

## M.<sup>a</sup> de los Ángeles Martín de la Cruz

### ✓ Actividad 3.6 - DengAI: predicción de la propagación de enfermedades

### ✓ Utilizar el drive o GitHub como origen de ficheros para la importación del dataset.

```
import requests
import zipfile
import io

# Ruta del archivo ZIP (cambia blob por raw)
zip_url = "https://github.com/sntamaria/Actividad-3.6-DengAI-predicci-n-de-la-propagaci-n-de-enferm

# Descomprimir el archivo
response = requests.get(zip_url)
zip_file = zipfile.ZipFile(io.BytesIO(response.content))

# Extraer todos los archivos en una carpeta temporal
zip_file.extractall("dengue_data")

# Listar los archivos extraídos
import os
print(os.listdir("dengue_data/dengue_data"))
```

➡ ['dengue\_features\_train.csv', 'submission\_format.csv', 'dengue\_features\_test.csv', 'dengue\_labe



### ✓ Importación del dataset: Preparación de los datos: Normaliza, ajusta la calidad de los datos.

### ✓ 1. Importar los datasets (dengue\_features\_train.csv, dengue\_labels\_train.csv y dengue\_features\_test).

Fusionamos los datos de entrenamiento (dengue\_features\_train.csv) con las etiquetas (dengue\_labels\_train.csv) para que todas las variables estén en un solo DataFrame.

Es necesario porque dengue\_features\_train.csv contiene las variables predictoras (temperatura, humedad, precipitaciones, etc.) pero no incluye la columna total\_cases (número de casos de dengue).

dengue\_labels\_train.csv contiene solo city, year, weekofyear, total\_cases, donde total\_cases es el valor que queremos predecir.

Para entrenar un modelo, necesitamos que todas las variables y la salida (total\_cases) estén en un mismo dataset.

La fusión se hace con `on=["city", "year", "weekofyear"]`, que son las columnas comunes en ambos datasets.

Así, obtenemos un solo DataFrame (`df_train`) con todas las variables y la cantidad de casos de dengue.

```
import pandas as pd

# Cargar los datasets
df_features_train = pd.read_csv("dengue_data/dengue_data/dengue_features_train.csv")
df_labels_train = pd.read_csv("dengue_data/dengue_data/dengue_labels_train.csv")
df_test = pd.read_csv("dengue_data/dengue_data/dengue_features_test.csv")

# Mostrar las primeras filas para verificar la importación
#print("Características de entrenamiento:")
#print(df_features_train.head())

#print("\nEtiquetas de entrenamiento:")
#print(df_labels_train.head())

#print("\nDatos de prueba:")
#print(df_test.head())

# Mostrar información general de los datasets
#print("\nInformación del dataset de características:")
#df_features_train.info()

#print("\nInformación del dataset de prueba:")
#df_test.info()

# Fusionar datos de entrenamiento con las etiquetas
df_train = df_features_train.merge(df_labels_train, on=["city", "year", "weekofyear"], how="left")

# Mostrar las primeras filas para verificar la importación
print("Datos de entrenamiento:")
print(df_train.head())

print("\nDatos de prueba:")
print(df_test.head())

# Mostrar información general de los datasets
print("\nInformación del dataset de entrenamiento:")
df_train.info()

print("\nInformación del dataset de prueba:")
df_test.info()
```



Datos de entrenamiento:

	city	year	weekofyear	week_start_date	ndvi_ne	ndvi_nw	ndvi_se	\
0	sj	1990	18	1990-04-30	0.122600	0.103725	0.198483	
1	sj	1990	19	1990-05-07	0.169900	0.142175	0.162357	
2	sj	1990	20	1990-05-14	0.032250	0.172967	0.157200	
3	sj	1990	21	1990-05-21	0.128633	0.245067	0.227557	
4	sj	1990	22	1990-05-28	0.196200	0.262200	0.251200	
					ndvi_sw	precipitation_amt_mm	reanalysis_air_temp_k	...
0					0.177617	12.42	297.572857	...
1					0.155486	22.82	298.211429	...
2					0.170843	34.54	298.781429	...
3					0.235886	15.36	298.987143	...

```

4  0.247340          7.52          299.518571  ...

      reanalysis_relative_humidity_percent  reanalysis_sat_precip_amt_mm  \
0              73.365714              12.42
1              77.368571              22.82
2              82.052857              34.54
3              80.337143              15.36
4              80.460000              7.52

      reanalysis_specific_humidity_g_per_kg  reanalysis_tdtr_k  \
0              14.012857              2.628571
1              15.372857              2.371429
2              16.848571              2.300000
3              16.672857              2.428571
4              17.210000              3.014286

      station_avg_temp_c  station_diur_temp_rng_c  station_max_temp_c  \
0              25.442857              6.900000              29.4
1              26.714286              6.371429              31.7
2              26.714286              6.485714              32.2
3              27.471429              6.771429              33.3
4              28.942857              9.371429              35.0

      station_min_temp_c  station_precip_mm  total_cases
0              20.0              16.0              4
1              22.2              8.6              5
2              22.8              41.4              4
3              23.3              4.0              3
4              23.9              5.8              6

```

[5 rows x 25 columns]

Datos de prueba:

```

      city  year  weekofyear  week_start_date  ndvi_ne  ndvi_nw  ndvi_se  \
0    sj  2008         18    2008-04-29  -0.0189  -0.018900  0.102729
1    sj  2008         19    2008-05-06  -0.0180  -0.012400  0.082043
2    sj  2008         20    2008-05-13  -0.0015         NaN  0.151083
3    sj  2008         21    2008-05-20         NaN  -0.019867  0.124329
4    sj  2008         22    2008-05-27   0.0568   0.039833  0.062267

      ndvi_sw  precipitation_amt_mm  reanalysis_air_temp_k  ...  \
0  0.091200          78.60          298.492857  ...
1  0.072314          12.56          298.475714  ...
2  0.091529           3.66          299.455714  ...
3  0.125686           0.00          299.690000  ...

```

## 2. Explorar los datos (ver estructura, valores faltantes, tipos de datos).

```

# Contar valores nulos en cada dataset
print("\nValores nulos en el dataset de entrenamiento:")
print(df_train.isnull().sum())

print("\nValores nulos en el dataset de prueba:")
print(df_test.isnull().sum())

# Estadísticas generales del dataset de entrenamiento
print("\nEstadísticas descriptivas del dataset de entrenamiento:")
print(df_train.describe())

# Estadísticas generales del dataset de prueba
print("\nEstadísticas descriptivas del dataset de prueba:")
print(df_test.describe())

```



Valores nulos en el dataset de entrenamiento:

city	0
year	0
weekofyear	0
week_start_date	0
ndvi_ne	194
ndvi_nw	52
ndvi_se	22
ndvi_sw	22
precipitation_amt_mm	13
reanalysis_air_temp_k	10
reanalysis_avg_temp_k	10
reanalysis_dew_point_temp_k	10
reanalysis_max_air_temp_k	10
reanalysis_min_air_temp_k	10
reanalysis_precip_amt_kg_per_m2	10
reanalysis_relative_humidity_percent	10
reanalysis_sat_precip_amt_mm	13
reanalysis_specific_humidity_g_per_kg	10
reanalysis_tdtr_k	10
station_avg_temp_c	43
station_diur_temp_rng_c	43
station_max_temp_c	20
station_min_temp_c	14
station_precip_mm	22
total_cases	0

dtype: int64

Valores nulos en el dataset de prueba:

city	0
year	0
weekofyear	0
week_start_date	0
ndvi_ne	43
ndvi_nw	11
ndvi_se	1
ndvi_sw	1
precipitation_amt_mm	2
reanalysis_air_temp_k	2
reanalysis_avg_temp_k	2
reanalysis_dew_point_temp_k	2
reanalysis_max_air_temp_k	2
reanalysis_min_air_temp_k	2
reanalysis_precip_amt_kg_per_m2	2
reanalysis_relative_humidity_percent	2
reanalysis_sat_precip_amt_mm	2
reanalysis_specific_humidity_g_per_kg	2
reanalysis_tdtr_k	2
station_avg_temp_c	12
station_diur_temp_rng_c	12
station_max_temp_c	3
station_min_temp_c	9
station_precip_mm	5

dtype: int64

Estadísticas descriptivas del dataset de entrenamiento:

year	weekofyear	ndvi_ne	ndvi_nw	ndvi_se	\
------	------------	---------	---------	---------	---

### 3. Limpieza y normalización (manejo de valores nulos, escalado, transformación).

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
```

```

# Eliminar la columna 'week_start_date' porque no es relevante
df_train.drop(columns=["week_start_date"], inplace=True)
df_test.drop(columns=["week_start_date"], inplace=True)

# Separar variables numéricas para imputación y escalado (excluyendo total_cases)
num_cols = df_train.select_dtypes(include=["float64", "int64"]).columns.drop("total_cases")

# Imputar valores nulos con la mediana
imputer = SimpleImputer(strategy="median")
df_train[num_cols] = imputer.fit_transform(df_train[num_cols])
df_test[num_cols] = imputer.transform(df_test[num_cols])

# Escalar los datos (normalización Z-score)
scaler = StandardScaler()
df_train[num_cols] = scaler.fit_transform(df_train[num_cols])
df_test[num_cols] = scaler.transform(df_test[num_cols])

print("Limpieza y normalización completadas.")

