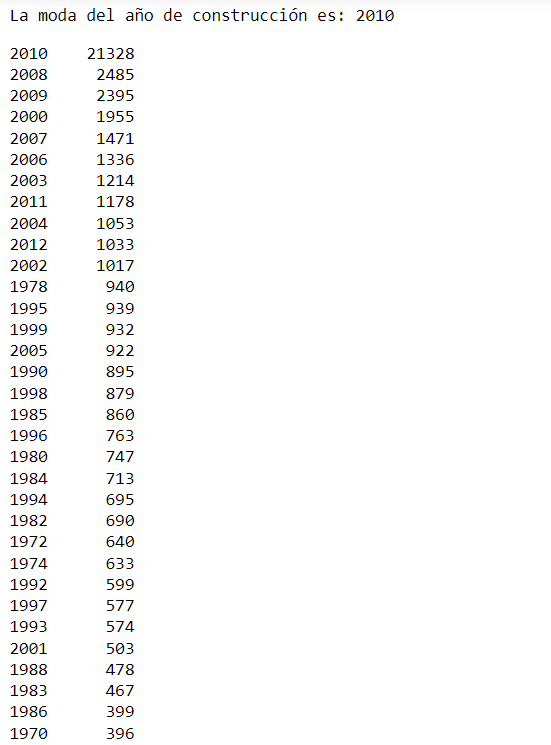
df["construction\_year\_imp"] = df["construction\_year"].apply(lambda x: x if x != 0 else moda\_construction)

*#Eliminamos la columna original para dejar más limpio el df*

df = df.drop(columns="construction\_year")

print("La moda del año de construcción es:", moda\_construction)

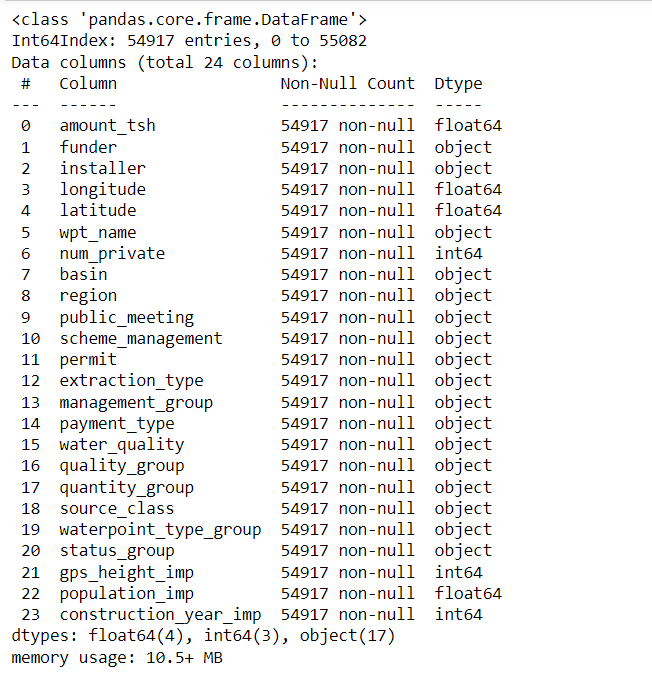
df["construction\_year\_imp"].value\_counts()



En términos generales estas son, en principio, las anomalías que detectamos en el DataSet. De todas maneras, al no tener la suficiente información, consideramos precipitada la idea de tomar cualquier tipo de determinación respecto de ellas. Lo correcto sería realizar las preguntas pertinentes para tomar las decisiones de forma consensuada.

**Transformad todas las variables objetos en categóricas o numéricas (se pondrán todas las filas nulas como una categoría más). Esto lo podéis hacer con un bucle, con apply, poniendo una a una las columnas, …**

df.info()



*#Definimos en la variable "status\_group", "0" para las bombas que no funcionan y "1" para las que sí funcionan*

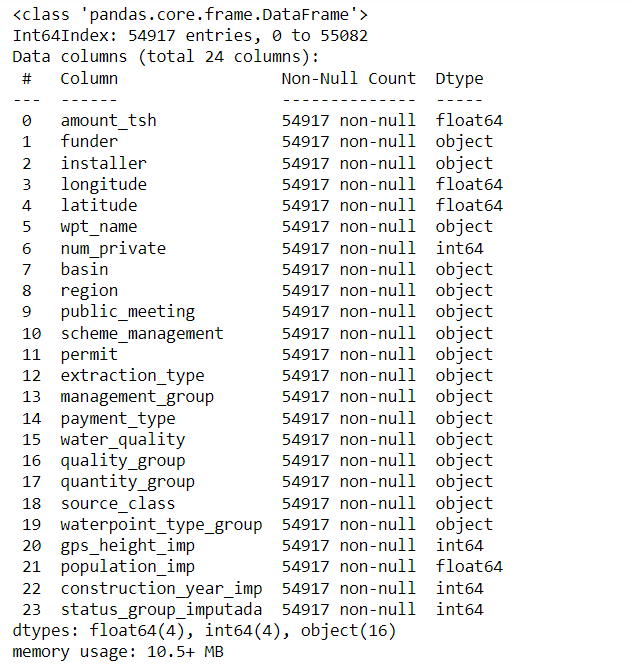
orden\_bombas = {"non functional":0, "functional":1}

df["status\_group\_imputada"] = df["status\_group"]

df["status\_group\_imputada"].replace(orden\_bombas, inplace=True)

df = df.drop(columns= "status\_group")

df.info()



*#Variables que hay que cambiar de tipo:*

*#"funder", "installer", "wpt\_name", "basin", "region", "public\_meeting", "scheme\_management", "permit", "extraction\_type",*

*#"management\_group", "payment\_type", "water\_quality", "quality\_group", "quantity\_group", "source\_class", "waterpoint\_type\_group",*

*#Primero las cambiamos a tipo Category según dice el enunciado del ejercicio*

*#Creamos una función para transformar el tipo de la columna*

def transformar\_columna(df):

for i in df.columns:

if df[i].dtype=='object':

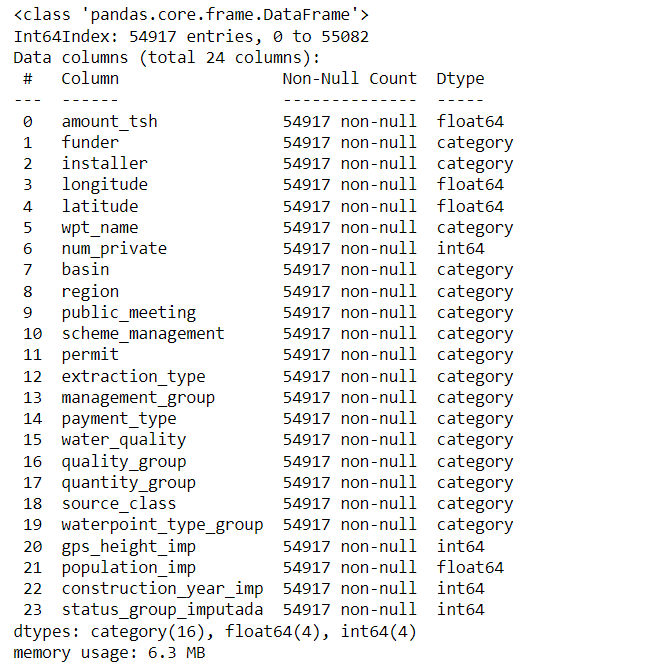
df=df.astype({i:"category"})

return df

*# Aplicar la transformación a todas las columnas del DataFrame*

df = transformar\_columna(df)

df.info()



*#Creamos un dataframe secundario con las variables que vamos a trasformar a Label Encoder*

df\_label = df[["funder", "installer", "wpt\_name", "basin", "region", "public\_meeting", "scheme\_management", "permit", "extraction\_type",

"management\_group", "payment\_type", "water\_quality", "quality\_group", "quantity\_group", "source\_class", "waterpoint\_type\_group"]]

*#Aplicamos un LabelEncoder para convertir a números la variables category*

from sklearn import preprocessing

le = preprocessing.LabelEncoder()

for columna in df\_label.columns:

le.fit(df\_label[columna])

df\_label[columna+'\_imputada'] = le.transform(df[columna])

df\_label = df\_label.drop(columns=["funder", "installer", "wpt\_name", "basin", "region", "public\_meeting", "scheme\_management", "permit", "extraction\_type",

"management\_group", "payment\_type", "water\_quality", "quality\_group", "quantity\_group", "source\_class", "waterpoint\_type\_group"])

df\_label



*#Concatenamos el df con el que estamos trabajando y el df\_label que ya tiene las variables category en numéricas*

df = pd.concat([df, df\_label], axis=1)

