PROYECTO FINAL DE INTRODUCCIÓN A LA CIENCIA DE DATOS

Nombre: Juan Torres **Cédula**: E-8-160393 **Grupo**: 1AN213

Materia: Introducción a la Ciencia de Datos **Profesor**: Juan Montenegro

En el año 2020, llegó a Panamá la pandemia del COVID-19, desatando varios casos en todo el país.

El **dataset** nos presenta la cantidad de contagios de COVID-19 en Panamá, registrados a la fecha del 10 de mayo de 2020, en las diferentes provincias, distritos y corregimientos del país. Nos dice cuántos de estos contagios terminaron hospitalizados, en aislamiento domiciliario, UCI, fallecieron o se recuperaron.

Nuestro objetivo es determinar el mejor **modelo de regresión** que nos permita calcular el **número de casos**, basándonos en los datos geográficos como provincia, distrito, corregimiento, latitud y longitud, y también en las condiciones de los pacientes: hospitalizados, UCI, fallecimientos, etc.

1. **Comprensión del problema y del dataset**

Tenemos un total de 13 columnas y 677 filas. Solo las columnas de: **Cantidad, Hospitalizado, Aislamiento Domiciliario, Fallecido, UCI y Recuperado** presentan datos nulos, con un total de 449 filas vacías. Sin embargo, luego de explorar los datos, vemos que la mayor cantidad de datos nulos o faltantes se encuentra en:

- **Provincias: Los Santos**, **Herrera** y **Chiriquí**
- **Distritos: Tonosí (Los Santos)**, **Los Pozos (Herrera)** y **Tolé (Chiriquí)**

Mi pensamiento inicial fue que los datos nulos se debían a que no había reportes en estas áreas, es decir, no había casos.

Sin embargo, analizando nuevamente el dataset, vemos que la ausencia de casos se registra como cero (0). Lo que me llevó a evaluar otras posibilidades.

Considerando que las regiones que presentan mayor cantidad de valores nulos se encuentran en el interior, en áreas de difícil acceso, donde la indumentaria médica tal vez no está tan desarrollada, podríamos decir que los valores nulos en nuestro dataset, se deben a que estos datos directamente no se recopilaron, debido a complicaciones logísticas.

2. **Preparación de los datos**

Siguiendo esta idea, lo mejor sería imputar los datos faltantes, basándonos en datos ya existentes. Por eso, se realizó la imputación utilizando el método de **KNN Imputer**, que nos

permite considerar la similitud entre distritos o corregimientos cercanos. De esta manera, evitamos descartar datos que podrían ser significativos para el análisis.

3. Feature Engineering

Con el objetivo de ayudar a mejorar el modelo predictivo, se agregaron al dataset (con los datos imputados), las siguientes variables:

a) Ratios entre casos

a. **Hospitalizados vs Fallecidos (H/F):** Nos indica la cantidad de personas hospitalizadas que terminan falleciendo. Nos permite entender la **gravedad de la enfermedad**. En este caso, entre más alto el valor, más severo es el COVID-19. Un aumento en el número de casos puede llevar a un aumento en la hospitalización, y potencialmente a un aumento en los fallecimientos.

b. **Recuperados vs Aislados (R/A):** Nos da la proporción de personas que se recuperaron, con relación a los que están aislados. Nos ayuda a determinar qué tan eficaces son las **medidas de aislamiento** en la **recuperación de los pacientes**. Un aumento en los casos afectará directamente el número de aislados y posiblemente, el de recuperados.

c. **Hospitalizados vs Recuperados (H/R):** La proporción de **personas hospitalizadas que se han recuperado**. Nos indica la efectividad del tratamiento hospitalario. Un aumento en los casos puede llevar a un aumento en las hospitalizaciones, pero si el tratamiento es efectivo, también puede significar un aumento en las recuperaciones.

b) Relación entre las coordenadas geográficas (LONG, LAT) y la Cantidad de Casos

a. **Distancia Cantidad**: Es la relación que existe entre las coordenadas geográficas y la cantidad de casos.