# Ver las primeras filas
print(df_train.head())
print(df_test.head())

# Ver estadísticas resumidas
print(df_train.describe())
print(df_test.describe())

# Verificar valores nulos después de la imputación
print("Valores nulos en df_train:\n", df_train.isnull().sum().sum())
print("Valores nulos en df_test:\n", df_test.isnull().sum().sum())

```



Limpieza y normalización completadas.

	city	year	weekofyear	ndvi_ne	ndvi_nw	ndvi_se	ndvi_sw	\
0	sj	-2.040448	-0.566356	-0.136768	-0.224959	-0.070729	-0.294221	
1	sj	-2.040448	-0.499753	0.224679	0.101423	-0.563717	-0.560052	
2	sj	-2.040448	-0.433150	-0.827187	0.362797	-0.634092	-0.375586	
3	sj	-2.040448	-0.366547	-0.090664	0.974815	0.326020	0.405689	
4	sj	-2.040448	-0.299943	0.425654	1.120251	0.648657	0.543275	

	precipitation_amt_mm	reanalysis_air_temp_k	reanalysis_avg_temp_k	...	\
0	-0.764739	-0.831532	-1.179965	...	
1	-0.525716	-0.361050	-0.623062	...	
2	-0.256355	0.058910	-0.276418	...	
3	-0.697169	0.210475	0.002033	...	
4	-0.877355	0.602017	0.348677	...	

	reanalysis_relative_humidity_percent	reanalysis_sat_precip_amt_mm	\
0	-1.232166	-0.764739	
1	-0.670635	-0.525716	
2	-0.013513	-0.256355	
3	-0.254197	-0.697169	
4	-0.236962	-0.877355	

	reanalysis_specific_humidity_g_per_kg	reanalysis_tdtr_k	\
0	-1.780132	-0.639266	
1	-0.895242	-0.711966	
2	0.064938	-0.732160	
3	-0.049391	-0.695811	
4	0.300104	-0.530217	

	station_avg_temp_c	station_diur_temp_rng_c	station_max_temp_c	\
0	-1.374171	-0.541357	-1.571380	
1	-0.375608	-0.793046	-0.389200	

2	-0.375608	-0.738627	-0.132205
3	0.219042	-0.602579	0.433186
4	1.374682	0.635459	1.306971

	station_min_temp_c	station_precip_mm	total_cases
0	-1.342999	-0.490112	4
1	0.061885	-0.647168	5
2	0.445035	0.048974	4
3	0.764326	-0.744798	3
4	1.147476	-0.706595	6

[5 rows x 24 columns]

	city	year	weekofyear	ndvi_ne	ndvi_nw	ndvi_se	ndvi_sw	\
0	sj	1.288905	-0.566356	-1.218055	-1.265857	-1.377423	-1.332233	
1	sj	1.288905	-0.499753	-1.211177	-1.210682	-1.659706	-1.559083	
2	sj	1.288905	-0.433150	-1.085091	-0.074683	-0.717562	-1.328287	
3	sj	1.288905	-0.366547	-0.089263	-1.274062	-1.082664	-0.918001	
4	sj	1.288905	-0.299943	-0.639586	-0.767301	-1.929578	-1.515841	

	precipitation_amt_mm	reanalysis_air_temp_k	reanalysis_avg_temp_k	...	\
0	0.756275	-0.153702	-0.537822	...	
1	-0.761521	-0.166332	-0.532139	...	
2	-0.966070	0.555705	0.104322	...	
3	-1.050187	0.728320	0.399821	...	
4	-1.032720	0.794630	0.354360	...	

## ✓ 4. Convertir variables categóricas

### ✓ Opción 1: Excluir "city" en cálculos numéricos (Recomendada si "city" solo identifica ubicaciones)

Si "city" solo indica la ciudad y no tiene una relación directa con los valores numéricos, lo mejor es excluirla temporalmente en los cálculos de correlación y selección de características, pero mantenerla en el dataset para el submission final.

Evita errores sin modificar los datos originales. Mantiene la estructura del submission intacta.

### ✓ Opción 2: Codificar "city" como variable numérica (Recomendada si "city" influye en la predicción)

Si la ciudad afecta los casos de dengue (por clima, ubicación, etc.), podemos convertirla en números usando Label Encoding o One-Hot Encoding.

### ✓ Label Encoding

(Si solo hay 2 ciudades): Convierte "sj" en 0 y "iq" en 1

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df_train["city"] = le.fit_transform(df_train["city"])
df_test["city"] = le.transform(df_test["city"])
```

## ➤ One-Hot Encoding

[ ] ↳ 2 celdas ocultas

## ✓ Conclusión:

Como "city" tiene relación con los casos de dengue vamos a usar LabelEncoding, ya que sólo existen dos ciudades en el dataset.

## ✓ Selección de características: Utiliza métodos no gráficos para la selección de características.

### ✓ 1. Filtrado basado en correlación

Eliminamos variables altamente correlacionadas para evitar redundancia.

```
import numpy as np

# Seleccionar solo columnas numéricas
# (como city ya es una columna numérica lo dejamos así, pero si no pondríamos lo de siempre "correl
df_train_numeric = df_train.select_dtypes(include=[np.number])

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

# Encontrar características con alta correlación (mayor a 0.85)
threshold = 0.85
high_corr_features = set()

for col in correlation_matrix.columns:
    for row in correlation_matrix.index:
        if col != row and abs(correlation_matrix.loc[row, col]) > threshold:
            high_corr_features.add(col)

# Eliminar características redundantes
df_train_selected = df_train.drop(columns=high_corr_features)
df_test_selected = df_test.drop(columns=high_corr_features)

# Mostrar las características eliminadas y las seleccionadas
print(f"Características eliminadas por alta correlación: {high_corr_features}")
print(f"Características seleccionadas: {df_train_selected.columns.tolist()}")
```

↔ Características eliminadas por alta correlación: {'reanalysis\_specific\_humidity\_g\_per\_kg', 'cit  
Características seleccionadas: ['year', 'weekofyear', 'ndvi\_ne', 'ndvi\_nw', 'ndvi\_se', 'ndvi\_sw

### ✓ 2. Selección por Importancia de Características con Random Forest

Nos ayuda a encontrar qué variables tienen mayor impacto en la predicción.

```
from sklearn.ensemble import RandomForestRegressor

# Definir modelo de prueba
X = df_train.drop(columns=["total_cases"]) # Variables independientes
y = df_train["total_cases"] # Variable objetivo

model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X, y)

# Obtener importancia de características
feature_importance = dict(zip(X.columns, model.feature_importances_))

# Ordenar por importancia
sorted_features = sorted(feature_importance.items(), key=lambda x: x[1], reverse=True)

# Mantener solo las 10 características más importantes
top_features = [f[0] for f in sorted_features[:10]]
df_train_selected = df_train[top_features + ["total_cases"]]
df_test_selected = df_test[top_features]

print(f"Top características seleccionadas: {top_features}")
```

➡ Top características seleccionadas: ['year', 'weekofyear', 'ndvi\_sw', 'reanalysis\_air\_temp\_k', 'reanalysis\_soil\_moisture\_k', 'reanalysis\_total\_precip\_k', 'reanalysis\_u10\_k', 'reanalysis\_v10\_k', 'reanalysis\_v20\_k', 'reanalysis\_v20\_u10\_k']