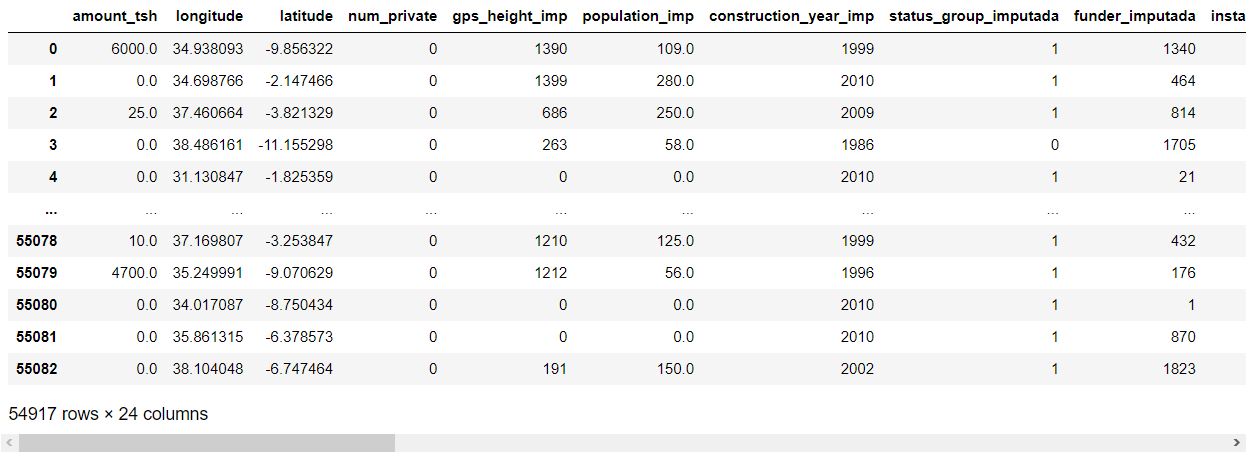
*#Eliminamos las columnas con las variables antes de imputar*

df = df.drop(columns=["funder", "installer", "wpt\_name", "basin", "region", "public\_meeting", "scheme\_management", "permit", "extraction\_type",

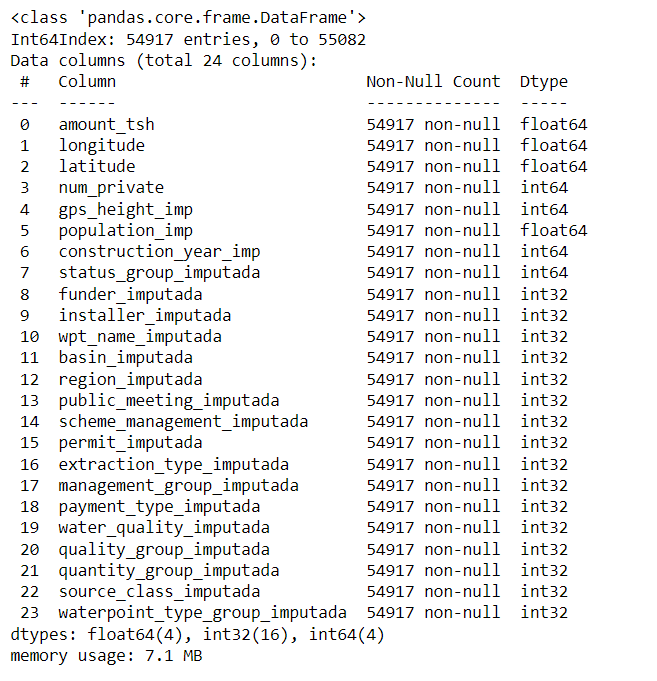
"management\_group", "payment\_type", "water\_quality", "quality\_group", "quantity\_group", "source\_class", "waterpoint\_type\_group"])

*#Mostramos el df donde podemos observar que hemos hecho tratamiento en todas las columnas que llevan \_imp e \_imputada*

df



df.info()



### **2.2. Ejercicio 2**

**Ahora, vamos a entrenar el modelo:**

* **Dividid los datos en variable independiente y target**
* **Dividid el modelo en un conjunto de datos para el test (20%) y otro para el train (80%) y random\_state=42**

x = df.drop(columns="status\_group\_imputada").values

y = df["status\_group\_imputada"].values

*#Dividimos el modelo en entrenamiento y prueba*

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.20, random\_state= 42)

* **Entrenad varios modelos con los datos de train, validadlo con el test y seleccionad el que mejor resultado obtiene.**

Probamos con una REGRESIÓN LOGÍSTICA y evaluamos el resultado:

modelo\_regresion\_logistica = LogisticRegression().fit(x\_train, y\_train)

*#Realizamos las predicciones con el conjunto de prueba de test*

pred\_regresion\_logistica = modelo\_regresion\_logistica.predict(x\_test)

*#Realizamos las predicciones con el conjunto de prueba de entrenamiento*

pred\_regresion\_logistica\_train = modelo\_regresion\_logistica.predict(x\_train)

*#Evaluamos la precisión del modelo*

resultado\_regresion\_logistica = precision\_score(y\_test, pred\_regresion\_logistica)

print("La precisión del modelo es:", resultado\_regresion\_logistica)

*#Evaluamos Accuracy con el test*

accuracy\_regresion\_logistica = accuracy\_score(y\_test, pred\_regresion\_logistica)

print("El accuracy del test del modelo es:", accuracy\_regresion\_logistica)

*#Evaluamos Accuracy con el train*

accuracy\_train\_regresion\_logistica = accuracy\_score(y\_train, pred\_regresion\_logistica\_train)

print("El accuracy del entrenamiento del modelo es:", accuracy\_train\_regresion\_logistica)

*#Evaluamos Recall*

recall\_regresion\_logistica = recall\_score(y\_test, pred\_regresion\_logistica)

print("El recall del modelo es:", recall\_regresion\_logistica)

*#Matriz de confusión*

matriz\_regresion\_logistica = confusion\_matrix(y\_test, pred\_regresion\_logistica)

print(matriz\_regresion\_logistica)

*#Graficamos la matriz de confusión*

def mostrar\_resultados(y\_test, y\_pred):

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(5, 5))

sns.heatmap(conf\_matrix, annot=True, fmt="d")

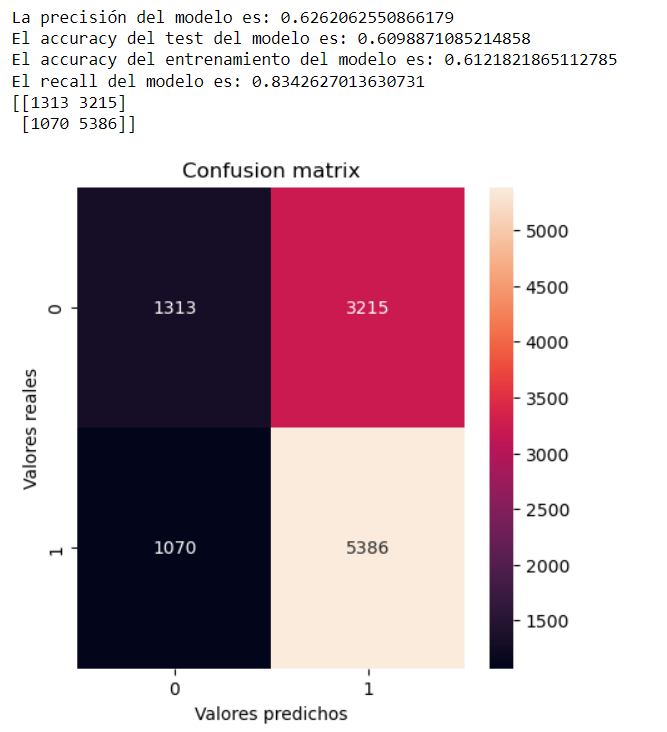
plt.title("Confusion matrix")

plt.ylabel('Valores reales')

plt.xlabel('Valores predichos')

plt.show()

mostrar\_resultados(y\_test, pred\_regresion\_logistica)



Probamos con un ÁRBOL DE DECISIÓN y evaluamos el resultado.

Sacamos la profundidad máxima del árbol para decidir por donde cortar y evitar el sobreajuste.