Para determinar la relación entre nuestras características actuales, de manera que pudiéramos elegir las más indicadas para mejorar nuestro modelo predictivo, se elaboró un **mapa de calor**.

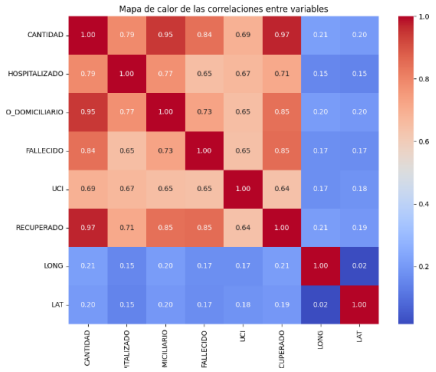

***Figura 1**: Mapa de calor de las correlaciones entre variables*

4. División del Dataset

Nuestro objetivo es entrenar modelos que nos ayuden a determinar la cantidad de casos, basándonos en las características geográficas, los datos registrados en el dataset y las variables nuevas que se crearon en el punto anterior. Para ello, nos enfocamos en dos provincias: **Bocas del Toro** y **Chiriquí**.

4.1. Datasets Nuevos, para Bocas del Toro y Chiriquí

Lo primero que se realizó fue separar nuestros datos en dos datasets individuales, que contuvieran únicamente los datos para cada provincia respectivamente.

4.2. Transformación a Características Numéricas

Para poder considerar las columnas de **provincia**, **distrito** y **corregimiento** como parte de nuestro modelo, fue necesario transformar sus datos a numéricos. Para esto, aplicamos ***One-Hot Encoding***, que transformó los datos de estas columnas a **variables dummies**, con valores numéricos (0 y 1).

5. Modelado y Validación Cruzada

5.1. Entrenamiento de los Modelos

Primero, definimos nuestra **variable objetivo** (**cantidad**) y las **variables independientes**. Para así poder dividir nuestros datos en un **conjunto de entrenamiento**, que abarca el 80% de los datos, y un **conjunto de prueba**, con el otro el 20%.

- Se tomó la decisión de no incluir la variable de: **GlobalID** en el estudio

Los modelos que utilizamos fueron los siguientes:

a) Modelo de Regresión Lineal

Este modelo busca describir la relación lineal entre una **variable dependiente** y una o varias **variables independientes**.

b) Modelo de Random Forest

Este modelo de aprendizaje supervisado que combina el resultado de múltiples árboles de decisión para llegar a un resultado único. Cada árbol se entrena con un conjunto de datos ligeramente diferente.

Es muy útil para describir las relaciones no lineales, pues de por sí los árboles de decisión trabajan de forma no lineal.

Cada árbol de decisión es capaz de identificar patrones complejos en los datos. Por eso, al entregarnos el resultado de varios árboles de decisión, este modelo puede representar el comportamiento de los datos de manera más precisa.

5.2. Validación K-Folds

Para evitar el sobreajuste de cada uno de nuestros modelos, implementamos la **validación cruzada de k-folds.** Esta técnica, nos ayuda a determinar cómo nuestro modelo se comporta con datos desconocidos.

La validación k-folds dividió nuestros datos en k subconjuntos (folds), en este caso se indicó que fueran cinco (k=5). Se entrena el modelo en cuatro folds y se prueba en el último, el que no se usó para entrenamiento. Este proceso se repite cinco veces.

De esta manera, nos aseguramos de que el modelo no se ajuste únicamente a un solo conjunto de datos. Sino que aprenda patrones generales, en lugar de solo aprender a partir de los datos de entrenamiento.

Después de entrenar el modelo en cada fold, se calcularon tres métricas de error: **MAE** (Error Absoluto Medio), **MSE** (Error Cuadrático Medio) y **RMSE** (Raíz del Error Cuadrático Medio), para saber qué tan bien trabajó el modelo.