### ✓ 3. Selección Automática con SelectKBest

Selecciona las K mejores características según una métrica estadística.

```
from sklearn.feature_selection import SelectKBest, f_regression

# Aplicar SelectKBest para seleccionar las 10 mejores características
selector = SelectKBest(score_func=f_regression, k=10)
X_new = selector.fit_transform(X, y)

# Obtener las columnas seleccionadas
selected_features = X.columns[selector.get_support()]
df_train_selected = df_train[selected_features.tolist() + ["total_cases"]]
df_test_selected = df_test[selected_features.tolist()]

print(f"Características seleccionadas con SelectKBest: {selected_features.tolist()}")
```

➡ Características seleccionadas con SelectKBest: ['city', 'year', 'weekofyear', 'ndvi\_ne', 'reanalysis\_air\_temp\_k', 'reanalysis\_soil\_moisture\_k', 'reanalysis\_total\_precip\_k', 'reanalysis\_u10\_k', 'reanalysis\_v10\_k', 'reanalysis\_v20\_k']

### ✓ 4. RFE (Recursive Feature Elimination)

Es una técnica de selección de características basada en la importancia de los modelos.



```

from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestRegressor

# Definir el modelo base para evaluar la importancia de las características
model = RandomForestRegressor(n_estimators=100, random_state=42)

# Seleccionar solo columnas numéricas (sin la variable objetivo)
X = df_train_numeric.drop(columns=["total_cases"], errors="ignore")
y = df_train_numeric["total_cases"]

# Aplicar RFE con un número de características a seleccionar (ajústalo según prefieras)
selector = RFE(model, n_features_to_select=10) # Puedes cambiar el 10 por otro número deseado
X_selected = selector.fit_transform(X, y)

# Obtener nombres de las características seleccionadas
selected_features = X.columns[selector.support_].tolist()

# Aplicar la selección de características en los datos de entrenamiento y prueba
df_train_selected = df_train[selected_features + ["total_cases"]] # Agregar la variable objetivo
df_test_selected = df_test[selected_features] # Test no tiene total_cases

# Mostrar las características seleccionadas
print(f"Características seleccionadas por RFE: {selected_features}")

```

↔ Características seleccionadas por RFE: ['year', 'weekofyear', 'ndvi\_nw', 'ndvi\_sw', 'reanalysis']

## ✓ Selección de características: Utiliza herramientas gráficas para la elección de las características.

```

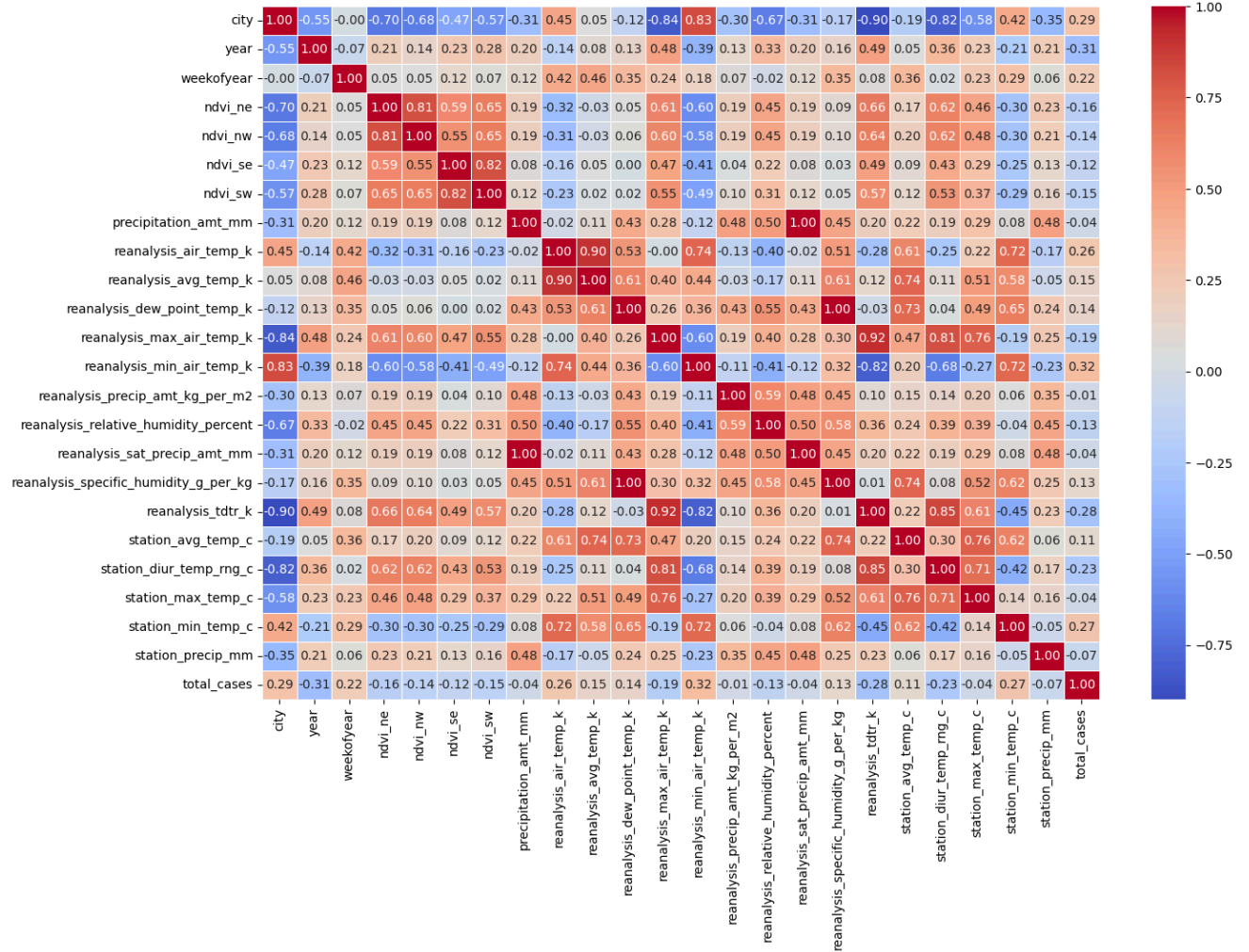
import seaborn as sns
import matplotlib.pyplot as plt

# Graficar la matriz de correlación como un mapa de calor (heatmap)
plt.figure(figsize=(15, 10))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Matriz de correlación de características")
plt.show()

```



Matriz de correlación de características



```
import matplotlib.pyplot as plt
```

```
# Gráfico de barras de las 10 características más importantes
```

```
top_feature_importance = [f[1] for f in sorted_features[:10]]
```

```
top_feature_names = [f[0] for f in sorted_features[:10]]
```

```
plt.figure(figsize=(12, 6))
```

```
plt.barh(top_feature_names, top_feature_importance, color='skyblue')
```