*#Definimos el modelo*

modelo\_arbol\_decision = DecisionTreeClassifier().fit(x\_train, y\_train)

*#Sacamos la profundidad máxima para ver por donde vamos a podar*

max\_depth = modelo\_arbol\_decision.tree\_.max\_depth

print("Profundidad:", max\_depth)

*#Realizamos las predicciones con el conjunto de prueba*

pred\_arbol\_decision = modelo\_arbol\_decision.predict(x\_test)

*#Realizamos las predicciones con el conjunto de prueba de entrenamiento*

pred\_arbol\_decision\_train = modelo\_arbol\_decision.predict(x\_train)

*#Evaluamos la precisión del modelo*

resultado\_arbol\_decision = precision\_score(y\_test, pred\_arbol\_decision)

print("La precisión del modelo es:", resultado\_arbol\_decision)

*#Evaluamos Accuracy con el test*

accuracy\_arbol\_decision = accuracy\_score(y\_test, pred\_arbol\_decision)

print("El accuracy del test del modelo es:", accuracy\_arbol\_decision)

*#Evaluamos Accuracy con el train*

accuracy\_arbol\_decision\_train = accuracy\_score(y\_train, pred\_arbol\_decision\_train)

print("El accuracy del entrenamiento del modelo es:", accuracy\_arbol\_decision\_train)

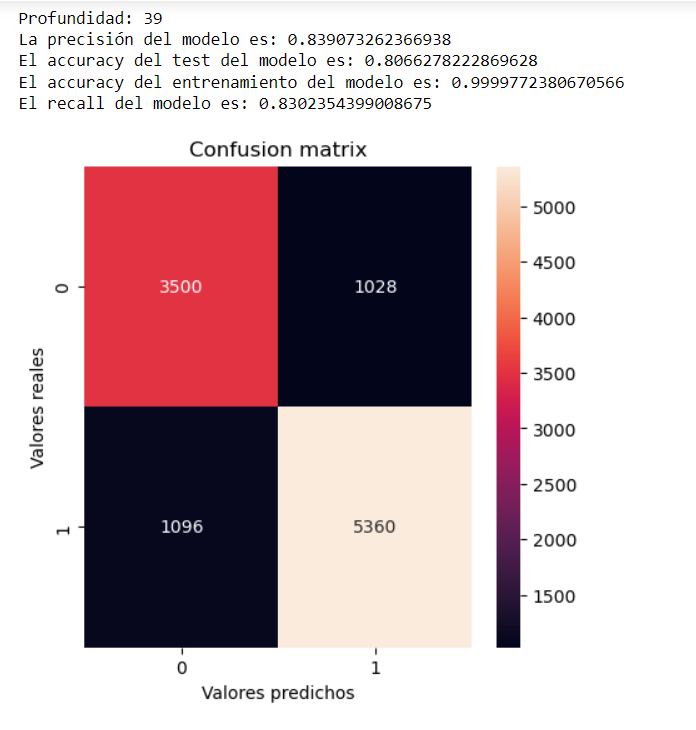
*#Evaluamos Recall*

recall\_arbol\_decision = recall\_score(y\_test, pred\_arbol\_decision)

print("El recall del modelo es:", recall\_arbol\_decision)

*#Graficamos la matriz de confusión*

mostrar\_resultados(y\_test, pred\_arbol\_decision)



Probamos el árbol con una profundidad de 24:

*#Definimos el modelo*

modelo\_arbol\_decision\_24 = DecisionTreeClassifier(max\_depth=24).fit(x\_train, y\_train)

*#Realizamos las predicciones con el conjunto de prueba*

pred\_arbol\_decision\_24 = modelo\_arbol\_decision\_24.predict(x\_test)

*#Realizamos las predicciones con el conjunto de prueba de entrenamiento*

pred\_arbol\_decision\_24\_train = modelo\_arbol\_decision\_24.predict(x\_train)

*#Evaluamos Accuracy con el test*

accuracy\_arbol\_decision\_24 = accuracy\_score(y\_test, pred\_arbol\_decision\_24)

print("El accuracy del test del modelo es:", accuracy\_arbol\_decision\_24)

*#Evaluamos Accuracy con el train*

accuracy\_arbol\_decision\_24\_train = accuracy\_score(y\_train, pred\_arbol\_decision\_24\_train)

print("El accuracy del entrenamiento del modelo es:", accuracy\_arbol\_decision\_24\_train)

Nos entrega:

El accuracy del test del modelo es: 0.8132738528769119

El accuracy del entrenamiento del modelo es: 0.9782851159720484

Probamos el árbol con una profundidad de 20:

*#Definimos el modelo*

modelo\_arbol\_decision\_20 = DecisionTreeClassifier(max\_depth=20).fit(x\_train, y\_train)

*#Realizamos las predicciones con el conjunto de prueba*

pred\_arbol\_decision\_20 = modelo\_arbol\_decision\_20.predict(x\_test)

*#Realizamos las predicciones con el conjunto de prueba de entrenamiento*

pred\_arbol\_decision\_20\_train = modelo\_arbol\_decision\_20.predict(x\_train)

*#Evaluamos Accuracy con el test*

accuracy\_arbol\_decision\_20 = accuracy\_score(y\_test, pred\_arbol\_decision\_20)

print("El accuracy del test del modelo es:", accuracy\_arbol\_decision\_20)

*#Evaluamos Accuracy con el train*

accuracy\_arbol\_decision\_20\_train = accuracy\_score(y\_train, pred\_arbol\_decision\_20\_train)

print("El accuracy del entrenamiento del modelo es:", accuracy\_arbol\_decision\_20\_train)

Nos entrega:

El accuracy del test del modelo es: 0.8156409322651129

El accuracy del entrenamiento del modelo es: 0.945917647326611

Probamos el árbol con una profundidad de 8:

*#Definimos el modelo*

modelo\_arbol\_decision\_8 = DecisionTreeClassifier(max\_depth=8).fit(x\_train, y\_train)

*#Realizamos las predicciones con el conjunto de prueba*

pred\_arbol\_decision\_8 = modelo\_arbol\_decision\_8.predict(x\_test)

*#Realizamos las predicciones con el conjunto de prueba de entrenamiento*

pred\_arbol\_decision\_8\_train = modelo\_arbol\_decision\_8.predict(x\_train)

*#Evaluamos Accuracy con el test*

accuracy\_arbol\_decision\_8 = accuracy\_score(y\_test, pred\_arbol\_decision\_8)

print("El accuracy del test del modelo es:", accuracy\_arbol\_decision\_8)

*#Evaluamos Accuracy con el train*

accuracy\_arbol\_decision\_8\_train = accuracy\_score(y\_train, pred\_arbol\_decision\_8\_train)

print("El accuracy del entrenamiento del modelo es:", accuracy\_arbol\_decision\_8\_train)

Nos entrega:

El accuracy del test del modelo es: 0.7826839038601602

El accuracy del entrenamiento del modelo es: 0.7884505952245464

Concluimos que el Árbol de decisión no es el mejor ya que todavía no hemos hecho una selección de variables.

Sin embargo decidimos que lo mejor sería una profundidad de 8 ya que la eficacia no varía mucho y evitamos mucho sobreajuste.

Probamos con un RANDOM FOREST y evaluamos el resultado:

*#Definimos el modelo*

modelo\_random\_forest = RandomForestClassifier().fit(x\_train, y\_train)

*#Realizamos las predicciones con el conjunto de prueba*

pred\_random\_forest = modelo\_random\_forest.predict(x\_test)

*#Realizamos las predicciones con el conjunto de prueba de entrenamiento*

pred\_random\_forest\_train = modelo\_random\_forest.predict(x\_train)

*#Evaluamos la precisión del modelo*

resultado\_random\_forest = precision\_score(y\_test, pred\_random\_forest)

print("La precisión del modelo es:", resultado\_random\_forest)

*#Evaluamos Accuracy con el test*

accuracy\_random\_forest = accuracy\_score(y\_test, pred\_random\_forest)

print("El accuracy del test del modelo es:", accuracy\_random\_forest)

*#Evaluamos Accuracy con el train*

accuracy\_random\_forest\_train = accuracy\_score(y\_train, pred\_random\_forest\_train)

print("El accuracy del entrenamiento del modelo es:", accuracy\_random\_forest\_train)