Por medio de estas métricas podemos obtener una idea de cómo podría comportarse el modelo al trabajar con datos nuevos. Pues evalúan la precisión del modelo, al determinar qué tanto varían las predicciones del modelo con respecto a los valores reales.

Al terminar los cinco ciclos de entrenamiento y prueba, calculamos el promedio de los errores y los imprimimos. Esto nos dice, en general, cómo está funcionando el modelo.

Este proceso de validación k-folds, lo aplicamos tanto para el **modelo de regresión lineal**, como el **modelo de random forest**.

5.3. Modelos Finales

Luego de finalizar la validación, entrenamos y probamos nuestros modelos con los datos de prueba y entrenamiento que nosotros habíamos separado.

Para probarlos, se calcularon las mismas métricas de error, pero con los resultados obtenidos luego de trabajar con los datos de prueba. Estas nos dicen cómo se comportan los modelos con datos nuevos, totalmente desconocidos.

Adicionalmente, también calculamos el **coeficiente de determinación (R^2)**. Este nos dice qué tan bien se ajusta nuestro modelo de regresión a los datos, en este caso, los de prueba.

Todo esto se realizó para los datos de la provincia de **Bocas del Toro** y **Chiriquí**.

6. Evaluación de los Modelos

6.1. Resultados

a. Bocas del Toro

***Tabla 1**: Resultados de las métricas de error y coeficiente de determinación para los modelos de **Regresión Lineal** y **Random Forest**, para la provincia de **Bocas del Toro**.*

Modelo	MAE	MSE	RMSE	R^2
Random Forest	1.235	9.15135	3.0251	0.943002
Regresión Lineal	0.2096	0.2111	0.4594	0.998685

b. Chiriquí

***Tabla 2**: Resultados de las métricas de error y coeficiente de determinación para los modelos de **Regresión Lineal** y **Random Forest**, para la provincia de **Chiriquí**.*

Modelo	MAE	MSE	RMSE	R^2
Random Forest	0.02238	0.007128	0.08443	0.999963
Regresión Lineal	0.063390	0.007203	0.084869	0.999963

6.2. Análisis de Resultados

a. Precisión (MAE, MSE, RMSE)

Para la provincia de **Bocas del Toro**, Los valores de las **métricas de los errores (MAE, MSE y RSME)** para el modelo de **regresión lineal** son **mucho más bajos** que los de las métricas del **modelo de random forest**. Esto es un indicativo de que la **precisión de este modelo es mayor**.

Por el contrario, vemos que para la provincia de **Chiriquí**, los valores de las métricas de los errores para ambos modelos **son bastante similares en general**, siendo **ligeramente más bajas** las métricas del **modelo random forest**.

b. Generalización (R^2)

Los **coeficientes de determinación** para ambos modelos en ambas provincias son bastante altos, con valores por arriba del 0.9, lo que indica una excelente capacidad de generalización por parte de ambos.

Vale la pena mencionar que, para provincia de Bocas del Toro, el coeficiente del modelo de regresión lineal es un poco más alto, lo que indicaría que, para estos datos, este modelo generaliza un poco mejor.

6.3. Balance entre Presición y Generalización

Para la provincia de **Bocas del Toro,** el modelo de regresión lineal arrojó los mejores resultados.

Y aunque para la provincia de **Chiriquí**, el modelo de random forest tuvo mejores resultados, por la mínima diferencia, los valores del modelo de regresión lineal para esta provincia tuvieron valores excelentes.

En conclusión, podríamos decir, en base a estos resultados, que el **modelo de regresión lineal** logra un **mejor rendimiento** y ofrece un mejor balance entre precisión y generalización.

7. CONCLUSIONES

Entrenamos y modelamos dos modelos de regresión diferentes: regresión lineal y **random forest**, para determinar cuál sería el más efectivo al momento de predecir la cantidad de casos de COVID-19, basándonos e los datos recolectados hasta mayo de 2020.