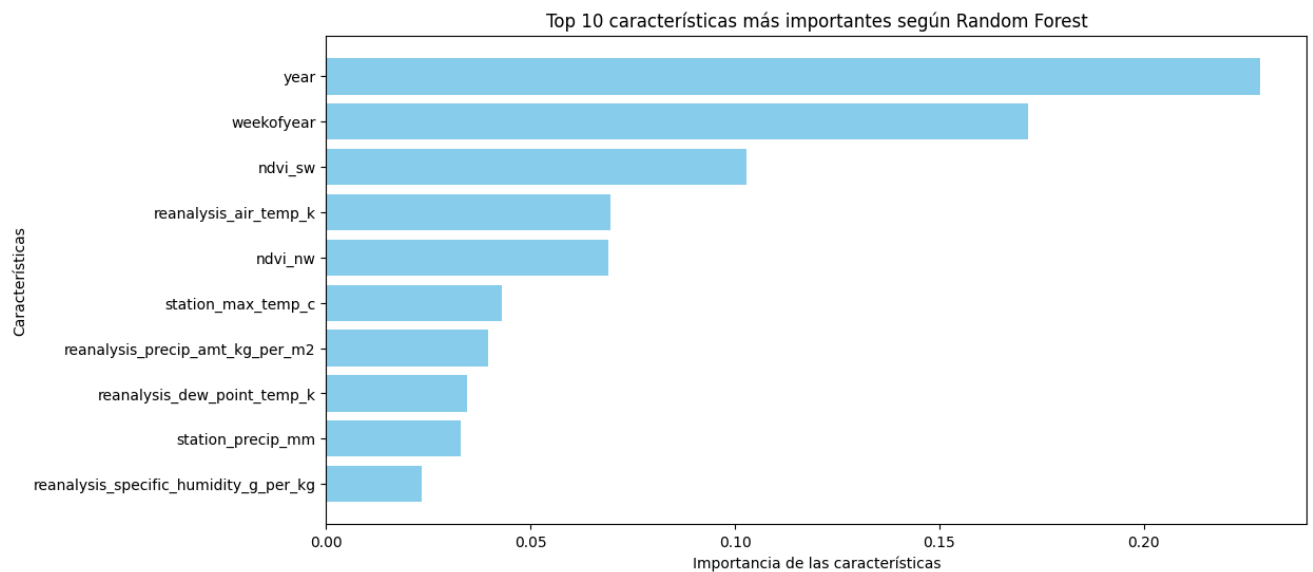
```
plt.xlabel("Importancia de las características")
```

```
plt.ylabel("Características")
```

```
plt.title("Top 10 características más importantes según Random Forest")
```

```
plt.gca().invert_yaxis() # Para mostrar la característica más importante en la parte superior
```

```
plt.show()
```



```
# Puntuaciones F de las características seleccionadas por SelectKBest
```

```
selected_scores = selector.scores_[selector.get_support()]
```

```
selected_feature_names = selected_features
```

```
# Ordenar características por puntuación F de mayor a menor
```

```
sorted_features_scores = sorted(zip(selected_feature_names, selected_scores), key=lambda x: x[1], r
```

```
sorted_feature_names, sorted_scores = zip(*sorted_features_scores)
```

```
# Gráfico de barras de las puntuaciones F, ahora ordenadas
```

```
plt.figure(figsize=(12, 6))
```

```
plt.barh(sorted_feature_names, sorted_scores, color='lightcoral')
```

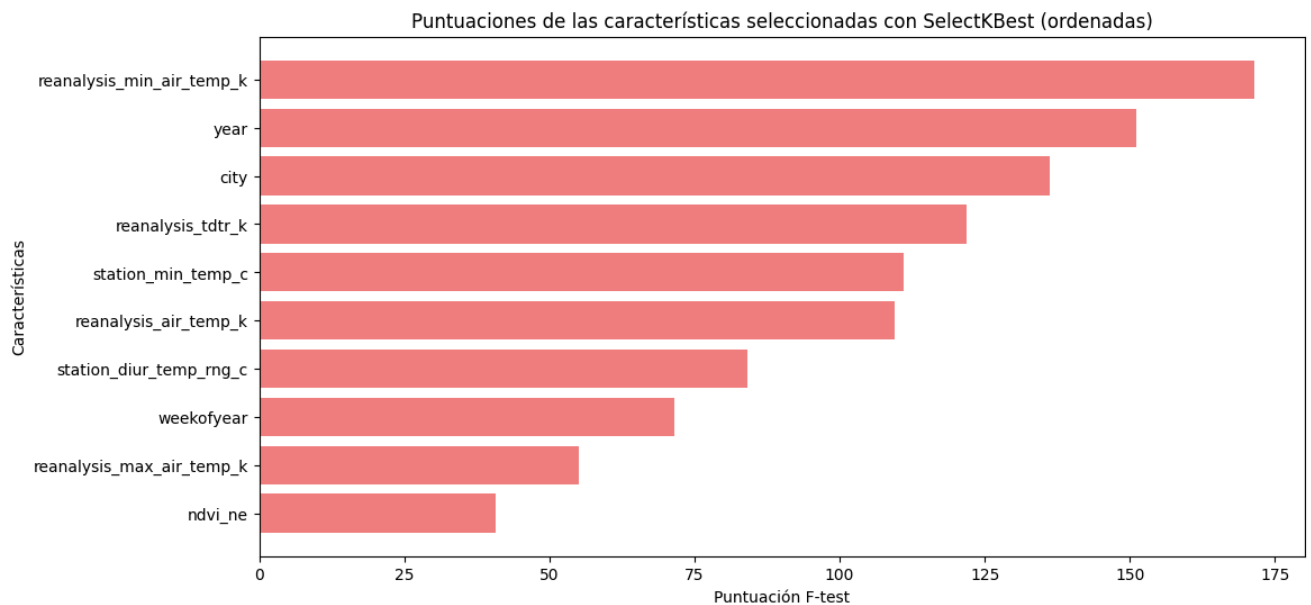
```
plt.xlabel("Puntuación F-test")
```

```
plt.ylabel("Características")
```

```
plt.title("Puntuaciones de las características seleccionadas con SelectKBest (ordenadas)")
```

```
plt.gca().invert_yaxis() # Para mostrar la característica con la mayor puntuación en la parte supe
```

```
plt.show()
```

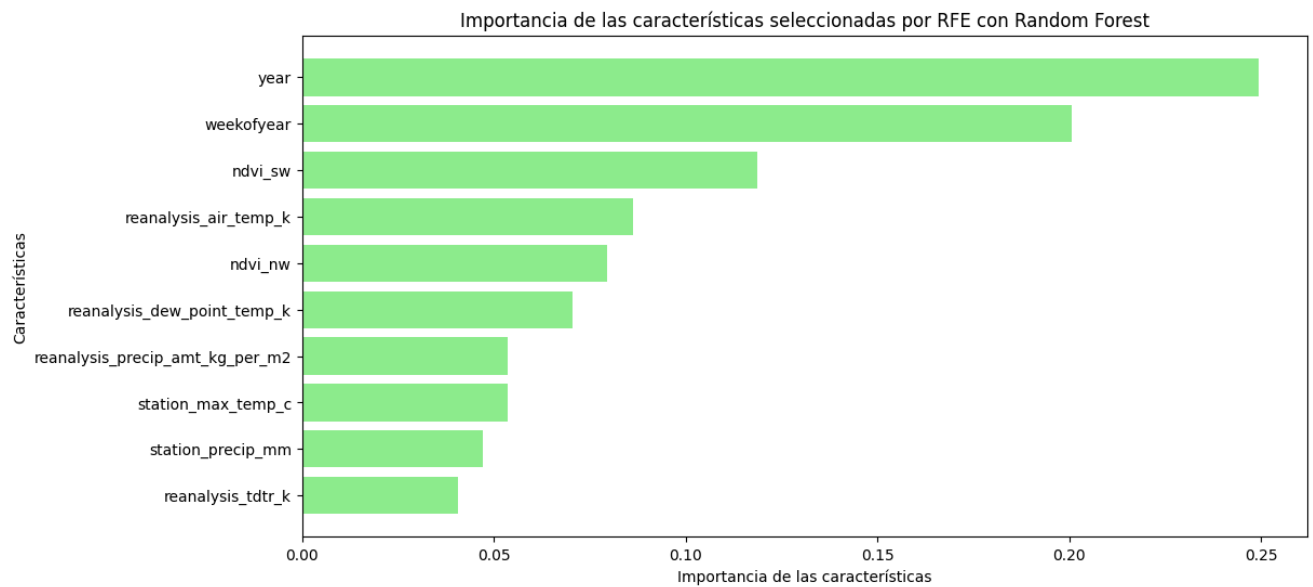


```
# Primero, ajustamos el modelo con las características seleccionadas por RFE
model.fit(X_selected, y) # Usamos X_selected que contiene las características seleccionadas

# Obtener la importancia de las características seleccionadas por el modelo Random Forest
feature_importance_rfe = dict(zip(selected_features, model.feature_importances_))

# Ordenar por importancia
sorted_importance_rfe = sorted(feature_importance_rfe.items(), key=lambda x: x[1], reverse=True)
sorted_features_rfe, sorted_importance_rfe = zip(*sorted_importance_rfe)

# Gráfico de barras de la importancia de las características seleccionadas por RFE
plt.figure(figsize=(12, 6))
plt.barh(sorted_features_rfe, sorted_importance_rfe, color='lightgreen')
plt.xlabel("Importancia de las características")
plt.ylabel("Características")
plt.title("Importancia de las características seleccionadas por RFE con Random Forest")
plt.gca().invert_yaxis() # Para mostrar la característica más importante en la parte superior
plt.show()
```



