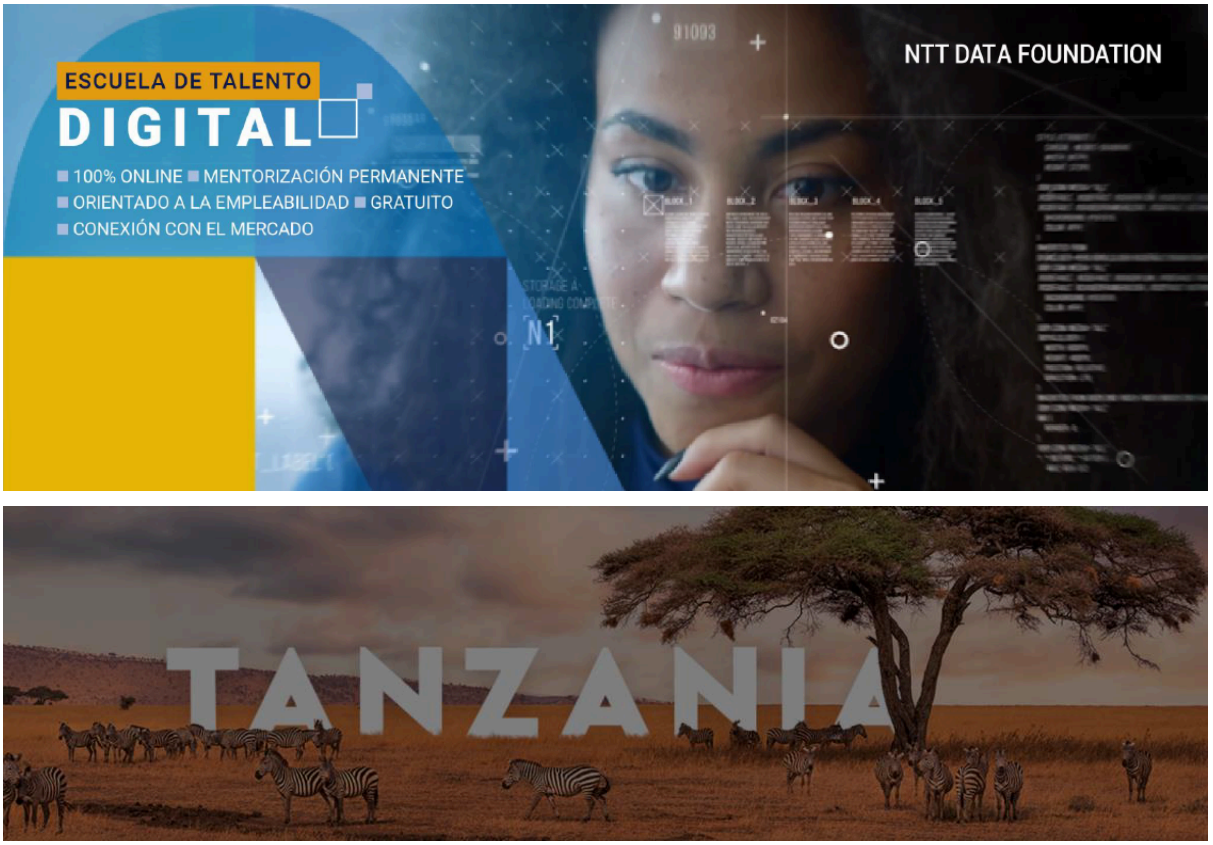
Reto de Análisis de Datos con Python realizado en el curso de la Escuela de Talento Digital de NTT Data Foundation

BOMBAS DE AGUA DE TANZANIA



Alberto Zambrano

INTRUDUCCIÓN

Tanzania se enfrenta a inmensos problemas relacionados con sus sistemas de agua y saneamiento. Casi el 43% de la población carece de acceso al agua potable básica, y sólo un 25% utiliza un saneamiento gestionado de forma segura. Más del 70% de las catástrofes naturales del país están relacionadas con el cambio climático y vinculadas a las sequías recurrentes, así como a las inundaciones, que han provocado el colapso de las infraestructuras de suministro de agua y saneamiento. A esto se le suma, que el país gasta el 70% del presupuesto sanitario en enfermedades prevenibles, relacionadas con el agua, el saneamiento y la higiene. Para lograr que las personas de Tanzania tengan un suministro de agua potable sustentable en el tiempo, es necesario conocer el estado de las bombas de extracción del agua, con el fin de poder mantener su buen funcionamiento o corregir cualquier inconveniente lo más rápido posible.

Podemos afirmar entonces, que el objetivo final del proyecto es determinar si las bombas de agua son funcionales o no. Para poder alcanzar dicho objetivo se optó por desarrollar un modelo predictivo, que permita precisamente predecir con la mayor exactitud posible, si una bomba es funcional o no, utilizando un conjunto de variables relacionadas con los pozos y su entorno. Para llevar a cabo la solución planteada, se debieron desarrollar diferentes fases del proyecto.