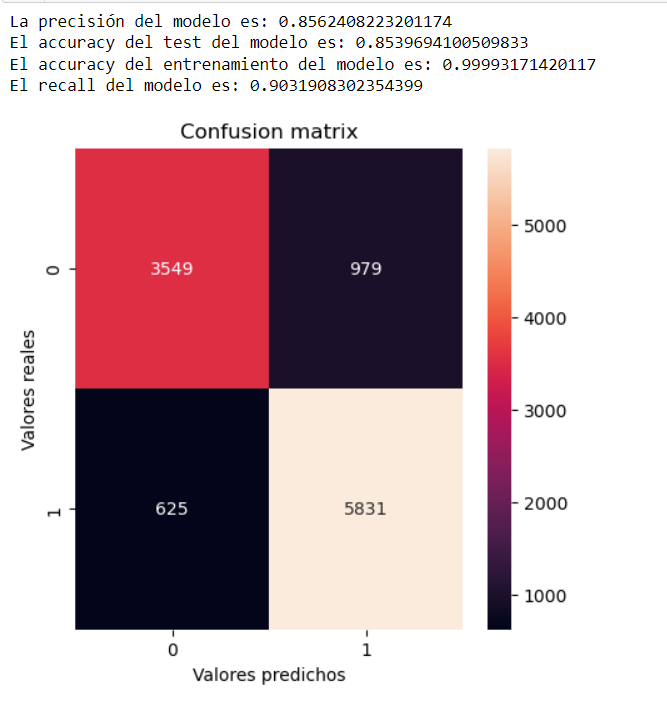
*#Evaluamos Recall*

recall\_random\_forest = recall\_score(y\_test, pred\_random\_forest)

print("El recall del modelo es:", recall\_random\_forest)

*#Graficamos la matriz de confusión*

mostrar\_resultados(y\_test, pred\_random\_forest)



Probamos Random Forest con una profundidad de 8 que es la que hemos decidido antes y 10 árboles:

*#Definimos el modelo*

modelo\_random\_forest\_8 = RandomForestClassifier(n\_estimators = 10, max\_depth=8).fit(x\_train, y\_train)

*#Realizamos las predicciones con el conjunto de prueba*

pred\_random\_forest\_8 = modelo\_random\_forest\_8.predict(x\_test)

*#Realizamos las predicciones con el conjunto de prueba de entrenamiento*

pred\_random\_forest\_8\_train = modelo\_random\_forest\_8.predict(x\_train)

*#Evaluamos Accuracy con el test*

accuracy\_random\_forest\_8 = accuracy\_score(y\_test, pred\_random\_forest\_8)

print("El accuracy con el test del modelo es:", accuracy\_random\_forest\_8)

*#Evaluamos Accuracy con el train*

accuracy\_random\_forest\_8\_train = accuracy\_score(y\_train, pred\_random\_forest\_8\_train)

print("El accuracy del entrenamiento del modelo es:", accuracy\_random\_forest\_8\_train)

Nos entrega:

El accuracy con el test del modelo es: 0.7892388929351785

El accuracy del entrenamiento del modelo es: 0.7908633601165411

Después de probar Regresión Logística, Árbol de decisión y Random Forest, decidimos que el mejor modelo que no esté sobreajustado es Random Forest con profundidad 8 y 10 árboles.

**Una vez hecho esto, responded a las siguientes preguntas:**

* **¿Qué score da el de entrenamiento y con el test?**
* **¿Creéis que puede tener sobreajuste (overfitting) o infraajuste (underfitting)?**

Al comparar en todos los modelos el accuracy con el test y con el train vemos que cuanta menos profundidad menos sobreajustado está el modelo.

Creemos que todos los modelos pueden tener sobreajuste porque no hemos hecho ninguna selección de variables.

Tanto en el Árbol de decisión como en el Random Forest hemos intentado reducir un poco el sobreajuste limitando la profundidad, aunque como vamos arrastrando el problema de las variables no solucionamos mucho.

### **2.3. Ejercicio 3**

**Seleccionad las 21 variables que más influyen en la predicción y entrenad de nuevo el modelo. ¿Mejora?**

**Usadlas para sacar los scoring ['accuracy', 'precision', 'recall'] del conjunto de train:**

* **¿Interpreta accuracy?**
* **¿Interpreta precision?**
* **¿Interpreta recall?**
* **¿Predice mejor los positivos o los negativos?**

Probamos con dos métodos de selección de variables para ver la importancia respecto al target

*#FEATURE IMPORTANCE MUESTRA IMPORTANCIA DE LAS VARIABLES*

x2 = df.drop(['status\_group\_imputada'], axis=1)

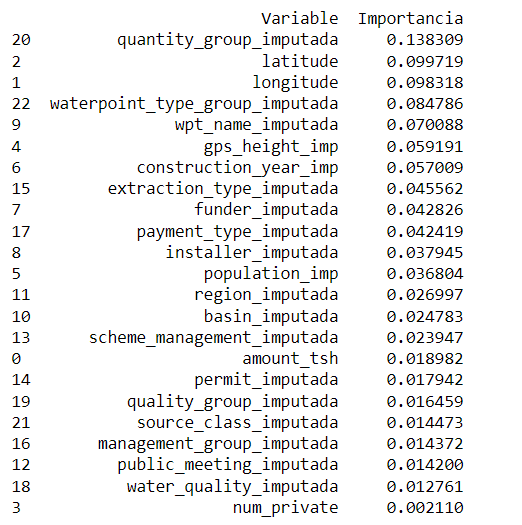
y2 = df['status\_group\_imputada']

feature = ExtraTreesClassifier().fit(x2, y2)

importance\_table = pd.DataFrame({'Variable': x2.columns, 'Importancia': feature.feature\_importances\_})

importance\_table\_sorted = importance\_table.sort\_values(by='Importancia', ascending=False)

print(importance\_table\_sorted)



*#FEATURE IMPORTANCE MUESTRA IMPORTANCIA DE LAS VARIABLES*

x2 = df.drop(['status\_group\_imputada'], axis=1)

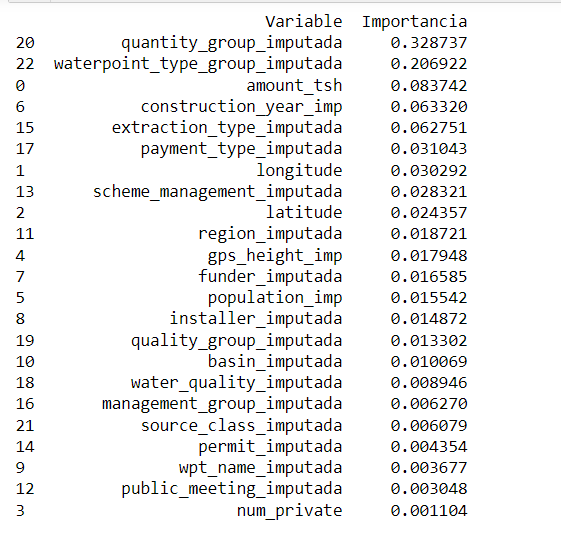
y2 = df['status\_group\_imputada']

feature2 = modelo\_random\_forest\_8

importance\_table2 = pd.DataFrame({'Variable': x2.columns, 'Importancia': feature2.feature\_importances\_})

importance\_table\_sorted2 = importance\_table2.sort\_values(by='Importancia', ascending=False)

print(importance\_table\_sorted2)



Buscamos la correlación entre variables para descartar colinealidad

*#Sacamos la matriz de correlación de Pearson para ver relación entre variables*

pearson = df.corr(method="pearson")

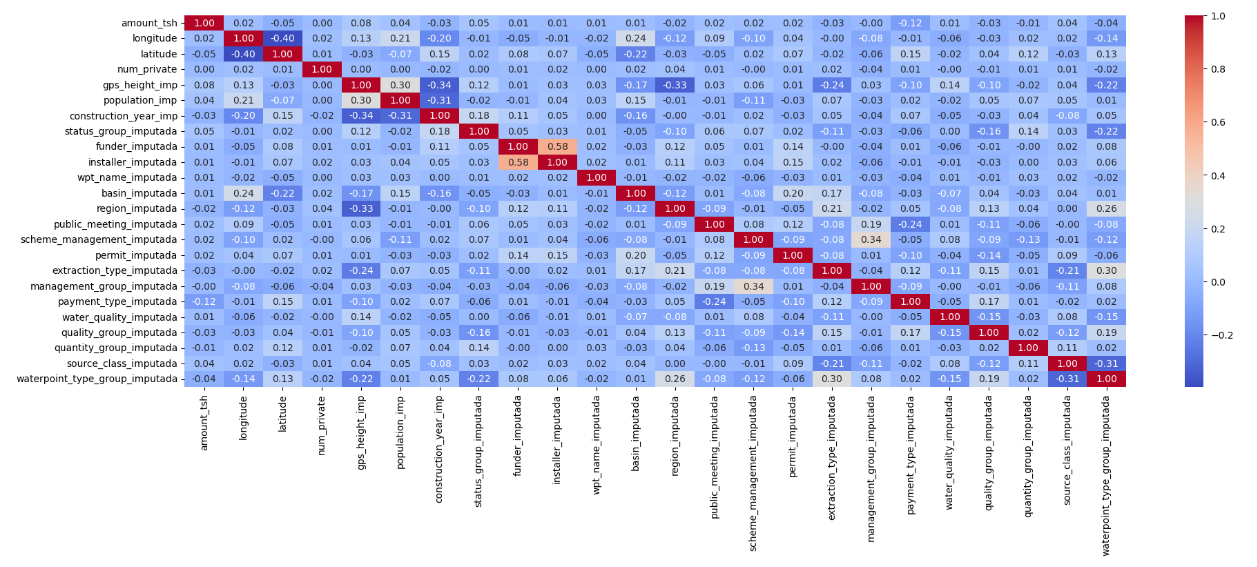
*# Ajustar el tamaño de la figura*

fig, ax = plt.subplots(figsize=(22, 7))