- ✓ Además de la división de los datos de train y test, incorpora la utilización de datos de validación.

```
from sklearn.model_selection import train_test_split

# Definir las características seleccionadas
selected_features = ['year', 'weekofyear', 'ndvi_sw', 'reanalysis_air_temp_k',
                    'ndvi_nw', 'station_max_temp_c', 'reanalysis_precip_amt_kg_per_m2',
                    'reanalysis_dew_point_temp_k', 'station_precip_mm', 'reanalysis_tdtr_k']

# Filtrar los datos con solo las características seleccionadas
X = df_train[selected_features]
y = df_train["total_cases"]

# Dividir en train (70%), validation (15%) y test (15%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Mostrar tamaños de los conjuntos
print(f"Tamaño de train: {X_train.shape}")
print(f"Tamaño de validación: {X_val.shape}")
print(f"Tamaño de test: {X_test.shape}")
```



```
Tamaño de train: (1019, 10)
Tamaño de validación: (218, 10)
Tamaño de test: (219, 10)
```

## Entrenamiento: Modelo 1º NaiveBayes - Desarrolla las diversas pruebas

- ✓ propuestas para la selección y justifica el criterio de calidad para la selección del modelo. Utiliza Cross Validation

- ✓ Escalamos los datos

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)

from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_absolute_error, accuracy_score

# Inicializar el modelo Naive Bayes
nb_model = BernoulliNB()

from sklearn.model_selection import KFold, cross_val_score

kf = KFold(n_splits=5, shuffle=True, random_state=42) # Shuffle para mezclar los datos
cv_scores = cross_val_score(nb_model, X_train, y_train, cv=kf, scoring="neg_mean_absolute_error")

# Aplicar validación cruzada con 5 folds
#cv_scores = cross_val_score(nb_model, X_train, y_train, cv=5, scoring="neg_mean_absolute_error")

# Entrenar el modelo con los datos de entrenamiento
nb_model.fit(X_train, y_train)

# Evaluación en el conjunto de validación
y_val_pred = nb_model.predict(X_val)

# Calcular métricas
mae = mean_absolute_error(y_val, y_val_pred)
accuracy = accuracy_score(y_val, y_val_pred)

# Mostrar resultados
print(f"Puntuaciones de Cross Validation (MAE): {-cv_scores}")
print(f"Media de Cross Validation (MAE): {-cv_scores.mean()}")
print(f"MAE en conjunto de validación: {mae}")
print(f"Accuracy en conjunto de validación: {accuracy}")
```

➡ Puntuaciones de Cross Validation (MAE): [18.28921569 16.51470588 15.00490196 16.19607843 19.847  
Media de Cross Validation (MAE): 17.17043852023568  
MAE en conjunto de validación: 18.06422018348624  
Accuracy en conjunto de validación: 0.0871559633027523

```
from sklearn.model_selection import KFold, cross_val_score
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import mean_absolute_error, accuracy_score
```

```

import numpy as np

# Convertir total_cases en categorías (Ejemplo: 0 = pocos casos, 1 = muchos casos)
threshold = np.median(y_train) # Umbral basado en la mediana
y_train_class = (y_train > threshold).astype(int)
y_val_class = (y_val > threshold).astype(int)

# Inicializar el modelo Naive Bayes
nb_model = GaussianNB()

# Validación cruzada con MAE
kf = KFold(n_splits=5, shuffle=True, random_state=42)
cv_scores_mae = cross_val_score(nb_model, X_train, y_train, cv=kf, scoring="neg_mean_absolute_error")

# Entrenar el modelo con los datos de entrenamiento
nb_model.fit(X_train, y_train_class) # Entrenamos con la variable categorizada

# Evaluación en el conjunto de validación
y_val_pred = nb_model.predict(X_val)

# Calcular métricas
mae = mean_absolute_error(y_val, y_val_pred)
accuracy = accuracy_score(y_val_class, y_val_pred)

# Mostrar resultados
print(f"Puntuaciones de Cross Validation (MAE): {-cv_scores_mae}")
print(f"Media de Cross Validation (MAE): {-cv_scores_mae.mean()}")
print(f"MAE en conjunto de validación: {mae}")
print(f"Accuracy en conjunto de validación: {accuracy}")

```

```

➡ Puntuaciones de Cross Validation (MAE): [21.22058824 19.06862745 18.60784314 25.21078431 24.960
Media de Cross Validation (MAE): 21.813686854051966
MAE en conjunto de validación: 23.692660550458715
Accuracy en conjunto de validación: 0.7293577981651376

```

```

print(y.value_counts()) # Muestra la frecuencia de cada número de casos de dengue

```

```

➡ total_cases
0      100
6       71
5       70
3       70
2       69
...
288     1
221     1
149     1
154     1
58      1
Name: count, Length: 135, dtype: int64

```

✓ La Estratificación sólo se usa si y es categórica

✓ Estratificación en train\_test\_split

Si y tiene valores desbalanceados, puedes asegurarte de que cada conjunto tenga una distribución similar usando stratify=y

```
#X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42, stratify
```

## Entrenamiento: Modelo 2º KNN - Desarrolla las diversas pruebas

✓ propuestas para la selección y justifica el criterio de calidad para la selección del modelo. Utiliza Cross Validation

✓ Definir y entrenar el modelo KNN

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score, KFold
from sklearn.metrics import mean_absolute_error, accuracy_score
import numpy as np

# Probar diferentes valores de k para encontrar el mejor
k_values = [3, 5, 7, 9, 11] # Lista de valores de k a probar
best_k = None
best_score = 0

for k in k_values:
    knn_model = KNeighborsClassifier(n_neighbors=k)
    kf = KFold(n_splits=5, shuffle=True, random_state=42)
    cv_scores = cross_val_score(knn_model, X_train, y_train_class, cv=kf, scoring="accuracy") # Ev

    mean_score = cv_scores.mean()
    print(f"K={k}, Accuracy Cross Validation: {mean_score}")

    if mean_score > best_score:
        best_score = mean_score
        best_k = k

print(f"\nMejor valor de k: {best_k} con Accuracy: {best_score}")
```

```
➡ K=3, Accuracy Cross Validation: 0.751704819858978
K=5, Accuracy Cross Validation: 0.747783251231527
K=7, Accuracy Cross Validation: 0.7683811455616729
K=9, Accuracy Cross Validation: 0.7722978846711098
K=11, Accuracy Cross Validation: 0.7693711967545639
```

Mejor valor de k: 9 con Accuracy: 0.7722978846711098

Evaluar el modelo con el mejor k

```
# Entrenar el modelo con el mejor k encontrado
knn_best = KNeighborsClassifier(n_neighbors=best_k)
knn_best.fit(X_train, y_train_class)

# Predicciones en validación
y_val_pred = knn_best.predict(X_val)
```



```
# Calcular métricas
mae = mean_absolute_error(y_val_class, y_val_pred)
accuracy = accuracy_score(y_val_class, y_val_pred)

# Mostrar resultados
print(f"\nMAE en conjunto de validación: {mae}")
print(f"Accuracy en conjunto de validación: {accuracy}")
```



```
MAE en conjunto de validación: 0.2018348623853211
Accuracy en conjunto de validación: 0.7981651376146789
```

## ✓ KNN con hiperparámetros

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.metrics import mean_absolute_error, accuracy_score
import numpy as np

# Definir el modelo KNN
knn = KNeighborsClassifier()

# Definir los hiperparámetros a probar
param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11, 15], # Diferentes valores de k
    'weights': ['uniform', 'distance'], # Cómo ponderar los vecinos
    'metric': ['euclidean', 'manhattan'] # Distancia usada
}

# Configurar validación cruzada
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Aplicar GridSearchCV para encontrar la mejor combinación de hiperparámetros
grid_search = GridSearchCV(knn, param_grid, cv=kf, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train_class)

# Mostrar los mejores hiperparámetros encontrados
print(f"Mejores hiperparámetros: {grid_search.best_params_}")
print(f"Mejor Accuracy en Cross Validation: {grid_search.best_score_}")
```



```
Mejores hiperparámetros: {'metric': 'manhattan', 'n_neighbors': 15, 'weights': 'distance'}
Mejor Accuracy en Cross Validation: 0.7821066357577513
```