Inicialmente, nuestro dataset estaba incompleto, por lo que, para poder trabajar con ambos modelos de regresión, fue necesario limpiarlo e imputar los datos faltantes. También generamos nuevas variables que ayudaran a mejorar nuestros modelos predictivos.

Luego, evaluamos el rendimiento de ambos modelos con el mismo conjunto de datos, utilizando las variables que seleccionamos para el estudio.

Los resultados nos indican que el modelo que tuvo un mejor rendimiento, logrando un mejor balance entre precisión y generalización, fue el **modelo de regresión lineal**, lo cual sugiere que entre mi variable objetivo y nuestras variables de estudio existe una relación lineal.

Sin embargo, el modelo de regresión random forest, el cual normalmente se usa para relaciones no lineales, tuvo un buen rendimiento también. Esto podría indicar que existen ciertos componentes no lineales o ruido en los datos, pero no lo suficiente para que el modelo de random forest nos ofrezca una mejora significativa sobre el de regresión lineal.

Ahora, también vemos que la generalización por parte de ambos modelos fue bastante alta, sobre todo al trabajar con el modelo lineal. Lo cual considero un buen resultado, considerando que estábamos trabajando con modelos muy complejos con muchas variables, y también con un gran número de datos imputados.

Esto fue gracias a que implementamos la validación cruzada. Sin embargo, seguimos corriendo el riesgo de sobreajuste.

Por eso considero que una buena opción para un estudio futuro podría ser trabajar con un modelo de regresión de Lasso, para evaluar cuales de nuestras variables son realmente necesarias para el estudio, lo cual nos ayudaría a trabajar con un modelo más simple. O en su defecto, si queremos conservar todas nuestras variables, podríamos implementar Ridge.