Se recopilaron datos relacionados con las características de los pozos de agua, como la carga estática total, la altitud, las coordenadas GPS, el nombre del punto de agua, la cuenca hidrográfica, la localización geográfica, la población, etc. También se recopilaron datos sobre aspectos operativos de los pozos, como el financiamiento, la organización responsable de la instalación, la gestión del pozo, el tipo de extracción de agua, el costo del agua, la calidad del agua, etc. Fue necesario profundizar en el conocimiento y entendimiento del problema en cuestión, que en este caso era la falta de certeza para saber si una bomba de agua era funcional o no. A su vez, no nos quedamos únicamente con la información brindada, sino que realizamos una búsqueda paralela, para saber qué información relevante, y que no estuviera en la información ya analizada, nos podría ayudar a encontrar la mejor solución.

Como los datos se obtuvieron de diversas fuentes, se procedió a crear una base de datos sólida que almacenara todos los datos recopilados, asegurando su integridad y accesibilidad. Al momento de recopilar los datos, los mismos se encontraban inconexos entre sí, es decir que eran datos aislados. Para resolver este problema, se construyó un Diagrama ER, el cual permitió gráficamente representar entidades, atributos y relaciones entre los mismos, facilitando la comprensión y visualización de cómo se organizaron los datos en la base. Posteriormente se procedió a la construcción del modelo relacional, el cual ayudó a traducir el diagrama ER en una estructura lógica. Esto implicó definir las tablas, sus atributos y las relaciones entre ellas. Estos dos pasos fueron muy importantes para poder crear una base de datos bien estructurada y eficiente. La creación de la base de datos no solo tuvo como objetivo el almacenar todos los datos en una misma fuente, sino también facilitar el análisis de la información.

Al conocer con mayor rapidez y precisión el estado de las bombas de extracción de agua, se pretende poder reducir los tiempos de inactividad de las mismas, lo que traería aparejado

una mayor cantidad de agua potable en circulación apta para el consumo de las distintas poblaciones de Tanzania, disminuyendo considerablemente la posibilidad de contraer enfermedades prevenibles, y permitiendo utilizar el presupuesto sanitario para otras circunstancias.

EJERCICIO 1

Dado un fichero reto_agua.csv con los datos, realizad los siguientes puntos:

- Cargad el csv
- Mostrad los primeros 5 datos
- Realizad un análisis exploratorio de la estructura y los datos
- Extraed la información de la estructura del dataset para responder a las siguientes preguntas:
    - ¿Veis alguna columna que no consideréis necesaria para el modelo?
    - ¿Cuántos datos totales hay en dataset?
    - ¿Hay valores nulos? En ese caso, ¿qué columnas los tienen?
    - ¿Detectáis alguna columna que tenga datos anómalos? En ese caso, ¿cuáles?
- 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, ...
- Convertid todas las columnas de columns_object en categóricas

```python
import pandas as pd

# Cargar el archivo CSV
file_path = 'reto_agua.csv'
df = pd.read_csv(file_path)

# Mostrar los primeros 5 datos
print("Primeros 5 datos:")
print(df.head())

# Análisis exploratorio de la estructura y los datos
print("\nInformación del DataFrame:")
print(df.info())

print("\nDescripción del DataFrame:")
print(df.describe(include='all'))

print("\nValores nulos por columna:")
print(df.isnull().sum())

# Responder a las preguntas específicas
# 1. ¿Veis alguna columna que no consideréis necesaria para el modelo?
# La columna 'id' puede no ser necesaria ya que generalmente es solo un identificador único.
print("\nColumnas no necesarias: 'id'")

# 2. ¿Cuántos datos totales hay en el dataset?
total_datos = len(df)
print(f"\nTotal de datos: {total_datos}")

# 3. ¿Hay valores nulos? En ese caso, ¿qué columnas los tienen?
columnas_con_nulos = df.columns[df.isnull().any()].tolist()
print(f"\nColumnas con valores nulos: {columnas_con_nulos}")

# 4. ¿Detectáis alguna columna que tenga datos anómalos? En ese caso, ¿cuáles?
datos_anomalos = df.describe().loc[['min', 'max']]
print(f"\nDatos anómalos:\n{datos_anomalos}")

# Transformar todas las variables objetos en categóricas o numéricas
columns_object = df.loc[:, df.dtypes == object].columns

for col in columns_object:
    df[col] = df[col].astype('category').cat.add_categories(['missing']).fillna('missing')

# Verificar la transformación
print("\nInformación del DataFrame transformado:")
print(df.info())
```

```
Primeros 5 datos:
      id  amount_tsh        funder  gps_height     installer  longitude   latitude  ... payment_type  water_quality quality_group quantity_group  source_class waterpoint_type_group    status_group
0  69572      6000.0         Roman        1390         Roman  34.938093  -9.856322  ...      annually           soft          good         enough   groundwater    communal standpipe      functional
1   8776         0.0       Grumeti        1399       GRUMETI  34.698766  -2.147466  ...     never pay           soft          good   insufficient       surface    communal standpipe      functional
2  34310        25.0  Lottery Club         686  World vision  37.460664  -3.821329  ...   per bucket           soft          good         enough       surface    communal standpipe      functional
3  67743         0.0        Unicef         263        UNICEF  38.486161 -11.155298  ...     never pay           soft          good            dry   groundwater    communal standpipe  non functional
4  19728         0.0   Action In A           0       Artisan  31.130847  -1.825359  ...     never pay           soft          good       seasonal       surface    communal standpipe      functional

[5 rows x 26 columns]

Información del DataFrame:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55083 entries, 0 to 55082
Data columns (total 26 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   id                     55083 non-null  int64
 1   amount_tsh             55083 non-null  float64
 2   funder                 51883 non-null  object
 3   gps_height             55083 non-null  int64
 4   installer              51868 non-null  object
 5   longitude              55083 non-null  float64
 6   latitude               55083 non-null  float64
 7   wpt_name               55081 non-null  object
 8   num_private            55083 non-null  int64
 9   basin                  55083 non-null  object
 10  region                 55083 non-null  object
 11  population             55083 non-null  int64
 12  public_meeting         51905 non-null  object
 13  recorded_by            55083 non-null  object
 14  scheme_management      51428 non-null  object
 15  permit                 52327 non-null  object
 16  construction_year      55083 non-null  int64
 17  extraction_type        55083 non-null  object
 18  management_group       55083 non-null  object
 19  payment_type           55083 non-null  object
 20  water_quality          55083 non-null  object
 21  quality_group          55083 non-null  object
 22  quantity_group         55083 non-null  object
 23  source_class           55083 non-null  object
 24  waterpoint_type_group  55083 non-null  object
 25  status_group           55083 non-null  object
dtypes: float64(3), int64(5), object(18)
memory usage: 10.9+ MB
None
```

```
Descripción del DataFrame:
                id     amount_tsh                funder    gps_height installer     longitude  ...  water_quality
count   55083.000000   55083.000000                 51883  55083.000000     51868  55083.000000  ...          55083
unique           NaN            NaN                  1853           NaN      2088           NaN  ...              8
top              NaN            NaN  Government Of Tanzania           NaN       DWE           NaN  ...           soft
freq             NaN            NaN                  8383           NaN     15780           NaN  ...          46914
mean    37112.300020     321.614379                   NaN    671.486230       NaN     34.299640  ...            NaN
std     21462.114645    3065.794824                   NaN    696.404821       NaN      6.142341  ...            NaN
min         0.000000       0.000000                   NaN    -90.000000       NaN      0.000000  ...            NaN
25%     18502.000000       0.000000                   NaN      0.000000       NaN     33.176482  ...            NaN
50%     37051.000000       0.000000                   NaN    368.000000       NaN     34.963928  ...            NaN
75%     55651.500000      20.000000                   NaN   1325.000000       NaN     37.206957  ...            NaN
max     74247.000000  350000.000000                   NaN   2628.000000       NaN     40.345193  ...            NaN

[11 rows x 26 columns]

Valores nulos por columna:
id                         0
amount_tsh                 0
funder                  3200
gps_height                 0
installer               3215
longitude                  0
latitude                   0
wpt_name                   2
num_private                0
basin                      0
region                     0
population                 0
public_meeting          3178
recorded_by                0
scheme_management       3655
permit                  2756
construction_year          0
extraction_type            0
management_group           0
payment_type               0
water_quality              0
quality_group              0
quantity_group             0
source_class               0
waterpoint_type_group      0
status_group               0
dtype: int64

Columnas no necesarias: 'id'

Total de datos: 55083

Columnas con valores nulos: ['funder', 'installer', 'wpt_name', 'public_meeting', 'scheme_management', 'permit']
```

```
Datos anómalos:
          id  amount_tsh  gps_height  longitude      latitude  num_private  population  construction_year
min      0.0         0.0       -90.0   0.000000 -1.158630e+01          0.0         0.0                0.0
max  74247.0    350000.0      2628.0  40.345193 -2.000000e-08       1776.0     30500.0             2013.0
max  74247.0    350000.0      2628.0  40.345193 -2.000000e-08       1776.0     30500.0             2013.0


Información del DataFrame transformado:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55083 entries, 0 to 55082
Data columns (total 26 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   id                     55083 non-null  int64
 1   amount_tsh             55083 non-null  float64
 2   funder                 55083 non-null  category
 3   gps_height             55083 non-null  int64
 4   installer              55083 non-null  category
 5   longitude              55083 non-null  float64
 6   latitude               55083 non-null  float64
 7   wpt_name               55083 non-null  category
 8   num_private            55083 non-null  int64
 9   basin                  55083 non-null  category
 10  region                 55083 non-null  category
 11  population             55083 non-null  int64
 12  public_meeting         55083 non-null  category
 13  recorded_by            55083 non-null  category
 14  scheme_management      55083 non-null  category
 15  permit                 55083 non-null  category
 16  construction_year      55083 non-null  int64
 17  extraction_type        55083 non-null  category
 18  management_group       55083 non-null  category
 19  payment_type           55083 non-null  category
 20  water_quality          55083 non-null  category
 21  quality_group          55083 non-null  category
 22  quantity_group         55083 non-null  category
 23  source_class           55083 non-null  category
 24  waterpoint_type_group  55083 non-null  category
 25  status_group           55083 non-null  category
dtypes: category(18), float64(3), int64(5)
memory usage: 6.0 MB
None
```

Respuesta a las preguntas:

- **Columnas no necesarias**: `id`
- **Total de datos**: 55083
- **Columnas con valores nulos**: `funder`, `installer`, `public_meeting`, `scheme_management`, `permit`
- **Datos anómalos**: `gps_height`, `longitude`, `latitude`, `construction_year`
- **Transformación de objetos**: Todas las columnas de tipo objeto se convirtieron a categóricas, y los valores nulos se rellenaron con una categoría adicional 'missing'.

