DengAI predicción de la propagación de enfermedades



Adrián Yared Armas de la Nuez

**Contenido**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

[**1. Objetivo 2**](#_2c35568x6wg8)

[**2. Actividad 3**](#_34l6p03hepl7)

[**2.1 Imports 3**](#_t8cwbo6n4466)

[**2.4.5 Comando 3**](#_aig92oe7k2m0)

[**2.1 Dataset 4**](#_bab8rs6cyzry)

[**2.2.2 Descarga del dataset desde github 4**](#_zbwta4ttp5o)

[**2.2.3 Ejecución 4**](#_7iok6tlp8r23)

[**2.2.2 Descomprimir 4**](#_6h8oo1dg2e9u)

[**2.2.3 Verificar la descarga y descompresión 5**](#_10cwnrfo93hw)

[**2.2.4 Ejecución 5**](#_pujyz3hrpd3s)

[**2.3 Limpieza de datos 5**](#_j2tpsyifem24)

[**2.3.1 Carga de dengue\_features\_test.csv 5**](#_8wm6kuu3bffy)

[**2.3.2 Fusionar funciones y etiquetas 5**](#_9krj5nfrxlo7)

[**2.4 Preprocesamiento 5**](#_jc6y376brbj5)

[**2.4.1 Codificar ciudad 5**](#_glyygtps60q)

[**2.4.2 Extraer mes del conjunto week\_start\_date 6**](#_gd13u4hfhf5r)

[**2.4.3 características cíclicas para el mes y la semana del año 6**](#_n7tdki4m4lr7)

[**2.4.4 Eliminado de columnas innecesarias 6**](#_rnk1gd2onbxm)

[**2.4.5 Rellenado de valores (mediana) 6**](#_agqwzh1dhts5)

[**2.4.6 División de train x e y 7**](#_gvq4y1gz1sj1)

[**2.4.7 Set de validación 7**](#_n6ew4tr1a0gi)

[**2.4.8 Selección de funciones 7**](#_rfnuzkn92ezg)

[**2.4.8.1 Usando un metodo no gráfico 7**](#_cq1jgdyq6t9y)

[**2.4.8.1.1 Código 7**](#_hu4t8lq6abms)

[**2.4.8.1.2 Ejecución 7**](#_j5nu81gudvww)

[**2.4.8.2 Usando un metodo gráfico 8**](#_lut8dgscai7c)

[**2.4.8.2.1 Código 8**](#_xhdbjyd5x3a7)

[**2.4.8.2.2 Ejecución 8**](#_eugyxma16uo3)

[**2.4.8.3 Matriz de correlación 8**](#_q8y3c1n6lnxv)

[**2.4.8.3.1 Código 8**](#_7ct1khxy22ao)

[**2.4.8.3.2 Ejecución 9**](#_wyllutql4lj1)

[**2.5 Elegir y entrenar el modelo 9**](#_68g1043p09fw)

[**2.5.1 Comparación de modelos 9**](#_rt1md15j4gai)

[**2.5.1.1 Código 9**](#_w3pmd4rlmtg7)

[**2.5.2 Validación cruzada y evaluación del modelo 10**](#_gg2bfgutcppz)

[**2.5.2.1 Código 10**](#_3b1wb44l7xmq)

[**2.5.2.2 Ejecución 10**](#_csrjw1eyn8w1)

[**2.5.3 Comparación gráfica 10**](#_xf933indac56)

[**2.5.3.1 Código 10**](#_qskvrorim87y)

[**2.5.3.2 Ejecución 11**](#_rvnqt8b1ikkw)

[**2.5.4 Entrenamiento del modelo 11**](#_1srseywqy2y7)

[**2.5.4.1 Ejecución 11**](#_m5ze39lb2g9f)

[**2.5.4.2 Ejecución 11**](#_4v7b6qbow84i)

[**2.6 Evaluar el modelo (MAE) 12**](#_akysghcnqkxi)

[**2.6.1 No gráficamente 12**](#_x1j8ixluffy4)

[**2.6.1.1 Código 12**](#_zgc5e0f4o9i9)

[**2.6.1.2 Ejecución 12**](#_ugwlmlsw9rni)

[**2.6.1 Gráficamente 12**](#_p60ctdnk0ivn)