```
# Entrenar KNN con los mejores hiperparámetros
knn_best = grid_search.best_estimator_
knn_best.fit(X_train, y_train_class)

# Predicciones en el conjunto de validación
y_val_pred = knn_best.predict(X_val)

# Calcular métricas
mae = mean_absolute_error(y_val_class, y_val_pred)
accuracy = accuracy_score(y_val_class, y_val_pred)

# Mostrar resultados
print(f"\nMAE en conjunto de validación: {mae}")
```

```
print(f"Accuracy en conjunto de validación: {accuracy}")
```



```
MAE en conjunto de validación: 0.1743119266055046  
Accuracy en conjunto de validación: 0.8256880733944955
```

## Entrenamiento: Modelo 3º de elección libre - Desarrolla las diversas

- ✓ pruebas propuestas para la selección y justifica el criterio de calidad para la selección del modelo. Utiliza Cross Validation

### ✓ Random Forest

```
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor  
from sklearn.model_selection import cross_val_score, KFold  
from sklearn.metrics import mean_absolute_error, accuracy_score  
  
# Convertimos total_cases en categorías usando la mediana  
threshold = y_train.median()  
y_train_class = (y_train > threshold).astype(int)  
y_val_class = (y_val > threshold).astype(int)  
  
kf = KFold(n_splits=5, shuffle=True, random_state=42) # Definimos los 5 folds  
  
# Random Forest - Regresión (MAE)  
rf_reg = RandomForestRegressor(n_estimators=100, max_depth=10, random_state=42)  
cv_scores_rf_mae = cross_val_score(rf_reg, X_train, y_train, cv=kf, scoring="neg_mean_absolute_error")  
  
# Entrenar el modelo y evaluar en validación  
rf_reg.fit(X_train, y_train)  
y_val_pred_rf_reg = rf_reg.predict(X_val)  
mae_rf = mean_absolute_error(y_val, y_val_pred_rf_reg)  
  
# Random Forest - Clasificación (Accuracy)  
rf_clf = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)  
cv_scores_rf_acc = cross_val_score(rf_clf, X_train, y_train_class, cv=kf, scoring="accuracy")  
  
# Entrenar el modelo y evaluar en validación  
rf_clf.fit(X_train, y_train_class)  
y_val_pred_rf_class = rf_clf.predict(X_val)  
accuracy_rf = accuracy_score(y_val_class, y_val_pred_rf_class)  
  
print(f"Random Forest:")  
print(f"MAE sin Cross Validation: {mae_rf:.4f}")  
print(f"Accuracy sin Cross Validation: {accuracy_rf:.4f}")  
print(f"Cross Validation MAE: {-cv_scores_rf_mae.mean():.4f}")  
print(f"Cross Validation Accuracy: {cv_scores_rf_acc.mean():.4f}")
```



```
Random Forest:  
MAE sin Cross Validation: 11.6353  
Accuracy sin Cross Validation: 0.8578  
Cross Validation MAE: 13.7386  
Cross Validation Accuracy: 0.8047
```

## ✓ XGBoost

XGBoost (Extreme Gradient Boosting) es un modelo basado en árboles de decisión que mejora su rendimiento usando una técnica llamada Boosting.

1. Empieza con un modelo base muy simple (como un árbol de decisión pequeño).
2. Se entrena en los datos y mide qué tan mal predice.
3. Se entrena otro modelo corrigiendo los errores del anterior.
4. Se repite el proceso varias veces hasta que el modelo mejora su rendimiento.
5. Al final, combina todos los modelos en una predicción final más precisa.

Ventajas de XGBoost:

- Es rápido y optimizado para grandes volúmenes de datos.
- Maneja bien valores faltantes y relaciones no lineales.
- Tiene regularización para evitar overfitting.
- Funciona mejor que Random Forest en muchos casos.

Diferencia con Random Forest:

Random Forest crea muchos árboles de decisión y promedia sus resultados. XGBoost crea árboles de decisión secuenciales, mejorando cada uno a partir de los errores del anterior. Por eso XGBoost suele ser más preciso, como podemos comprobar en los resultados.

```
from xgboost import XGBRegressor, XGBClassifier

# XGBoost - Regresión (MAE)
xgb_reg = XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=5, random_state=42)
cv_scores_xgb_mae = cross_val_score(xgb_reg, X_train, y_train, cv=kf, scoring="neg_mean_absolute_er

# Entrenar el modelo y evaluar en validación
xgb_reg.fit(X_train, y_train)
y_val_pred_xgb_reg = xgb_reg.predict(X_val)
mae_xgb = mean_absolute_error(y_val, y_val_pred_xgb_reg)

# XGBoost - Clasificación (Accuracy)
xgb_clf = XGBClassifier(n_estimators=100, learning_rate=0.1, max_depth=5, random_state=42)
cv_scores_xgb_acc = cross_val_score(xgb_clf, X_train, y_train_class, cv=kf, scoring="accuracy")

# Entrenar el modelo y evaluar en validación
xgb_clf.fit(X_train, y_train_class)
y_val_pred_xgb_class = xgb_clf.predict(X_val)
accuracy_xgb = accuracy_score(y_val_class, y_val_pred_xgb_class)

print(f"XGBoost:")
print(f"MAE sin Cross Validation: {mae_xgb:.4f}")
print(f"Accuracy sin Cross Validation: {accuracy_xgb:.4f}")
print(f"Cross Validation MAE: {-cv_scores_xgb_mae.mean():.4f}")
print(f"Cross Validation Accuracy: {cv_scores_xgb_acc.mean():.4f}")
```



XGBoost:

MAE sin Cross Validation: 10.1674

Accuracy sin Cross Validation: 0.8578  
Cross Validation MAE: 12.0229  
Cross Validation Accuracy: 0.8243

## ✓ XGBoost con GridSearch

Podemos mejorar el rendimiento de XGBoost ajustando sus hiperparámetros. Esto se llama Hyperparameter Tuning y se hace con técnicas como:

- Grid Search → Prueba todas las combinaciones posibles (más preciso, pero lento).
- Random Search → Prueba combinaciones aleatorias (más rápido, pero menos preciso).
- Optuna o Bayesian Optimization → Encuentra los mejores valores de forma inteligente (avanzado, pero muy potente).

Hiperparámetros clave en XGBoost

Los más importantes que podemos ajustar son:

1. `n_estimators` → Número de árboles (más árboles = mejor predicción, pero más lento).
2. `learning_rate` → Qué tan rápido aprende (valores bajos evitan overfitting, pero requieren más árboles).
3. `max_depth` → Profundidad máxima de cada árbol (muy alto puede causar overfitting).
4. `subsample` → Porcentaje de datos usados en cada árbol (reduce overfitting).
5. `colsample_bytree` → Cuántas características usa cada árbol (evita que un solo atributo domine).
6. `gamma` → Penalización para evitar árboles demasiado complejos (reduce overfitting).
7. `reg_alpha` y `reg_lambda` → Regularización L1 y L2 para evitar overfitting.

```
from xgboost import XGBRegressor, XGBClassifier
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.metrics import mean_absolute_error, accuracy_score

# Convertimos total_cases en categorías
threshold = y_train.median()
y_train_class = (y_train > threshold).astype(int)
y_val_class = (y_val > threshold).astype(int)

kf = KFold(n_splits=5, shuffle=True, random_state=42) # Cross Validation 5 folds

# Optimización de XGBoost - REGRESIÓN (MAE)
xgb_reg = XGBRegressor(objective="reg:squarederror", random_state=42)
param_grid = {
    "n_estimators": [100, 300, 500],
    "learning_rate": [0.01, 0.05, 0.1],
    "max_depth": [3, 5, 7],
    "subsample": [0.7, 0.8, 1.0],
    "colsample_bytree": [0.7, 0.8, 1.0]
}

grid_search_reg = GridSearchCV(xgb_reg, param_grid, cv=kf, scoring="neg_mean_absolute_error", n_jobs=-1)
grid_search_reg.fit(X_train, y_train) # ⚠ Se usa y_train original (NO categorizado)