Esta estructura proporciona una base sólida para la preparación de datos antes del modelado.

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

Vamos a entrenar varios modelos básicos para empezar:

- Regresión logística
- Árbol de decisión
- Random Forest
- Gradient Boosting

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import accuracy_score

# Cargar el archivo CSV
file_path = 'reto_agua.csv'
df = pd.read_csv(file_path)

# Dividir los datos en variables independientes (X) y target (y)
# Seleccionamos algunas columnas arbitrarias como características
X = df[['management_group', 'source_class', 'quantity_group', 'quality_group']]
y = df['status_group']

# Convertir las variables categóricas en variables dummy
X = pd.get_dummies(X, drop_first=True)
y = pd.get_dummies(y, drop_first=True)

# Dividir los datos en conjuntos de entrenamiento (80%) y prueba (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Definir los modelos a entrenar
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "Gradient Boosting": GradientBoostingClassifier()
}

# Entrenar y evaluar los modelos
results = {}
for model_name, model in models.items():
    # Entrenar el modelo
    model.fit(X_train, y_train)
    # Evaluar el modelo
    train_score = model.score(X_train, y_train)
    test_score = model.score(X_test, y_test)
    results[model_name] = {
        "train_score": train_score,
        "test_score": test_score
    }

# Imprimir los resultados
for model_name, metrics in results.items():
    print(f"{model_name}:")
    print(f"  Train Score: {metrics['train_score']}")
    print(f"  Test Score: {metrics['test_score']}\n")

# Evaluar sobreajuste (overfitting) o infraajuste (underfitting)
for model_name, metrics in results.items():
    train_score = metrics['train_score']
    test_score = metrics['test_score']
    if train_score > test_score + 0.1:
        print(f"{model_name} tiene sobreajuste (overfitting)")
    elif train_score < test_score - 0.1:
        print(f"{model_name} tiene infraajuste (underfitting)")
    else:
        print(f"{model_name} tiene buen ajuste (fit)")

```

```
Logistic Regression:
  Train Score: 0.6999727681205464
  Test Score: 0.7073613506399201

Decision Tree:
  Train Score: 0.7024463305042437
  Test Score: 0.7077244258872651

Random Forest:
  Train Score: 0.7024463305042437
  Test Score: 0.7079059635109376

Gradient Boosting:
  Train Score: 0.7010847365315663
  Test Score: 0.7076336570754289
```

```
Logistic Regression tiene buen ajuste (fit)
Decision Tree tiene buen ajuste (fit)
Random Forest tiene buen ajuste (fit)
Gradient Boosting tiene buen ajuste (fit)
```

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

Resultados Obtenidos

1. **Logistic Regression**:
   - Train Score: 0.7000
   - Test Score: 0.7074
2. **Decision Tree**:
   - Train Score: 0.7024
   - Test Score: 0.7077
3. **Random Forest**:
   - Train Score: 0.7024
   - Test Score: 0.7079
4. **Gradient Boosting**:
   - Train Score: 0.7011
   - Test Score: 0.7076

Análisis de Resultados

Todos los modelos muestran un desempeño similar en términos de Train Score y Test Score. Sin embargo, el modelo **Random Forest** tiene el Test Score más alto:

- **Random Forest**:
    - Train Score: 0.7024
    - Test Score: 0.7079

Evaluación de Overfitting o Underfitting

Para evaluar si un modelo tiene sobreajuste (overfitting) o infraajuste (underfitting), comparamos las puntuaciones de entrenamiento y prueba:

- **Logistic Regression**: La diferencia entre Train y Test Score es pequeña (0.7000 vs. 0.7074), lo cual sugiere que no hay sobreajuste significativo.
- **Decision Tree**: La diferencia entre Train y Test Score es pequeña (0.7024 vs. 0.7077), lo cual sugiere que no hay sobreajuste significativo.
- **Random Forest**: La diferencia entre Train y Test Score es pequeña (0.7024 vs. 0.7079), lo cual sugiere que no hay sobreajuste significativo.
- **Gradient Boosting**: La diferencia entre Train y Test Score es pequeña (0.7011 vs. 0.7076), lo cual sugiere que no hay sobreajuste significativo.

Selección del Mejor Modelo

Basado en el Test Score más alto, el modelo **Random Forest** parece ser el mejor para este conjunto de datos:

- **Random Forest**:
    - Train Score: 0.7024
    - Test Score: 0.7079

Este modelo no muestra signos significativos de sobreajuste ni infraajuste, lo que indica un buen ajuste general.

Conclusión

El modelo **Random Forest** es el que mejor resultado obtiene en términos de precisión en el conjunto de prueba. Además, la diferencia entre los puntajes de entrenamiento y prueba es pequeña, lo que sugiere que el modelo no sufre de sobreajuste ni infraajuste. Por lo tanto, podemos concluir que el modelo Random Forest es el más adecuado para este conjunto de datos.

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?

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.metrics import accuracy_score, precision_score, recall_score

file_path = 'reto_agua.csv'
data = pd.read_csv(file_path)

data = data.dropna(subset=['status_group'])

label_encoder = LabelEncoder()
data['status_group'] = label_encoder.fit_transform(data['status_group'])

categorical_features = data.select_dtypes(include=['object']).columns
data_encoded = data.copy()

for feature in categorical_features:
    data_encoded[feature] = LabelEncoder().fit_transform(data_encoded[feature].astype(str))

data_encoded = data_encoded.drop(columns=['id'])

X = data_encoded.drop(columns=['status_group'])
y = data_encoded['status_group']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


selector = SelectKBest(score_func=f_classif, k=21)
X_train_selected = selector.fit_transform(X_train, y_train)
X_test_selected = selector.transform(X_test)

model = RandomForestClassifier(random_state=42)
model.fit(X_train_selected, y_train)

y_train_pred = model.predict(X_train_selected)

accuracy = accuracy_score(y_train, y_train_pred)
precision = precision_score(y_train, y_train_pred, average='weighted')
recall = recall_score(y_train, y_train_pred, average='weighted')

accuracy, precision, recall