[**2.6.1.1 Código 12**](#_j4so7fvm9yfz)

[**2.6.1.2 Ejecución 13**](#_ejsf18da9klk)

[**2.7 Entrenamiento final 13**](#_ecnkmnofo1pl)

[**2.7.1 Fit 13**](#_2yag8amdyi2t)

[**2.7.1.1 Código 13**](#_cbvto2pxl8cy)

[**2.7.1.2 Ejecución 13**](#_jobp35owr97d)

[**2.7.2 Preparación de los datos de test 14**](#_owl9bu91xp0k)

[**2.7.1.1 Código 14**](#_3ie8z9qh04zl)

[**2.8 Descarga de csv 14**](#_wvef4jwd9p6o)

[**2.7.1 Formato correcto 14**](#_masb7f6yf5gu)

[**2.7.1.1 Código 14**](#_wj7jhidugsdp)

[**2.7.2 Añado la ciudad 14**](#_jnkyv8rf6483)

[**2.7.2.1 Código 14**](#_vt50abmq8ny3)

[**2.7.3 Orden 15**](#_xftp8n9iv7vr)

[**2.7.3.1 Código**](#_ybrkpn6t4p9) **15**

[**2.7.4 Guardar el csv 15**](#_y6mkpt4cwi4p)

[**2.7.4.1 Código 15**](#_dokkgly55qun)

[**2.7.5 Descarga del csv 15**](#_et345cbdy344)

[**2.7.5.1 Código 15**](#_rltjvkv249lp)

[**3. Resultado de la competición 15**](#_zbzm2fwkttzd)

[**4. Bibliografía 16**](#_fygye0lrbs9e)

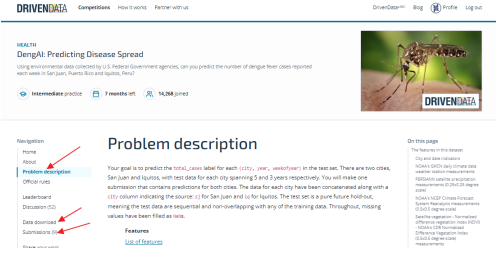
[**5. Github y Colab 16**](#_lmdngf8dlin3)

## **1. Objetivo**

El objeto de esta actividad es participar en la competición de ofrecida de la web de DrivenData denominada: DengAI: Predicting Disease Spread. Para ello accederemos a la siguiente web y nos crearemos un usuario:

**Título:** DengAI: Predicting Disease Spread

**Url**: <https://www.drivendata.org/competitions/44/dengai-predicting-disease-spread/>



La actividad consiste en subir a dicha web un fichero csv con la estimación que hayamos

obtenido al aplicar los modelos que consideres oportunos. Podrás observar que se pueden

realizar hasta un máximo de 3 subidas diarias y la propia web realizará una valoración de tu

solución. (La valoración se realiza utilizando el MAE como criterio de valoración de calidad)

Hay que realizar todo el proceso completo de importación del dataset, ajuste de

características, selección de características, entrenamiento y selección de un modelo para

entrenarlo, predicción y subida del fichero para que la web lo valores y clasifique en la

competición.

Consideraciones a tener en cuenta:

• Es necesario realizar pruebas con al menos tres modelos diferentes considerando que

han de utilizarse como mínimo dos modelos de árboles y otro modelo que consideres

oportuno

• Realizar pruebas de hiperparametrización con las dos técnicas explicadas: GridSearch y

Random Search.

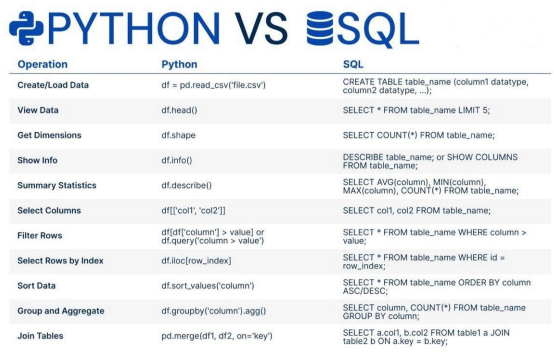
Una vez hayas realizado las diferentes pruebas, has de generar un documento pdf donde:

Expliques con detalle tu solución. Ha de contener capturas de tu posicionamiento en la

competición con la clasificación que has obtenido en la competición con los diferentes

SUBMITs realizados. Sería muy interesante que expliques las mejoras o no, que has

obtenido con las diferentes pruebas.