*# Generar el mapa de calor*

sns.heatmap(pearson, annot=True, fmt=".2f", cmap="coolwarm", ax=ax)

plt.show()



Decidimos eliminar "installer\_imputada" y "management\_group\_imputada" porque no tienen una gran importancia con el target y tienen correlación con "funder\_imputada" y con "scheme\_management\_imputada" respectivamente

df\_final = df.drop(columns=["installer\_imputada", "management\_group\_imputada"])

Ahora con nuestra selección de 21 variables probamos la Regresión Logística, el Árbol de decisión con profundidad 8 y el Random Forest con profundidad 8

Dividimos el nuevo df en entrenamiento y test

x = df\_final.drop(columns="status\_group\_imputada").values

y = df\_final["status\_group\_imputada"].values

*#Dividimos el modelo en entrenamiento y prueba*

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.20, random\_state= 42)

REGRESIÓN LOGÍSTICA

modelo\_regresion\_logistica = LogisticRegression(max\_iter=5000).fit(x\_train, y\_train)

*#Realizamos las predicciones con el conjunto de prueba*

pred\_regresion\_logistica = modelo\_regresion\_logistica.predict(x\_test)

*#Evaluamos la precisión del modelo*

resultado\_regresion\_logistica = precision\_score(y\_test, pred\_regresion\_logistica)

print("La precisión del modelo es:", resultado\_regresion\_logistica)

*#Evaluamos Accuracy*

accuracy\_regresion\_logistica = accuracy\_score(y\_test, pred\_regresion\_logistica)

print("El accuracy del modelo es:", accuracy\_regresion\_logistica)

*#Evaluamos Recall*

recall\_regresion\_logistica = recall\_score(y\_test, pred\_regresion\_logistica)

print("El recall del modelo es:", recall\_regresion\_logistica)

*#Matriz de confusión*

matriz\_regresion\_logistica = confusion\_matrix(y\_test, pred\_regresion\_logistica)

print(matriz\_regresion\_logistica)

*#Graficamos la matriz de confusión*

def mostrar\_resultados(y\_test, y\_pred):

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

plt.figure(figsize=(5, 5))

sns.heatmap(conf\_matrix, annot=True, fmt="d")

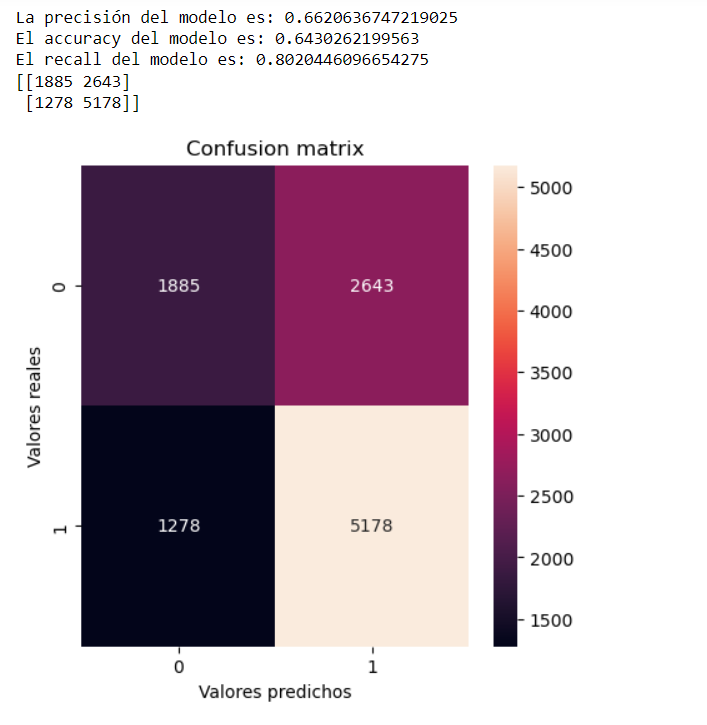
plt.title("Confusion matrix")

plt.ylabel('Valores reales')

plt.xlabel('Valores predichos')

plt.show()

mostrar\_resultados(y\_test, pred\_regresion\_logistica)



Definimos las bombas funcionales con el valor 1 y las no funcionales con 0. Esto nos permite saber que cuando las metricas toman el valor 0 corresponden a los negativos y 1 a los positivos.

* La precisión es de un 66 % y esto indica la cantidad de aciertos de bombas funcionales con respecto a las pedicciones positivas totales.
* El accuracy es de un 64 % y esto indica el porcentaje de registros correctamente clasificados.
* En este caso el recall nos indica que nuestro modelo tiene la capacidad de detectar 80% de las bombas funcionales correctamente respecto del total de bombas funcionales.
* La matriz de confusión nos ayuda a entender los porcentajes entregados por cada métrica.

En resumen, el modelo tiene una mayor capacidad para identificar correctamente los resultados positivos en comparación con los negativos. Aunque puede haber algunos falsos positivos, en general el modelo logra capturar la mayoría de los resultados positivos (80% de recall) mientras mantiene una precisión del 66% en las predicciones positivas.

ÁRBOL PROFUNDIDAD 8

*#Definimos el modelo*

modelo\_arbol\_decision\_8 = DecisionTreeClassifier(max\_depth=8).fit(x\_train, y\_train)

*#Realizamos las predicciones con el conjunto de prueba*

pred\_arbol\_decision\_8 = modelo\_arbol\_decision\_8.predict(x\_test)

*#Evaluamos la precisión del modelo*

resultado\_arbol\_decision\_8 = precision\_score(y\_test, pred\_arbol\_decision\_8)

print("La precisión del modelo es:", resultado\_arbol\_decision\_8)

*#Evaluamos Accuracy*

accuracy\_arbol\_decision\_8 = accuracy\_score(y\_test, pred\_arbol\_decision\_8)

print("El accuracy del modelo es:", accuracy\_arbol\_decision\_8)

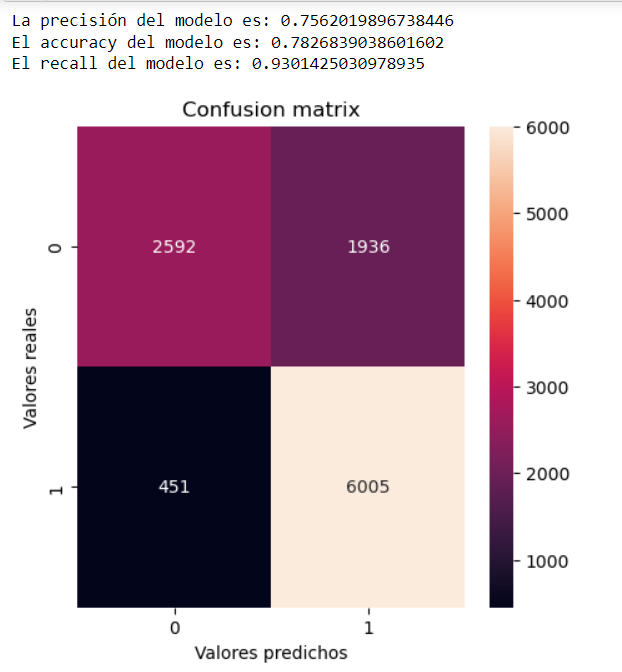
*#Evaluamos Recall*

recall\_arbol\_decision\_8 = recall\_score(y\_test, pred\_arbol\_decision\_8)

print("El recall del modelo es:", recall\_arbol\_decision\_8)

*#Graficamos la matriz de confusión*

mostrar\_resultados(y\_test, pred\_arbol\_decision\_8)



* La precisión es de un 75.6% y esto indica la cantidad de aciertos de bombas funcionales con respecto a las pedicciones positivas totales.
* El accuracy es de un 78.2% y esto indica el porcentaje de registros correctamente clasificados.
* En este caso el recall nos indica que nuestro modelo tiene la capacidad de detectar 93% de las bombas funcionales correctamente respecto del total de bombas funcionales.
* La matriz de confusión nos ayuda a entender los porcentajes entregados por cada métrica.