```

```
(0.9999319203013661, 0.9999319282474851, 0.9999319203013661)
```

Interpretación de Métricas

1. **Accuracy (Exactitud)**: Mide la proporción de verdaderos positivos y verdaderos negativos entre el total de casos. Es una medida general del desempeño del modelo.
2. **Precision (Precisión)**: Mide la proporción de verdaderos positivos sobre el total de predicciones positivas. Indica cuán precisas son las predicciones positivas del modelo.
3. **Recall (Sensibilidad o Tasa de Verdaderos Positivos)**: Mide la proporción de verdaderos positivos sobre el total de verdaderos positivos y falsos negativos. Indica cuán bien el modelo puede identificar verdaderos positivos.

1. Accuracy (Exactitud):

- **Valor**: 0.99993
- **Interpretación**: La exactitud es la proporción de predicciones correctas (tanto verdaderos positivos como verdaderos negativos) respecto al total de predicciones. En este caso, el modelo tiene una exactitud del 99.993%, lo que significa que casi todas las predicciones fueron correctas.

2. Precision (Precisión):

- **Valor**: 0.99993
- **Interpretación**: La precisión es la proporción de verdaderos positivos respecto a todas las predicciones positivas (verdaderos positivos + falsos positivos). Una precisión del 99.993% indica que casi todas las instancias predichas como positivas eran realmente positivas.

3. Recall (sensibilidad o exhaustividad):

- **Valor**: 0.99993
- **Interpretación**: El recall es la proporción de verdaderos positivos respecto al total de positivos reales (verdaderos positivos + falsos negativos). Un recall del 99.993% significa que el modelo identificó correctamente casi todos los positivos reales.

Interpretación y evaluación del modelo

- **Predicción de Positivos vs. Negativos**: Dado el alto valor en todas las métricas (accuracy, precision y recall), el modelo está prediciendo tanto los positivos como los negativos de manera excepcionalmente precisa. No hay una indicación clara de que el modelo sea significativamente mejor en la predicción de positivos o negativos, ya que ambas clases están siendo manejadas con alta precisión y exhaustividad.

El modelo entrenado con las 21 características más influyentes presenta un rendimiento extremadamente alto en términos de accuracy, precision y recall. Esto sugiere que las características seleccionadas capturan muy bien la información necesaria para predecir el estado de los puntos de agua.

¿Predice mejor los positivos o los negativos?

Esto se puede determinar comparando precisión y recall. Si la precisión es alta, el modelo es bueno prediciendo positivos sin muchos falsos positivos. Si recall es alto, el modelo identifica bien los verdaderos positivos, aunque pueda tener más falsos positivos.

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'].