**Fuente**: [https://www.linkedin.com/feed/update/urn:li:activity:7290306350243274752/?utm\_s ource=share&utm\_medium=member\_desktop](https://www.linkedin.com/feed/update/urn:li:activity:7290306350243274752/?utm_s)

Tïtulo: Modelos de machine learning: Guía básica para principiantes

**Url:**<https://planetachatbot.com/modelos-de-machine-learning-guia-basica-paraprincipiantes/>

## **2. Actividad**

### **2.1 Imports**

#### **2.4.5 Comando**

import pandas as pd

import zipfile

import os

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, cross\_val\_score, GridSearchCV

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import Ridge

from sklearn.metrics import mean\_absolute\_error

from google.colab import files

from sklearn.feature\_selection import SelectKBest, f\_regression

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

from sklearn.naive\_bayes import GaussianNB

from sklearn.neighbors import KNeighborsClassifier

### **2.1 Dataset**

#### **2.2.2 Descarga del dataset desde github**

# Download dataset from github

!wget -O DengAI\_dataset.zip "https://github.com/AdrianYArmas/IaBigData/raw/main/SNS/3%20%20-%20Algoritmos%20y%20herramientas%20para%20el%20aprendizaje%20supervisado%20/3.6%20DengAI%20predicci%C3%B3n%20de%20la%20propagaci%C3%B3n%20de%20enfermedades/Resources/DengAI%20dataset.zip"

#### **2.2.3 Ejecución**



#### **2.2.2 Descomprimir**

with zipfile.ZipFile("DengAI\_dataset.zip", 'r') as zip\_ref:

zip\_ref.extractall("DengAI\_dataset")

#### **2.2.3 Verificar la descarga y descompresión**

os.listdir("DengAI\_dataset")

#### **2.2.4 Ejecución**



### **2.3 Limpieza de datos**

#### **2.3.1 Carga de dengue\_features\_test.csv**

dengue\_features\_test.csv tiene como objetivo de este conjunto de datos es mostrar todos los datos recopilados, por lo que elegiré datos relevantes para hacer la predicción. (En este caso son columnas numéricas, no fechas ni cadenas)

# Data load

path = '/content/DengAI\_dataset/DengAI dataset/'

train\_features = pd.read\_csv(path+'dengue\_features\_train.csv')

train\_labels = pd.read\_csv(path+'dengue\_labels\_train.csv')

test\_features = pd.read\_csv(path+'dengue\_features\_test.csv')

#### **2.3.2 Fusionar funciones y etiquetas**

Fusioné etiquetas para garantizar que las funciones y las etiquetas correspondientes estén alineadas correctamente para el entrenamiento. Combina los datos en un único DataFrame para la entrada del modelo.

#### train = train\_features.merge(train\_labels, on=['city', 'year', 'weekofyear'])

### 

### **2.4 Preprocesamiento**

#### **2.4.1 Codificar ciudad**

Esta línea codifica la columna de la ciudad asignando los valores de cadena ('sj' y 'iq') a valores numéricos (0 y 1) para facilitar el procesamiento del modelo. Se aplica la misma codificación a los DataFrames de entrenamiento y de prueba.

# Encode city (sj -> 0, iq -> 1)

train['city'] = train['city'].map({'sj': 0, 'iq': 1})

test = test\_features.copy()

test['city'] = test['city'].map({'sj': 0, 'iq': 1})

#### **2.4.2 Extraer mes del conjunto week\_start\_date**

Agrega la información del mes extraída como una nueva columna en los DataFrames de tren y de prueba.

train['month'] = pd.to\_datetime(train['week\_start\_date']).dt.month

test['month'] = pd.to\_datetime(test['week\_start\_date']).dt.month

#### **2.4.3 características cíclicas para el mes y la semana del año**

Características cíclicas al convertir el mes y la semana del año en transformaciones de seno y coseno.

Estas nuevas columnas (mes\_sin, mes\_cos, semanadelaño\_sin, semanadelaño\_cos) se agregan a los DataFrames de entrenamiento y de prueba para capturar patrones periódicos

(Se usan transformaciones seno y coseno para representar meses y semanas de forma cíclica, añadiendo estas columnas a los DataFrames para capturar patrones temporales.)