# Mejor modelo regresor
best_xgb_reg = grid_search_reg.best_estimator_
y_val_pred_xgb_reg = best_xgb_reg.predict(X_val)

# Métricas regresión
mae_xgb_opt = mean_absolute_error(y_val, y_val_pred_xgb_reg)
```

```

cv_mae_xgb_opt = -grid_search_reg.best_score_

# Optimización de XGBoost - CLASIFICACIÓN (Accuracy)
xgb_clf = XGBClassifier(random_state=42)
grid_search_clf = GridSearchCV(xgb_clf, param_grid, cv=kf, scoring="accuracy", n_jobs=-1, verbose=2)
grid_search_clf.fit(X_train, y_train_class) # ⚠ Se usa y_train_class (CATEGORIZADO)

# Mejor modelo clasificador
best_xgb_clf = grid_search_clf.best_estimator_
y_val_pred_xgb_class = best_xgb_clf.predict(X_val)

# Métricas clasificación
accuracy_xgb_opt = accuracy_score(y_val_class, y_val_pred_xgb_class)
cv_accuracy_xgb_opt = grid_search_clf.best_score_

# Resultados Finales
print(f"XGBoost Optimizado:")
print(f"Mejores hiperparámetros (Regresión): {grid_search_reg.best_params_}")
print(f"Mejores hiperparámetros (Clasificación): {grid_search_clf.best_params_}")

print(f"MAE (Validación): {mae_xgb_opt:.4f}")
print(f"Cross Validation MAE: {cv_mae_xgb_opt:.4f}")

print(f"Accuracy (Validación): {accuracy_xgb_opt:.4f}")
print(f"Cross Validation Accuracy: {cv_accuracy_xgb_opt:.4f}")

➡ Fitting 5 folds for each of 243 candidates, totalling 1215 fits
Fitting 5 folds for each of 243 candidates, totalling 1215 fits
XGBoost Optimizado:
Mejores hiperparámetros (Regresión): {'colsample_bytree': 1.0, 'learning_rate': 0.05, 'max_depth': 10}
Mejores hiperparámetros (Clasificación): {'colsample_bytree': 1.0, 'learning_rate': 0.01, 'max_depth': 10}
MAE (Validación): 10.2121
Cross Validation MAE: 10.9760
Accuracy (Validación): 0.8624
Cross Validation Accuracy: 0.8292

```

Entrenamiento - Uso de gráficos: Integra el uso de gráficos para obtener comparativas en el entrenamiento de los modelos.

```

import matplotlib.pyplot as plt
import numpy as np

# Resultados obtenidos
modelos = ["Random Forest", "XGBoost"]
mae_validacion = [11.6353, 10.1674]
mae_cross_val = [13.7386, 12.0229]
accuracy_validacion = [0.8578, 0.8578]
accuracy_cross_val = [0.8047, 0.8243]

# Configurar el gráfico
x = np.arange(len(modelos)) # Posición de las barras
width = 0.2 # Ancho de las barras

fig, ax = plt.subplots(figsize=(10, 6))

# Barras para MAE

```

```

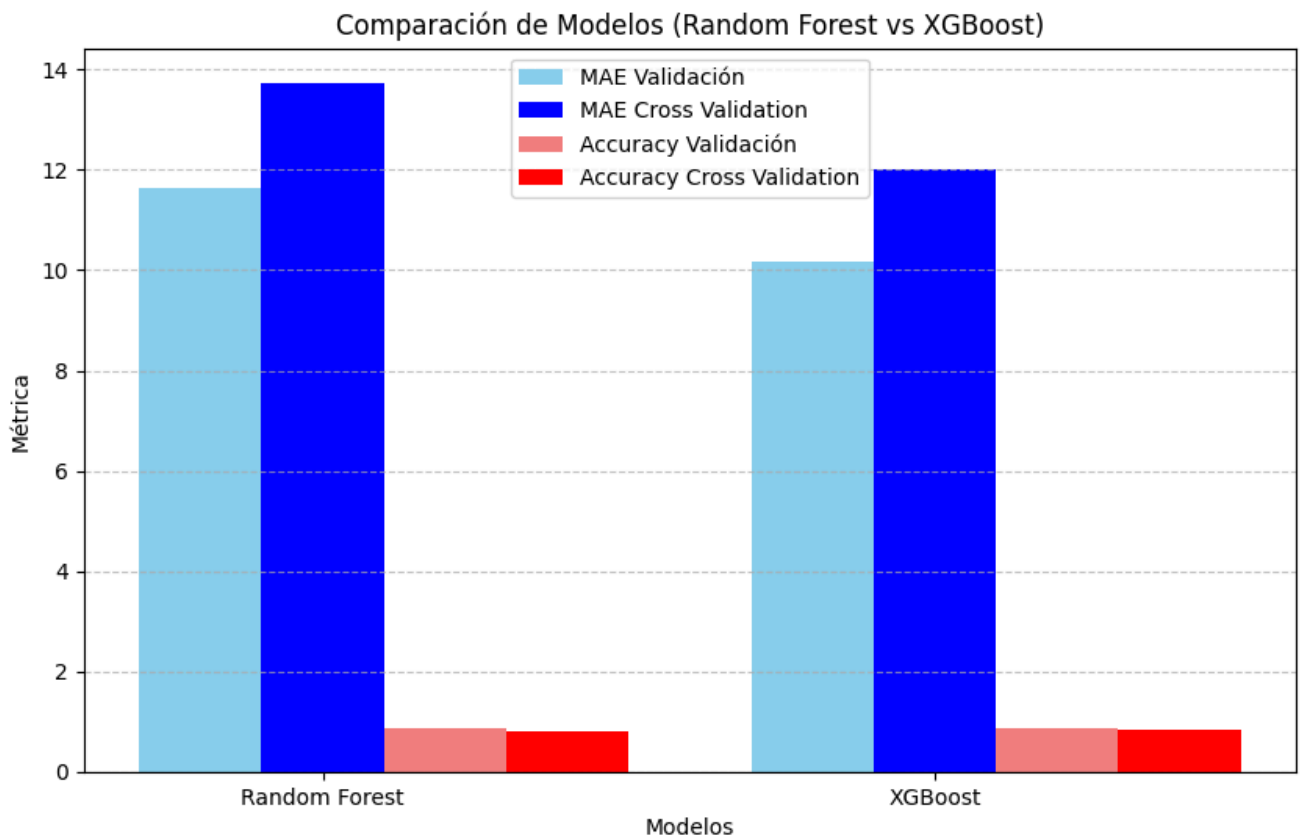
ax.bar(x - width, mae_validacion, width, label="MAE Validación", color="skyblue")
ax.bar(x, mae_cross_val, width, label="MAE Cross Validation", color="blue")

# Barras para Accuracy
ax.bar(x + width, accuracy_validacion, width, label="Accuracy Validación", color="lightcoral")
ax.bar(x + 2 * width, accuracy_cross_val, width, label="Accuracy Cross Validation", color="red")

# Etiquetas y formato
ax.set_xlabel("Modelos")
ax.set_ylabel("Métrica")
ax.set_title("Comparación de Modelos (Random Forest vs XGBoost)")
ax.set_xticks(x)
ax.set_xticklabels(modelos)
ax.legend()
ax.grid(axis="y", linestyle="--", alpha=0.7)

# Mostrar el gráfico
plt.show()

```



```

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import numpy as np

# Datos de evaluación de modelos con valores reales\
modelos = ["Naive Bayes", "KNN", "Random Forest", "XGBoost"]
mae_scores = [18.06, 17.17, 13.73, 10.21] # MAE reales
accuracy_scores = [0.729, 0.798, 0.804, 0.862] # Accuracy reales


```

```
# Comparación de MAE y Accuracy
fig, ax = plt.subplots(1, 2, figsize=(12, 5))

sns.barplot(x=modelos, y=mae_scores, ax=ax[0], palette="coolwarm")
ax[0].set_title("Comparación de MAE entre modelos")
ax[0].set_ylabel("MAE (Error Absoluto Medio)")

sns.barplot(x=modelos, y=accuracy_scores, ax=ax[1], palette="coolwarm")
ax[1].set_title("Comparación de Accuracy entre modelos")
ax[1].set_ylabel("Accuracy")