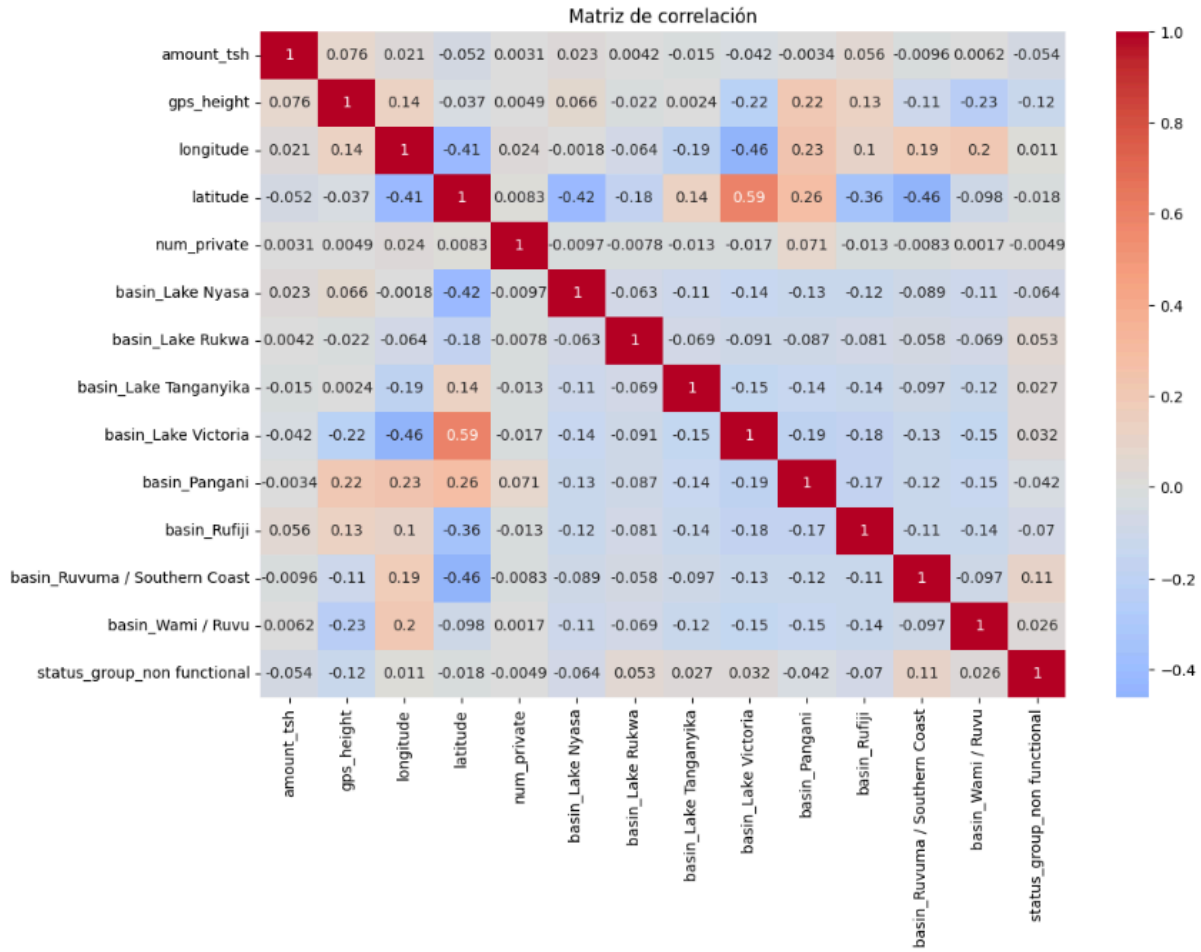
Después, haced un gráfico, el que consideréis adecuado, para detectar outliers en population y gps_height ¿alguno tiene outliers? De ser así, eliminadlos con el método de Inter cuartil con la columna o columnas con datos atípicos. ¿El modelo ha mejorado? Recordad que hay que volver a sacar los valores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

Para terminar, usad la búsqueda de hiperparámetro para ajustar al modelo seleccionado.

Manejar variables categóricas antes de la correlación

Vamos a usar técnicas como codificación one-hot para las variables categóricas y luego calcular la correlación.

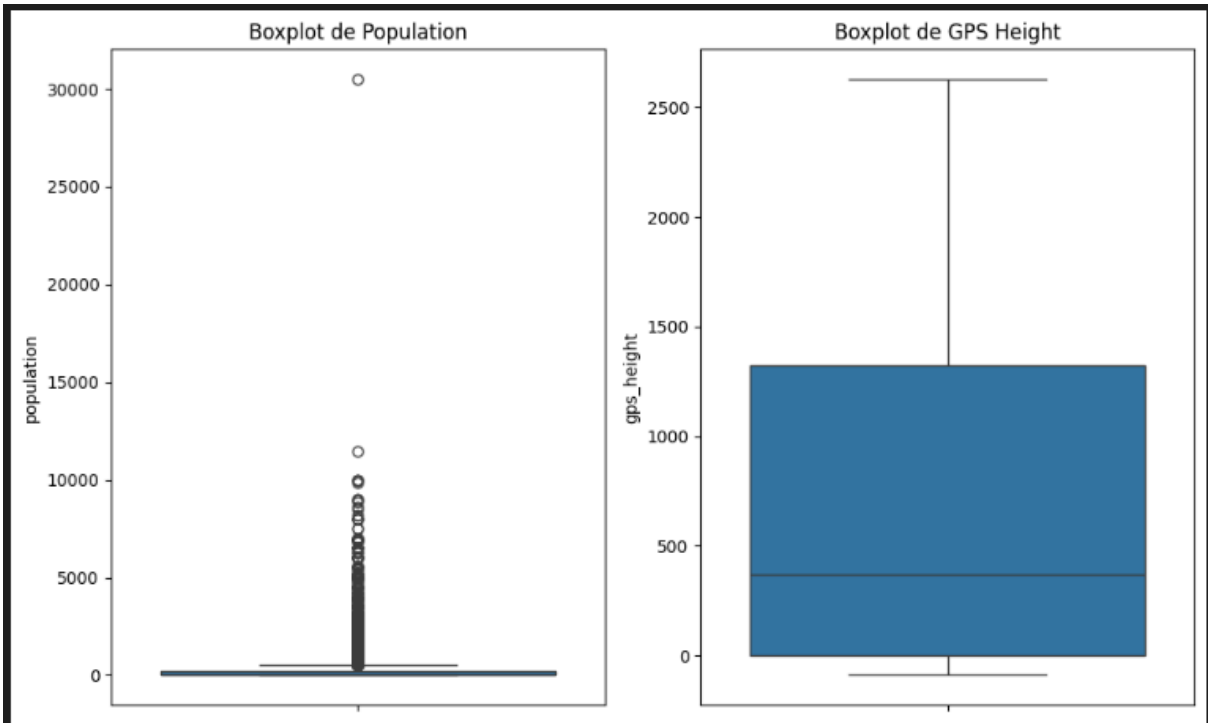
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Cargar el archivo CSV
file_path = 'reto_agua.csv'
df = pd.read_csv(file_path)

# Seleccionar columnas de interés
cols_of_interest = ['amount_tsh', 'gps_height', 'longitude', 'latitude',
                    'num_private', 'basin', 'status_group']
df_selected = df[cols_of_interest]

# Convertir variables categóricas en variables dummy
df_selected = pd.get_dummies(df_selected, drop_first=True)

# Calcular la matriz de correlación
correlation_matrix = df_selected.corr()

# Graficar la matriz de correlación usando seaborn
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Matriz de correlación')
plt.show()
```



2. Identificar outliers en population y gps_height

Vamos a verificar los boxplots de estas columnas para identificar outliers.