# Create cyclical features for month and weekofyear

for df in [train, test]:

df['month\_sin'] = np.sin(2 \* np.pi \* df['month'] / 12)

df['month\_cos'] = np.cos(2 \* np.pi \* df['month'] / 12)

df['weekofyear\_sin'] = np.sin(2 \* np.pi \* df['weekofyear'] / 52)

df['weekofyear\_cos'] = np.cos(2 \* np.pi \* df['weekofyear'] / 52)

#### **2.4.4 Eliminado de columnas innecesarias**

# Drop unnecessary columns

columns\_to\_drop = ['week\_start\_date', 'year', 'month', 'weekofyear']

train = train.drop(columns=columns\_to\_drop)

test = test.drop(columns=columns\_to\_drop)

#### **2.4.5 Rellenado de valores (mediana)**

Rellena valores nulos con la mediana para no tener valores tan extremos.

# Fill missing values

for col in train.columns:

if train[col].isnull().any():

# Fill with median grouped by city

train[col] = train.groupby('city')[col].transform(lambda x: x.fillna(x.median()))

for col in test.columns:

if test[col].isnull().any():

test[col] = test.groupby('city')[col].transform(lambda x: x.fillna(x.median()))

#### **2.4.6 División de train x e y**

# Split into features and target

X = train.drop(columns=['total\_cases'])

y = train['total\_cases']

#### **2.4.7 Set de validación**

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

#### **2.4.8 Selección de funciones**

##### **2.4.8.1 Usando un metodo no gráfico**

###### **2.4.8.1.1 Código**

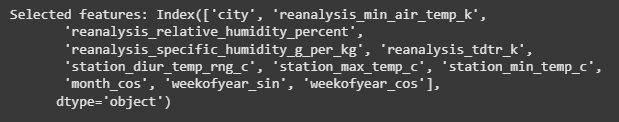
selector = SelectFromModel(XGBRegressor(objective='reg:absoluteerror', n\_estimators=1000, learning\_rate=0.05, random\_state=42))

selector.fit(X\_train, y\_train)

selected\_features = X\_train.columns[(selector.get\_support())]

print(f"Selected features: {selected\_features}")

###### **2.4.8.1.2 Ejecución**



##### **2.4.8.2 Usando un metodo gráfico**

###### **2.4.8.2.1 Código**

model = XGBRegressor(objective='reg:absoluteerror', n\_estimators=1000, learning\_rate=0.05, random\_state=42)

model.fit(X\_train, y\_train)

plt.figure(figsize=(10, 6))

plt.barh(X\_train.columns, model.feature\_importances\_)

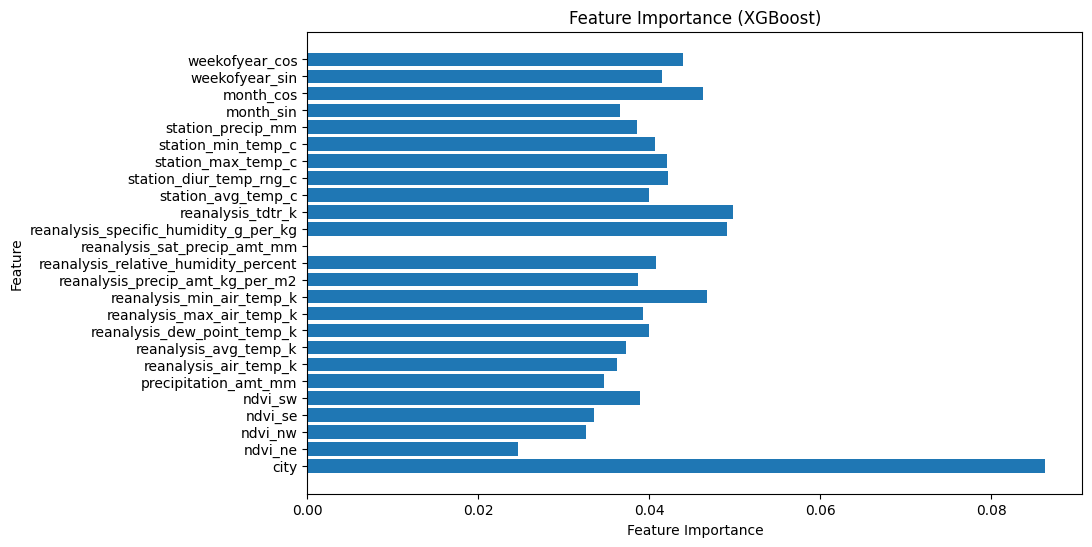
plt.xlabel('Feature Importance')

plt.ylabel('Feature')

plt.title('Feature Importance (XGBoost)')

plt.show()