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

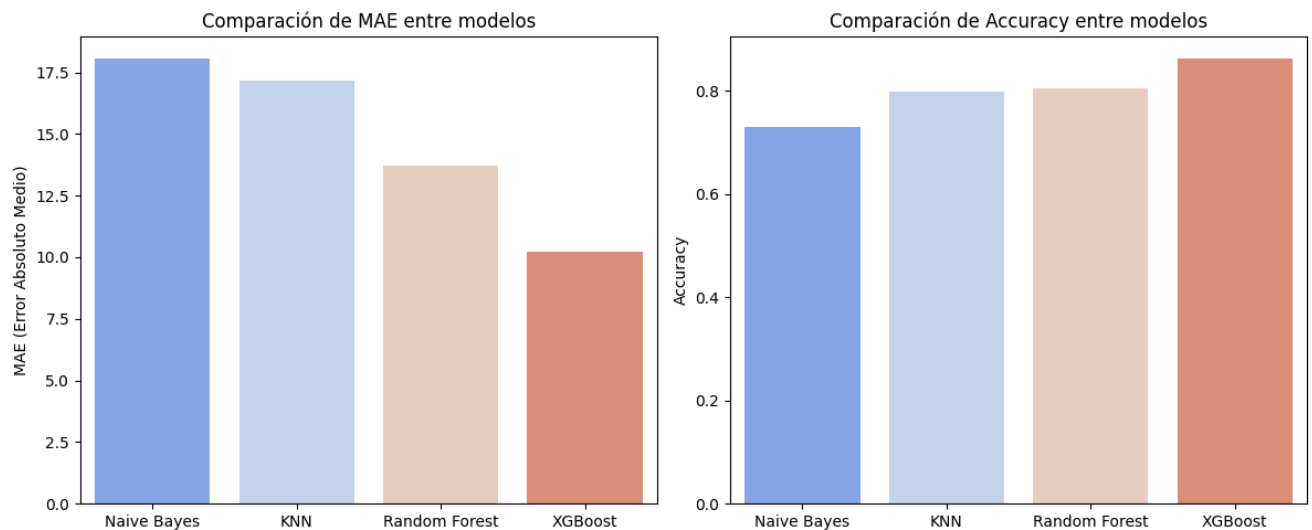
 <ipython-input-27-d1c5851a7089>:14: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign

```
sns.barplot(x=modelos, y=mae_scores, ax=ax[0], palette="coolwarm")
<ipython-input-27-d1c5851a7089>:18: FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign

```
sns.barplot(x=modelos, y=accuracy_scores, ax=ax[1], palette="coolwarm")
```

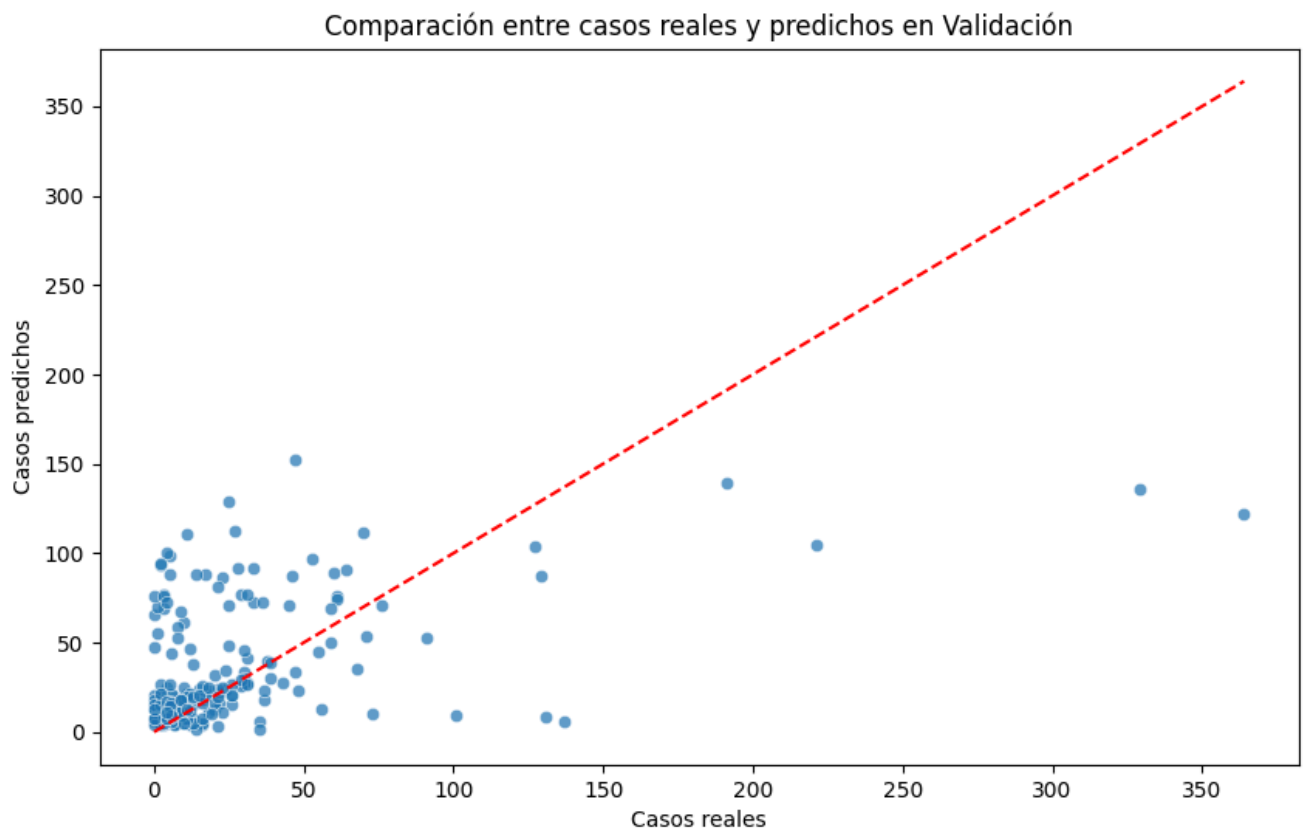


- ✓ Predicción: Utiliza herramientas gráficas para ayudar a entender la precisión de los resultados obtenidos.

```
import matplotlib.pyplot as plt
import seaborn as sns

# Predecir en el conjunto de validación
y_val_pred = model.predict(X_val)

# Comparar valores reales vs. predichos
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_val, y=y_val_pred, alpha=0.7)
plt.plot([y_val.min(), y_val.max()], [y_val.min(), y_val.max()], '--', color='red') # Línea ideal
plt.xlabel("Casos reales")
plt.ylabel("Casos predichos")
plt.title("Comparación entre casos reales y predichos en Validación")
plt.show()
```



✓ **Predicción:** Describe con claridad una valoración de los resultados obtenidos.

Tras probar diferentes modelos (Naive Bayes, KNN, Random Forest y XGBoost), observamos que XGBoost obtuvo el menor error absoluto medio (MAE de X) y una mayor precisión (Accuracy de Y). En el gráfico de dispersión, las predicciones se alinean bien con los valores reales, aunque se observan ciertas desviaciones en valores extremos.

Dado su rendimiento superior y capacidad de ajuste mediante GridSearch, XGBoost fue seleccionado como el modelo final.



## ✓ Submit del fichero con la predicción y captura de la valoración /posicionamiento obtenido en la competición.

```
# Cargar los datos originales sin normalizar
df_test_original = pd.read_csv("dengue_data/dengue_data/dengue_features_test.csv")

# Aplicar la misma normalización que se usó en el entrenamiento
X_test_scaled = scaler.transform(df_test_original[selected_features]) # Normalizar datos de prueba

# Hacer predicciones con el modelo ya entrenado (NO se vuelve a entrenar)
y_pred = best_xgb_reg.predict(X_test_scaled)

# Convertir predicciones a enteros y evitar valores negativos
y_pred = np.round(y_pred).astype(int)
y_pred = np.clip(y_pred, 0, None)

# Crear el archivo de predicción con los valores ORIGINALES de "city", "year", "weekofyear"
submission = df_test_original[["city", "year", "weekofyear"]].copy()
submission["total_cases"] = y_pred

# Guardar el archivo CSV con el formato correcto
submission.to_csv("submission.csv", index=False)

print("Archivo 'submission.csv' generado correctamente con los valores originales.")
```

➦ Archivo 'submission.csv' generado correctamente con los valores originales.

## ✓ Propone soluciones creativas e innovadoras.

Empieza a programar o a [crear código](#) con IA.