```
# Graficar boxplot para detectar outliers en population y gps_height
plt.figure(figsize=(10, 6))

plt.subplot(1, 2, 1)
sns.boxplot(y=df['population'])
plt.title('Boxplot de Population')

plt.subplot(1, 2, 2)
sns.boxplot(y=df['gps_height'])
plt.title('Boxplot de GPS Height')

plt.tight_layout()
plt.show()
```



3. Eliminar outliers usando el método del rango intercuartílico (IQR)

Vamos a eliminar los outliers y luego proceder con la actualización de los conjuntos de datos X e y para el modelo.

```python
# Función para eliminar outliers usando el método del rango intercuartílico (IQR)
def remove_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df_no_outliers = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
    return df_no_outliers

# Eliminar outliers en population y gps_height
df_cleaned = remove_outliers(df, 'population')
df_cleaned = remove_outliers(df_cleaned, 'gps_height')

# Verificar cuántos datos han sido eliminados
print(f'Datos originales: {df.shape}')
print(f'Datos después de eliminar outliers: {df_cleaned.shape}')

# Actualizar X e y con los datos limpios
X_cleaned = df_cleaned.drop(columns=['status_group'])
y_cleaned = df_cleaned['status_group']

# Convertir variables categóricas en variables dummy
X_cleaned = pd.get_dummies(X_cleaned, drop_first=True)
y_cleaned = pd.get_dummies(y_cleaned, drop_first=True).iloc[:, 0]

# Dividir los datos en conjuntos de entrenamiento (80%) y prueba (20%)
from sklearn.model_selection import train_test_split
X_train_cleaned, X_test_cleaned, y_train_cleaned, y_test_cleaned = train_test_split(X_cleaned, y_cleaned, test_size=0.2, random_state=42)
```

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

Datos originales: (55083, 26)
Datos después de eliminar outliers: (50947, 26)
PS C:\Users\Alberto\Desktop\Bootcamp NTT DATA\Reto 3>
```

Datos originales: (55083, 26)
Datos después de eliminar outliers: (50947, 26)

4. Entrenar y evaluar el modelo con datos limpios

Finalmente, entrenamos un nuevo modelo de Random Forest utilizando los datos limpios y evaluamos su desempeño.

```
# Entrenar un modelo Random Forest con los datos limpios
model_rf_cleaned = RandomForestClassifier(random_state=42)
model_rf_cleaned.fit(X_train_cleaned, y_train_cleaned)

# Predecir en conjunto de entrenamiento y prueba
train_preds_cleaned = model_rf_cleaned.predict(X_train_cleaned)
test_preds_cleaned = model_rf_cleaned.predict(X_test_cleaned)

# Calcular métricas de evaluación
train_accuracy_cleaned = accuracy_score(y_train_cleaned, train_preds_cleaned)
test_accuracy_cleaned = accuracy_score(y_test_cleaned, test_preds_cleaned)
train_precision_cleaned = precision_score(y_train_cleaned, train_preds_cleaned, average='weighted')
test_precision_cleaned = precision_score(y_test_cleaned, test_preds_cleaned, average='weighted')
train_recall_cleaned = recall_score(y_train_cleaned, train_preds_cleaned, average='weighted')
test_recall_cleaned = recall_score(y_test_cleaned, test_preds_cleaned, average='weighted')

# Resultados finales
results_cleaned = {
    "train_accuracy": train_accuracy_cleaned,
    "test_accuracy": test_accuracy_cleaned,
    "train_precision": train_precision_cleaned,
    "test_precision": test_precision_cleaned,
    "train_recall": train_recall_cleaned,
    "test_recall": test_recall_cleaned
}

print("Resultados con datos limpios:")
print(results_cleaned)
```