###### **2.4.8.2.2 Ejecución**



##### **2.4.8.3 Matriz de correlación**

###### **2.4.8.3.1 Código**

correlation\_matrix = train.corr()

# Heat map correlation matrix

plt.figure(figsize=(12, 8))

plt.imshow(correlation\_matrix, cmap='coolwarm', interpolation='none')

plt.colorbar()

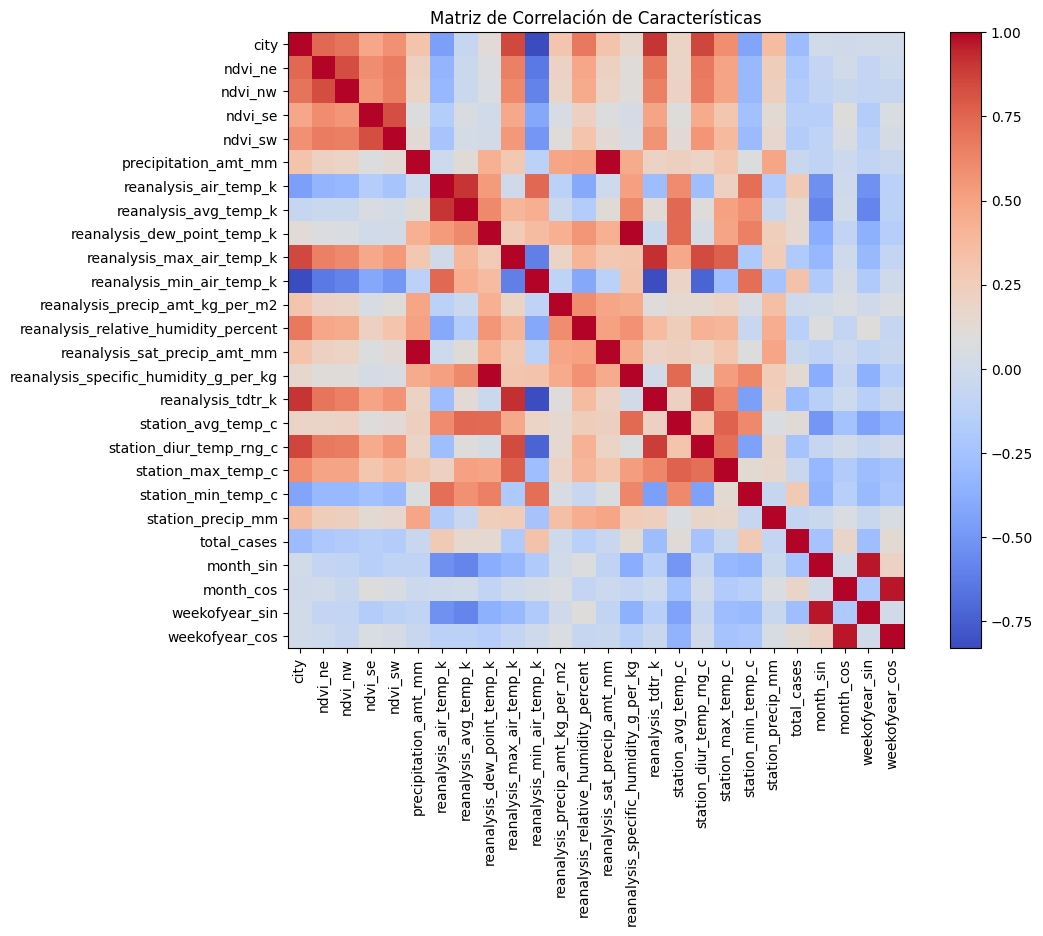
plt.xticks(np.arange(len(correlation\_matrix.columns)), correlation\_matrix.columns, rotation=90)

plt.yticks(np.arange(len(correlation\_matrix.columns)), correlation\_matrix.columns)

plt.title('Matriz de Correlación de Características')

plt.show()