En resumen, el modelo tiene una mayor capacidad para identificar correctamente los resultados positivos en comparación con los negativos. Aunque puede haber algunos falsos positivos, en general el modelo logra capturar la mayoría de los resultados positivos (93% de recall) mientras mantiene una precisión del 75.6% en las predicciones positivas.

RANDOM FOREST PROFUNDIDAD 8

*#Definimos el modelo*

modelo\_random\_forest\_8 = RandomForestClassifier(n\_estimators = 10, max\_depth=8).fit(x\_train, y\_train)

*#Realizamos las predicciones con el conjunto de prueba*

pred\_random\_forest\_8 = modelo\_random\_forest\_8.predict(x\_test)

*#Evaluamos la precisión del modelo*

resultado\_random\_forest\_8 = precision\_score(y\_test, pred\_random\_forest\_8)

print("La precisión del modelo es:", resultado\_random\_forest\_8)

*#Evaluamos Accuracy*

accuracy\_random\_forest\_8 = accuracy\_score(y\_test, pred\_random\_forest\_8)

print("El accuracy del modelo es:", accuracy\_random\_forest\_8)

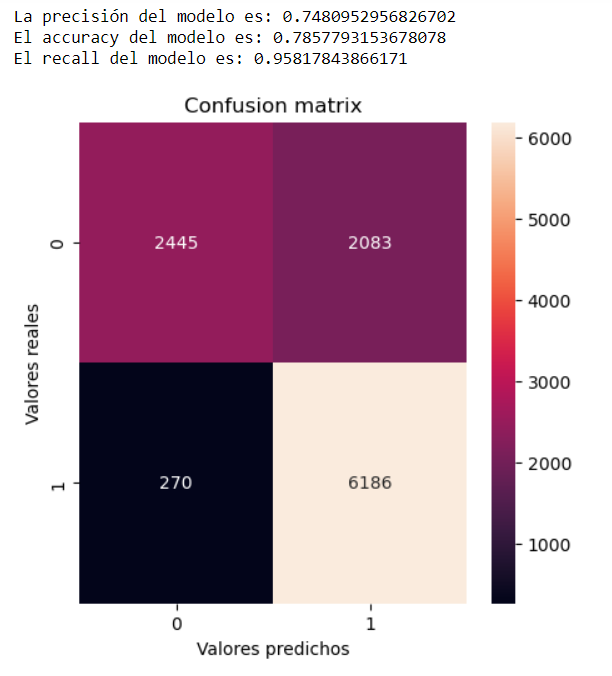
*#Evaluamos Recall*

recall\_random\_forest\_8 = recall\_score(y\_test, pred\_random\_forest\_8)

print("El recall del modelo es:", recall\_random\_forest\_8)

*#Graficamos la matriz de confusión*

mostrar\_resultados(y\_test, pred\_random\_forest\_8)



* La precisión es de un 74.8% y esto indica la cantidad de aciertos de bombas funcionales con respecto a las pedicciones positivas totales.
* El accuracy es de un 78.5% y esto indica el porcentaje de registros correctamente clasificados.
* En este caso el recall nos indica que nuestro modelo tiene la capacidad de detectar 95.8% de las bombas funcionales correctamente respecto del total de bombas funcionales.
* La matriz de confusión nos ayuda a entender los porcentajes entregados por cada métrica.

En resumen, el modelo tiene una mayor capacidad para identificar correctamente los resultados positivos en comparación con los negativos. Aunque puede haber algunos falsos positivos, en general el modelo logra capturar la mayoría de los resultados positivos (95.8% de recall) mientras mantiene una precisión del 74.8% en las predicciones positivas.

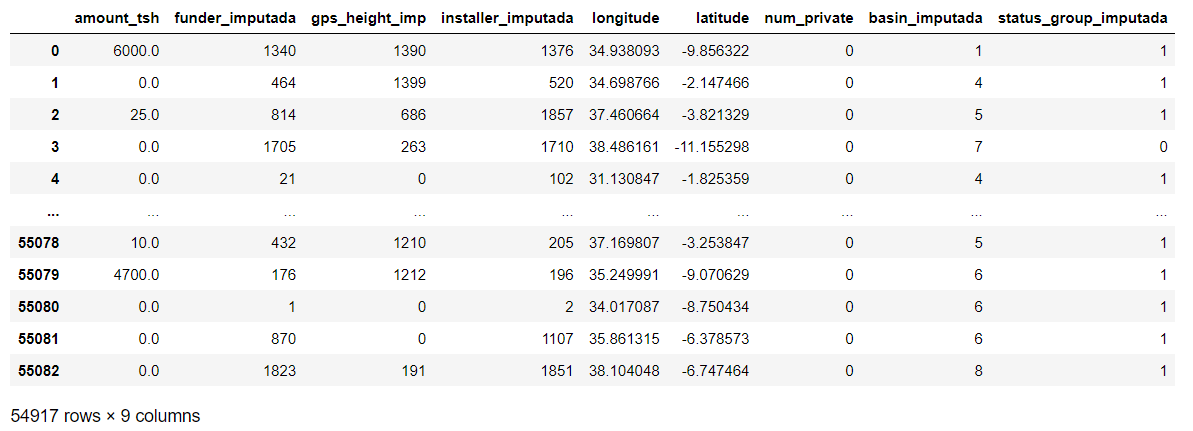
Podemos concluir que los modelos han mejorado un poco y estarán menos sobreajustados pero tampoco una gran diferencia en cuanto a predicción.

### **2.4. Ejercicio 4**

**Validad la correlación con uno o más gráficos con las columnas ['amount\_tsh', 'funder', 'gps\_height', 'installer', 'longitude', 'latitude', 'num\_private', 'basin','status\_group']**

df\_ejercicio\_4 = df[['amount\_tsh', 'funder\_imputada', 'gps\_height\_imp', 'installer\_imputada', 'longitude', 'latitude', 'num\_private', 'basin\_imputada','status\_group\_imputada']]

df\_ejercicio\_4



*#Sacamos la matriz de correlación de Pearson para ver relación entre variables*

pearson = df\_ejercicio\_4.corr(method="pearson")

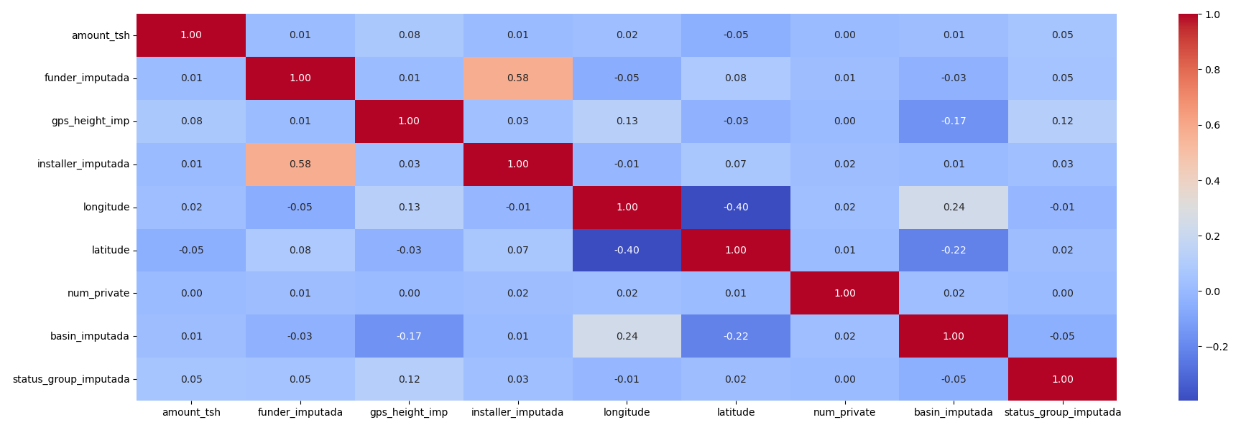
*# Ajustar el tamaño de la figura*

fig, ax = plt.subplots(figsize=(22, 7))