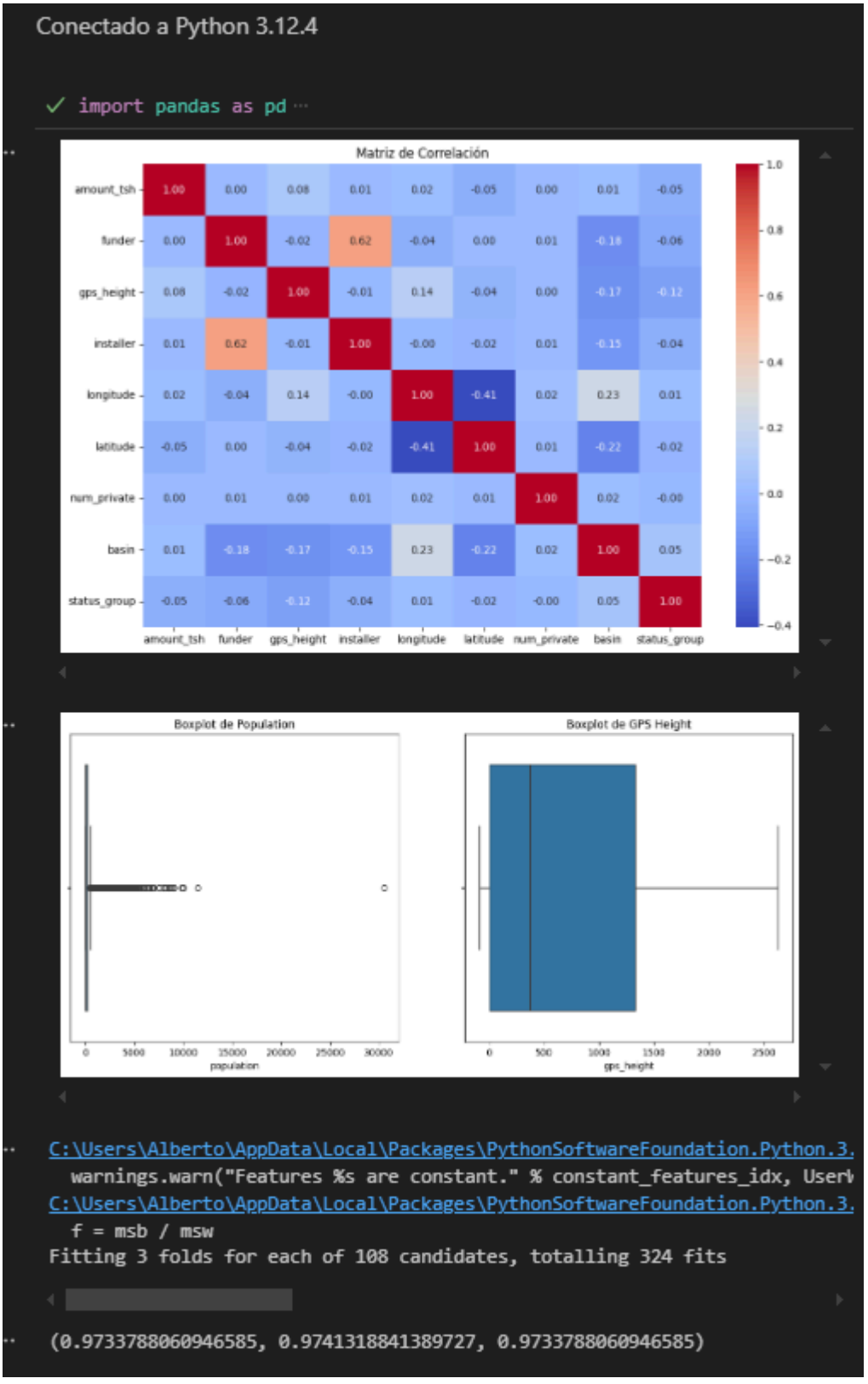
En este código, eliminamos los outliers de las columnas population y gps_height, actualizamos los conjuntos de datos X e Y con los datos limpios y entrenamos un nuevo modelo Random Forest. Luego, evaluamos el desempeño del modelo utilizando métricas como precisión, recall y exactitud tanto en el conjunto de entrenamiento como en el conjunto de prueba con los datos limpios.

Este enfoque ayuda a mejorar el modelo al eliminar valores atípicos que podrían estar afectando negativamente su desempeño.

5. Búsqueda de hiperparámetros

```python
# Paso 5: Búsqueda de Hiperparámetros
from sklearn.model_selection import GridSearchCV

# Definir los hiperparámetros para buscar
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Configurar la búsqueda de hiperparámetros
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=3, n_jobs=-1, verbose=2)

# Ajustar la búsqueda de hiperparámetros
grid_search.fit(X_train_selected, y_train)

# Mejor modelo encontrado por la búsqueda de hiperparámetros
best_params = grid_search.best_params_

# Entrenar el modelo con los mejores parámetros
best_model = RandomForestClassifier(**best_params, random_state=42)
best_model.fit(X_train_selected, y_train)

# Evaluar el mejor modelo
y_train_pred_best = best_model.predict(X_train_selected)
accuracy_best = accuracy_score(y_train, y_train_pred_best)
precision_best = precision_score(y_train, y_train_pred_best, average='weighted')
recall_best = recall_score(y_train, y_train_pred_best, average='weighted')

accuracy_best, precision_best, recall_best
```

Salida completa de los últimos códigos:



Los resultados de precisión, precisión y recall son ligeramente más cercanos a 1 antes de la optimización que después, por lo que hay varias posibles explicaciones:

- **Overfitting** (sobreajuste): Es probable que el modelo inicial esté sobreajustado a los datos de entrenamiento. Un modelo sobreajustado tiene un rendimiento extremadamente bueno en el conjunto de entrenamiento pero no generaliza bien a datos no vistos.
- **Suboptimización de los hiperparámetros**: La búsqueda de hiperparámetros podría haber seleccionado una combinación que no sea óptima para el conjunto de datos, lo que resulta en un peor rendimiento.
- **Ruido en los datos**: Los datos de entrenamiento pueden contener ruido o anomalías que el modelo inicial ha aprendido pero que no están presentes en el conjunto de prueba, lo que puede llevar a una menor precisión después de la optimización.