###### **2.4.8.3.2 Ejecución**



### **2.5 Elegir y entrenar el modelo**

#### **2.5.1 Comparación de modelos**

##### **2.5.1.1 Código**

He añadigo XGBRegresor porque en pruebas anteriores daba mejores resultados que otros modelos tales como randomForest u otros de regresión lineal.

models = {

'Naive Bayes': GaussianNB(),

'KNN': KNeighborsRegressor(),

'XGBRegressor': XGBRegressor(objective='reg:absoluteerror', n\_estimators=1000, learning\_rate=0.05, random\_state=42)

}

#### **2.5.2 Validación cruzada y evaluación del modelo**

##### **2.5.2.1 Código**

mae\_scores = {}

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

cv\_scores = cross\_val\_score(model, X\_train[selected\_features], y\_train, cv=5, scoring='neg\_mean\_absolute\_error')

mae\_scores[model\_name] = -cv\_scores.mean()

print(f"{model\_name} - Cross-validated MAE: {-cv\_scores.mean()}")

##### **2.5.2.2 Ejecución**



#### **2.5.3 Comparación gráfica**

##### **2.5.3.1 Código**

# Graficar comparación de MAE de los modelos

plt.figure(figsize=(8, 6))

plt.bar(mae\_scores.keys(), mae\_scores.values(), color=['blue', 'green', 'red'])

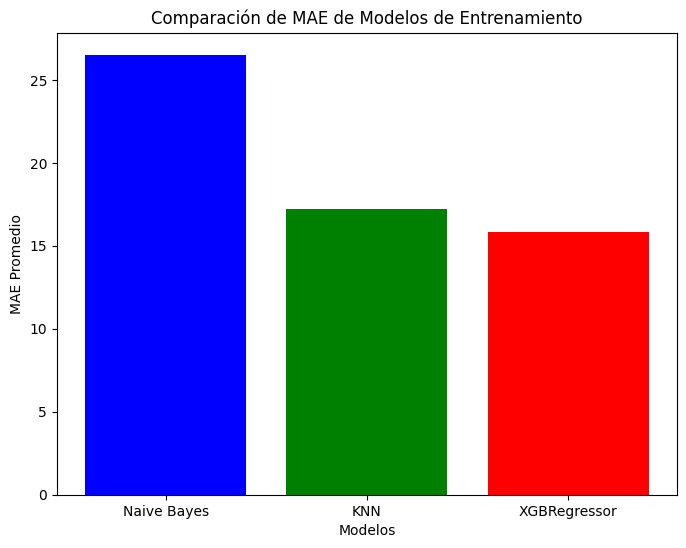
plt.xlabel('Modelos')

plt.ylabel('MAE Promedio')

plt.title('Comparación de MAE de Modelos de Entrenamiento')

plt.show()

##### **2.5.3.2 Ejecución**



#### **2.5.4 Entrenamiento del modelo**

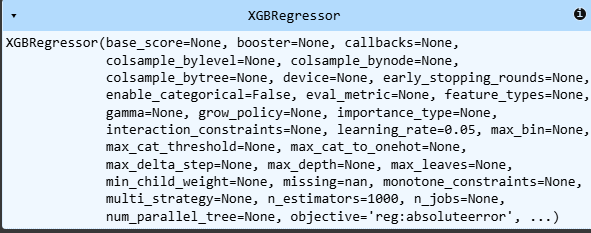
Entrena en mejor resultado de los modelos.

##### **2.5.4.1 Ejecución**

best\_model = XGBRegressor(objective='reg:absoluteerror', n\_estimators=1000, learning\_rate=0.05, random\_state=42)

best\_model.fit(X\_train, y\_train)

##### **2.5.4.2 Ejecución**



### **2.6 Evaluar el modelo (MAE)**

#### **2.6.1 No gráficamente**

##### **2.6.1.1 Código**

# Evaluate on validation set

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

mae = mean\_absolute\_error(y\_val, y\_pred)

print(f'MAE on Validation Set: {mae}')