*# Generar el mapa de calor*

sns.heatmap(pearson, annot=True, fmt=".2f", cmap="coolwarm", ax=ax)

plt.show()



X = df\_ejercicio\_4.drop(['status\_group\_imputada'], axis=1)

y = df\_ejercicio\_4['status\_group\_imputada']

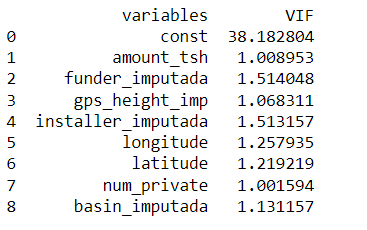
X = sm.add\_constant(X)

vif = pd.DataFrame()

vif["variables"] = X.columns

vif["VIF"] = [variance\_inflation\_factor(X.values, i) for i in range(X.shape[1])]

print(vif)



**Después, haced un gráfico, el que consideréis adecuado, para detectar outliers en population y gps\_height ¿alguno tiene outliers? De ser así, eliminadlos con el método de Inter cuartil con la columna o columnas con datos atípicos. ¿El modelo ha mejorado? Recordad que hay que volver a sacar los valores x e y (test y train).**

Como decidimos tratar los outliers desde el principio, no generamos modelos diferentes en cuanto a las variables population y gps\_height

**Para terminar, usad la búsqueda de hiperparámetro para ajustar al modelo seleccionado (buscad en** [**https://scikit-learn.org/**](https://scikit-learn.org/) **o en la página del modelo usado).**

Finalmente decidimos probar un modelo con menos variables para intentar conseguir un buen resultado con el menor de los sobreajustes, esto lo convertirá en un modelo que se podrá usar durante más tiempo ya que valdrá para diferentes circunstancias.

En este modelo usamos la búsqueda del mejor hiperparámetro.

*#Atendiendo a la importancia de variables que tenemos a continuación,decidimos quedarnos con las 6 primeras.*

"""

20 quantity\_group\_imputada 0.401684

22 waterpoint\_type\_group\_imputada 0.174342

0 amount\_tsh 0.072169

6 construction\_year\_imp 0.062822

15 extraction\_type\_imputada 0.044220

11 region\_imputada 0.028760

1 longitude 0.028016

13 scheme\_management\_imputada 0.027102

2 latitude 0.023880

17 payment\_type\_imputada 0.020581

5 population\_imp 0.019081

4 gps\_height\_imp 0.018931

7 funder\_imputada 0.016050

18 water\_quality\_imputada 0.013348

10 basin\_imputada 0.012090

8 installer\_imputada 0.011180

16 management\_group\_imputada 0.007015

9 wpt\_name\_imputada 0.004407

19 quality\_group\_imputada 0.003540

14 permit\_imputada 0.003335

12 public\_meeting\_imputada 0.003229

21 source\_class\_imputada 0.003183

3 num\_private 0.001034

"""

df\_grupal = df[["quantity\_group\_imputada", "waterpoint\_type\_group\_imputada", "amount\_tsh", "construction\_year\_imp",

"extraction\_type\_imputada", "region\_imputada", "status\_group\_imputada"]]

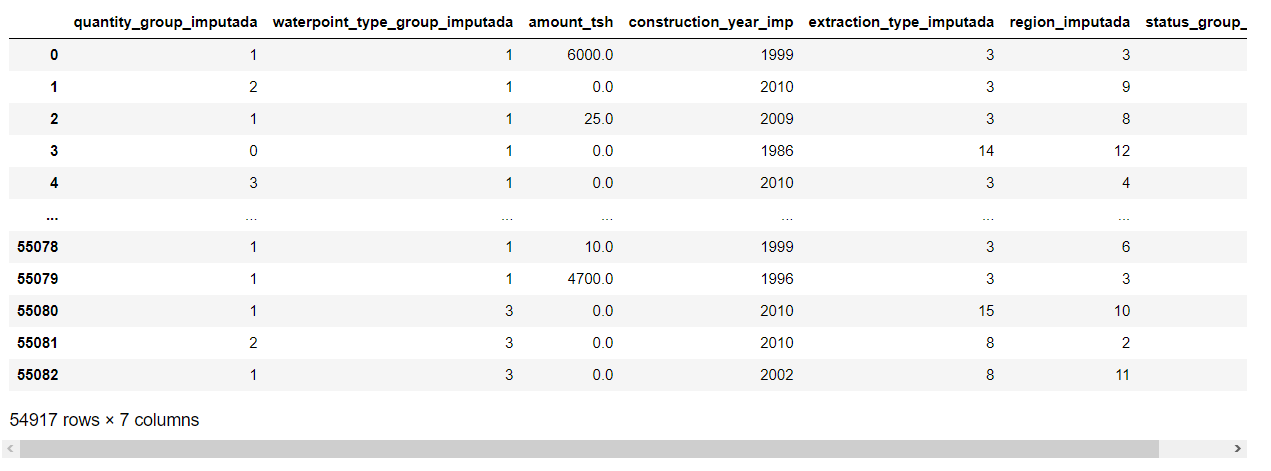
x = df\_grupal.drop(columns="status\_group\_imputada").values

y = df\_grupal["status\_group\_imputada"].values

*#Dividimos el modelo en entrenamiento y prueba*

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.20, random\_state= 42)

df\_grupal



Utilizamos el método GridSearch para ver que hiperparámetros propone:

*#Definimos el modelo*

modelo\_grupal\_random\_forest = RandomForestClassifier().fit(x\_train, y\_train)

*#Realizamos las predicciones con el conjunto de prueba*

pred\_random\_forest\_grupal = modelo\_grupal\_random\_forest.predict(x\_test)

*#Configuramos las parametrizaciones*

param\_grid = {"max\_depth":[2, 4, 8, 12, 24, 39], "criterion":["gini","entropy"]}

*#Entrenamos el modelo con la configuración*

rsearch = GridSearchCV (estimator=modelo\_grupal\_random\_forest, param\_grid=param\_grid)

rsearch.fit(x,y)

*#Vemos la puntuación del modelo, la mejor puntuación y configuración*

print("Puntuación del modelo:", rsearch.score(x,y))

print("Mejor puntuación:", rsearch.best\_score\_)

print("Configuración:", rsearch.best\_params\_)

*#Evaluamos la precisión del modelo*

resultado\_random\_forest\_grupal = precision\_score(y\_test, pred\_random\_forest\_grupal)

print("La precisión del modelo es:", resultado\_random\_forest\_grupal)

*#Evaluamos Accuracy*

accuracy\_random\_forest\_grupal = accuracy\_score(y\_test, pred\_random\_forest\_grupal)

print("El accuracy del modelo es:", accuracy\_random\_forest\_grupal)

*#Evaluamos Recall*

recall\_random\_forest\_grupal = recall\_score(y\_test, pred\_random\_forest\_grupal)

print("El recall del modelo es:", recall\_random\_forest\_grupal)

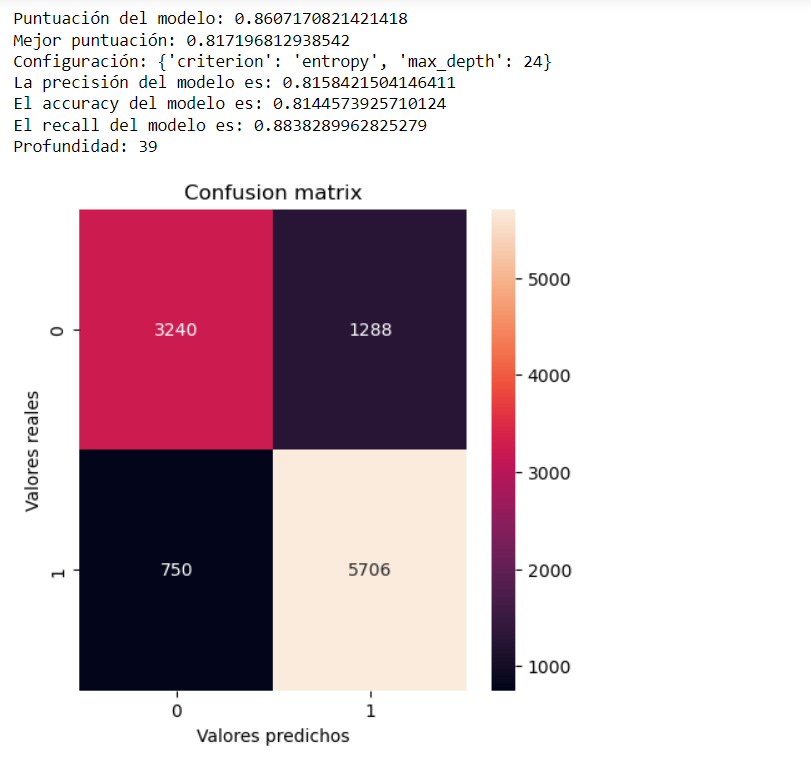
*#Sacamos la profundidad máxima para ver por donde vamos a podar*

profundidad\_maxima = max([estimator.tree\_.max\_depth for estimator in modelo\_grupal\_random\_forest.estimators\_])

print("Profundidad:", max\_depth)