##### **2.6.1.2 Ejecución**



#### **2.6.1 Gráficamente**

##### **2.6.1.1 Código**

# Visualization: Predicted vs Actual (Validation Set)

plt.figure(figsize=(10, 6))

plt.scatter(y\_val, y\_pred)

plt.plot([min(y\_val), max(y\_val)], [min(y\_val), max(y\_val)], color='red', lw=2)

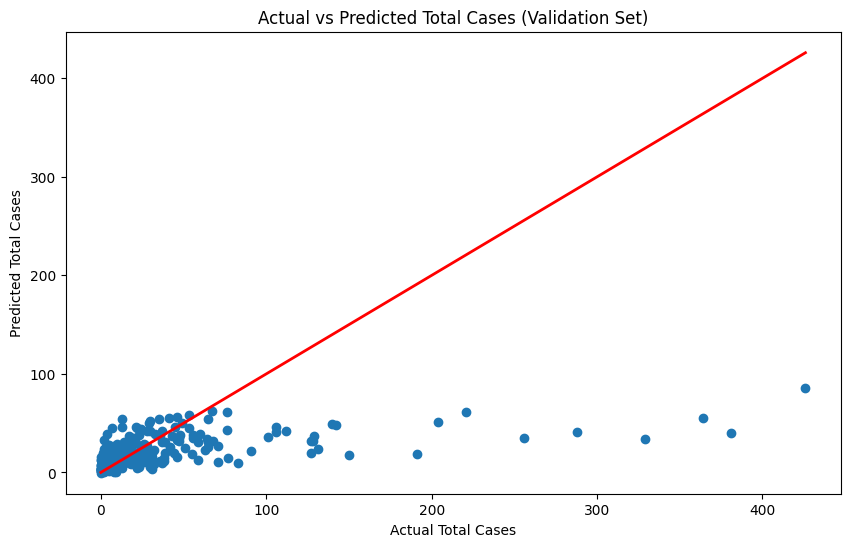
plt.xlabel('Actual Total Cases')

plt.ylabel('Predicted Total Cases')

plt.title('Actual vs Predicted Total Cases (Validation Set)')

plt.show()

##### **2.6.1.2 Ejecución**



Como podemos observar en la gráfica, se tiene una tendencia correcta pese a tener algunos outliers que según he podrido observar son valores que no cumplen todas las características correspondientes al tipo de dato o guardados en algunas columnas.

### **2.7 Entrenamiento final**

Entrenamiento usando todos los datos

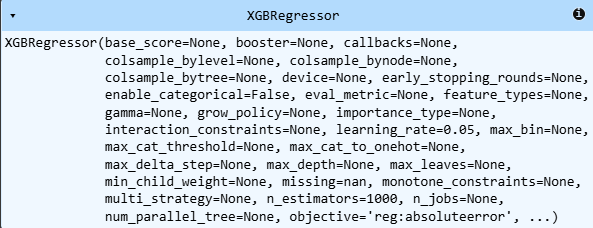
#### **2.7.1 Fit**

##### **2.7.1.1 Código**

# Train final model on all data

best\_model.fit(X, y)

##### **2.7.1.2 Ejecución**



#### **2.7.2 Preparación de los datos de test**

##### **2.7.1.1 Código**

El código usa un modelo entrenado (best\_model) para predecir los casos en los datos de prueba (test\_X). Luego, redondea las predicciones al entero más cercano para obtener valores realistas.

# Prepare test data (no total\_cases in test\_features)

test\_X = test.copy() # No need to drop 'total\_cases' since it doesn't exist in test data it was a temporal data

# Predict cases and round to nearest integer

predicted\_cases = best\_model.predict(test\_X)

predicted\_cases = np.round(predicted\_cases).astype(int)

### **2.8 Descarga de csv**

#### **2.7.1 Formato correcto**

##### **2.7.1.1 Código**

submission = test\_features[['year', 'weekofyear']].copy()

submission['total\_cases'] = predicted\_cases

#### **2.7.2 Añado la ciudad**

Aquí, asumimos que la predicción debe utilizar la ciudad codificada (0 o 1) según lo predicho por el modelo.

Usaremos el mapa inverso (0 -> 'sj', 1 -> 'iq') para la ciudad.

Puede utilizar una lógica para predecir la ciudad basándose en alguna regla, por ejemplo, a partir de otros datos o según los resultados del modelo.