*#Graficamos la matriz de confusión*

mostrar\_resultados(y\_test, pred\_random\_forest\_grupal)



El resultado es {'criterion': 'gini', 'max\_depth': 24}

Utilizamos el método RandomizedSearchCV para ver que hiperparámetros propone:

*#Definimos el modelo*

modelo\_grupal\_random\_forest = RandomForestClassifier().fit(x\_train, y\_train)

*#Realizamos las predicciones con el conjunto de prueba*

pred\_random\_forest\_grupal = modelo\_grupal\_random\_forest.predict(x\_test)

*#Entrenamos el modelo con la configuración*

rsearch = RandomizedSearchCV (modelo\_grupal\_random\_forest, param\_grid)

rsearch.fit(x,y)

*#Vemos la puntuación del modelo, la mejor puntuación y configuración*

print("Puntuación del modelo:", rsearch.score(x,y))

print("Mejor puntuación:", rsearch.best\_score\_)

print("Configuración:", rsearch.best\_params\_)

*#Evaluamos la precisión del modelo*

resultado\_random\_forest\_grupal = precision\_score(y\_test, pred\_random\_forest\_grupal)

print("La precisión del modelo es:", resultado\_random\_forest\_grupal)

*#Evaluamos Accuracy*

accuracy\_random\_forest\_grupal = accuracy\_score(y\_test, pred\_random\_forest\_grupal)

print("El accuracy del modelo es:", accuracy\_random\_forest\_grupal)

*#Evaluamos Recall*

recall\_random\_forest\_grupal = recall\_score(y\_test, pred\_random\_forest\_grupal)

print("El recall del modelo es:", recall\_random\_forest\_grupal)

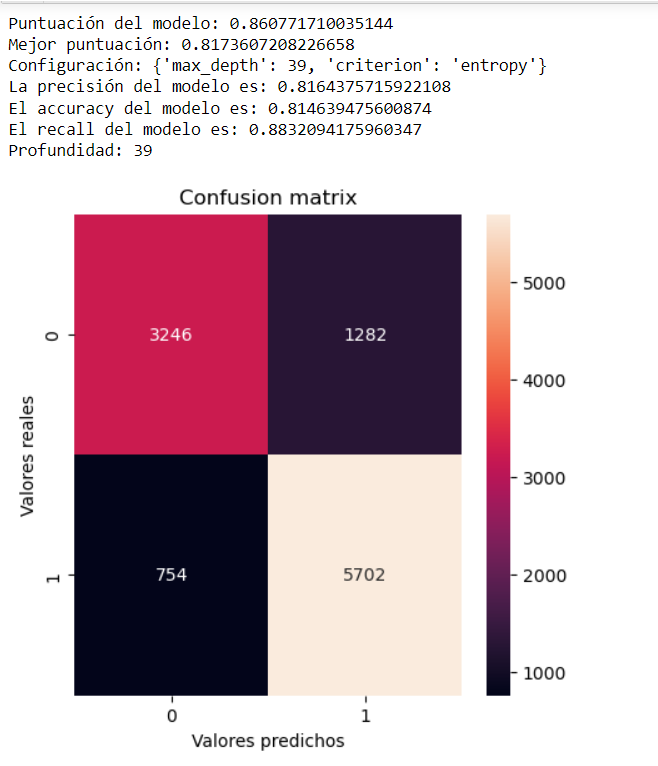
*#Sacamos la profundidad máxima para ver por donde vamos a podar*

profundidad\_maxima = max([estimator.tree\_.max\_depth for estimator in modelo\_grupal\_random\_forest.estimators\_])

print("Profundidad:", max\_depth)

*#Graficamos la matriz de confusión*

mostrar\_resultados(y\_test, pred\_random\_forest\_grupal)



El resultado es: {'max\_depth': 39, 'criterion': 'entropy'}

No nos convencen los resultados que nos dan estos algoritmos automáticos y decidimos probar manualmente nosotros:

*#Definimos el modelo*

modelo\_grupal\_random\_forest = RandomForestClassifier(n\_estimators = 5, max\_depth=12).fit(x\_train, y\_train)

*#Realizamos las predicciones con el conjunto de prueba*

pred\_random\_forest\_grupal = modelo\_grupal\_random\_forest.predict(x\_test)

*#Realizamos las predicciones con el conjunto de prueba de entrenamiento*

pred\_random\_forest\_grupal\_train = modelo\_grupal\_random\_forest.predict(x\_train)

*#Evaluamos la precisión del modelo*

resultado\_random\_forest\_grupal = precision\_score(y\_test, pred\_random\_forest\_grupal)

print("La precisión del modelo es:", resultado\_random\_forest\_grupal)

*#Evaluamos Accuracy con el test*

accuracy\_random\_forest\_grupal = accuracy\_score(y\_test, pred\_random\_forest\_grupal)

print("El accuracy del test del modelo es:", accuracy\_random\_forest\_grupal)

*#Evaluamos Accuracy con el train*

accuracy\_random\_forest\_grupal\_train = accuracy\_score(y\_train, pred\_random\_forest\_grupal\_train)

print("El accuracy del entrenamiento del modelo es:", accuracy\_random\_forest\_grupal\_train)

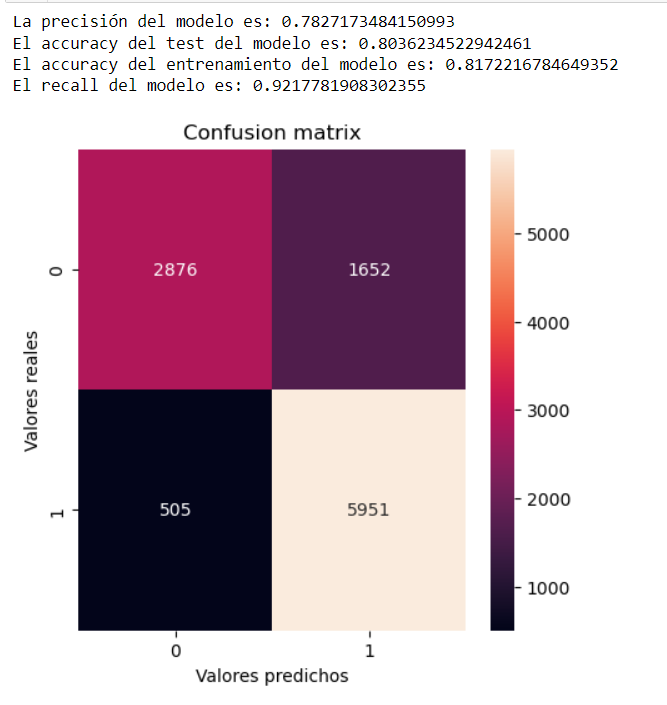
*#Evaluamos Recall*

recall\_random\_forest\_grupal = recall\_score(y\_test, pred\_random\_forest\_grupal)

print("El recall del modelo es:", recall\_random\_forest\_grupal)

*#Graficamos la matriz de confusión*

mostrar\_resultados(y\_test, pred\_random\_forest\_grupal)



Vemos que a mayor profundidad obtenemos un mejor Accuracy y una mejor precisión pero sacrificamos sensibilidad, y al contrario a menor profundidad tendremos una mejor sensibilidad, además de que conseguimos un menor sobreajuste.

Finalmente decidimos escoger un Random Forest, con 5 árboles y profundidad 12 porque al sacar las comprobaciones vemos que obtenemos una buena precisión y Accuracy sin restar tanta sensibilidad por lo que nos parece un modelo muy equilibrado.

También hemos hecho la prueba con el train para ver que el modelo no está muy sobreajustado.

Aún así la mejor praxis sería hablar con el cliente para ver cuales son sus prioridades:

* Evitar que bombas no funcionales sean clasificadas como funcionales, entonces atenderíamos a un modelo más preciso.
* Evitar que bombas funcionales no sean identificadas correctamente, entonces atenderíamos a un modelo más sensible.