Para simplificar, usaremos la misma ciudad para todas las muestras de prueba, ya que el modelo predice el total de casos para una ciudad específica.

##### **2.7.2.1 Código**

# Add the predicted city to the submission

submission.insert(0, 'city', predicted\_cities.map({0: 'sj', 1: 'iq'})) # Mapping city codes back to 'sj' and 'iq'

#### **2.7.3 Orden**

Añade el orden de las columnas

##### **2.7.3.1 Código**

## **submission = submission[['city', 'year', 'weekofyear', 'total\_cases']]**

#### 

#### **2.7.4 Guardar el csv**

Guarda los datos actualizados en el csv

##### **2.7.4.1 Código**

## **# Save to CSV with the correct format**

## **submission.to\_csv('submission.csv', index=False)**

#### 

#### **2.7.5 Descarga del csv**

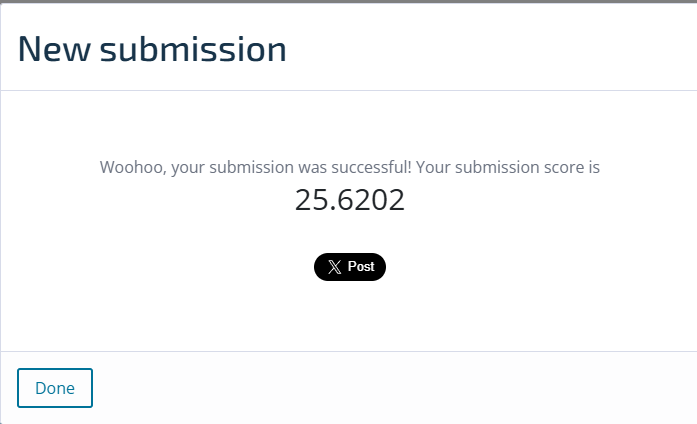
##### **2.7.5.1 Código**

# Download the file

files.download('submission.csv')

## **3. Resultado de la competición**

Tras subir el archivo csv, estos son los resultados obtenidos.



## 

## **3. Conclusiones**

Mi conclusión acerca de esta actividad y mi procedimiento de evolución en la misma es la siguiente, pese a la importancia de elección de modelo, entrenamiento y el conjunto de herramientas entorno al modelo descrito, la mayor carga de importancia y avance fue durante la búsqueda de mejoras a la hora de realizar la limpieza de datos, por lo que a partir de ahora comenzaré a preocuparme algo más en la limpieza y preprocesamiento de datos antes que en solo el uso del modelo o la mejor técnica.

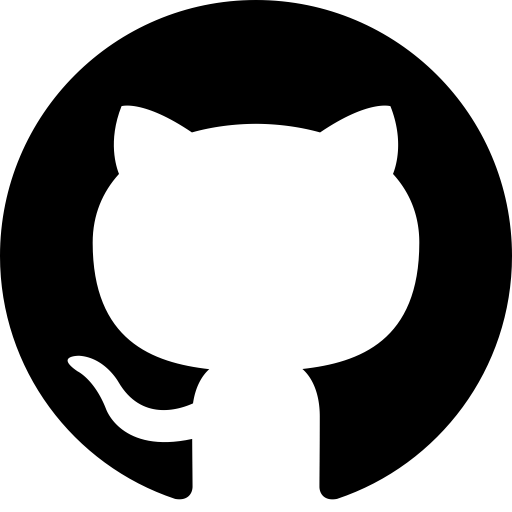
## 

## **4. Bibliografía**

Para el uso de la gráfica de importancia de las características use:  
<https://www.w3schools.com/python/matplotlib_pyplot.asp>

Y además utilicé la siguiente guía de XGBoost:  
<https://machinelearningmastery.com/xgboost-for-regression/>

## **5. Github y Colab**

[](https://github.com/AdrianYArmas/IaBigData/tree/main/SNS/3%20%20-%20Algoritmos%20y%20herramientas%20para%20el%20aprendizaje%20supervisado%20/3.6%20DengAI%20predicci%C3%B3n%20de%20la%20propagaci%C3%B3n%20de%20enfermedades) [](https://colab.research.google.com/drive/1h9Q0Pntu3UvmEdvuMPq8YMlQLHegYzxM?usp=sharing)