ANEXO: **CÓDIGO**

```
#1.	Comprensión del problema y del dataset

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.impute import KNNImputer

from sklearn.impute import SimpleImputer

from sklearn.model_selection import train_test_split, cross_val_score, KFold

from sklearn.linear_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

data = pd.read_csv('C:/Users/HP/OneDrive - Universidad Tecnológica de Panamá/MAESTRIA EN ANALITICA DE DATOS/INTRODUCCION A CIENCIA DE DATOS/PROYECTO FINAL INTRODUCCION A CIENCIA DE DATOS/Dataset Covid19 Panama 10-05-2020.csv')

print(data.head())

print(data.duplicated().sum())

#Parece no haber valores duplicados

print(data.shape)


#En total tengo 677 filas y 13 columnas

print(data.info())

print(data.isnull().sum())

#En total estamos trabajando con 449 datos faltantes en 7 columnas

# Total de datos no nulos por provincia

total_no_nulos_por_provincia = data.groupby('PROVINCIA').count()


print(total_no_nulos_por_provincia)
```

```
#Las provincias con mayor cantidad de valores nulos son: Los Santos, Veraguas y Chiriquí

# Total de datos no nulos por provincia

total_no_nulos_por_distrito = data.groupby('DISTRITO').count()


print(total_no_nulos_por_distrito)

# Total de datos no nulos por provincia

total_no_nulos_por_corregimiento = data.groupby('CORREGIMIENTO').count()


print(total_no_nulos_por_corregimiento)


print(data.isna().sum())


# Columnas con valores faltantes para imputar

columnas_a_imputar = ['CANTIDAD', 'HOSPITALIZADO', 'AISLAMIENTO_DOMICILIARIO', 'FALLECIDO', 'UCI', 'RECUPERADO']


# Seleccionamos las columnas numéricas

data_imputar = data[columnas_a_imputar]


# Imputador KNN

imputer = KNNImputer(n_neighbors=5)


# Aplicamos el KNNImputer a las columnas

data_imputada = imputer.fit_transform(data_imputar)


# Redondeamos los valores imputados
```

```
data_imputada = np.round(data_imputada).astype(int)

# Creamos un Dataframe para las columnas imputadas
data_imputada_df = pd.DataFrame(data_imputada, columns=columnas_a_imputar)
data[columnas_a_imputar] = data_imputada_df

# Cantidad de Valores Nulos después de la Imputación
print(data.isnull().sum())

# Primeras filas del dataset
print(data.head())

# Guardar el DataFrame imputado en un archivo CSV
data.to_csv('data_imputada.csv', index=False)

#3. Feature Engineering

#Mapa de Calor

import seaborn as sns

df = pd.read_csv('C:/Users/HP/OneDrive - Universidad Tecnológica de Panamá/MAESTRIA EN ANALITICA DE DATOS/INTRODUCCION A CIENCIA DE DATOS/PROYECTO FINAL INTRODUCCION A CIENCIA DE DATOS/data_imputada.csv')

selected_columns = ['CANTIDAD', 'HOSPITALIZADO', 'AISLAMIENTO_DOMICILIARIO', 'FALLECIDO', 'UCI', 'RECUPERADO', 'LONG', 'LAT']
```

```
#Matriz de Correlación
corr_matrix = df[selected_columns].corr()

#Mapa de Calor

plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Mapa de calor de las correlaciones entre variables')

plt.savefig('Mapa de calor de las correlaciones entre variables.png')
plt.show()

#Generación de nuevas características

# Como algunas características tienen valores de cero, añadimos un pequeño valor para evitar dividir entre cero.
#Este valor no nos afectará significativamente los resultados
epsilon = 1e-6

#1. Ratios entre los casos:

# Hospitalizados vs Fallecidos
df['ratio_hospitalizados_fallecidos'] = df['HOSPITALIZADO'] / (df['FALLECIDO'] + epsilon)

# Recuperados vs Aislados
```

```
df['ratio_recuperados_aislados'] = df['RECUPERADO'] / (df['AISLAMIENTO_DOMICILIARIO'] + epsilon)


# Hospitalizados vs Recuperados

df['ratio_hospitalizados_recuperados'] = df['HOSPITALIZADO'] / (df['RECUPERADO'] + epsilon)


#2. Relación entre las coordenadas geográficas (LONG, LAT) y la Cantidad de Casos.


# Relación entre las coordenadas geográficas y la cantidad de casos

df["DISTANCIA_CANTIDAD"] = (df['LONG'] ** 2 + df['LAT'] ** 2) ** 0.5 * df['CANTIDAD']


# Mostrar las primeras filas del dataset con las nuevas características

df.head()


#Guardamos el dataset con las nuevas características


df.to_csv('dataset con nuevas caracteristicas agregadas.csv', index=False)


#Generamos un Dataset para Bocas del Toro

dataframe = pd.read_csv('C:/Users/HP/OneDrive - Universidad Tecnológica de Panamá/MAESTRIA EN ANALITICA DE DATOS/INTRODUCCION A CIENCIA DE DATOS/PROYECTO FINAL INTRODUCCION A CIENCIA DE DATOS/dataset con nuevas caracteristicas agregadas.csv')

bocas_del_toro = dataframe[dataframe['PROVINCIA'] == 'BOCAS DEL TORO']


bocas_del_toro.to_csv('dataset_bocas_del_toro.csv', index=False)
```

```
#Generamos un Dataset para Chiriquí

dataframe = pd.read_csv('C:/Users/HP/OneDrive - Universidad Tecnológica de Panamá/MAESTRIA EN ANALITICA DE DATOS/INTRODUCCION A CIENCIA DE DATOS/PROYECTO FINAL INTRODUCCION A CIENCIA DE DATOS/dataset con nuevas caracteristicas agregadas.csv')

chiriqui = dataframe[dataframe['PROVINCIA'] == 'CHIRIQUÍ']


chiriqui.to_csv('dataset_chiriqui.csv', index=False)


#BOCAS DEL TORO

#PARA PODER CONSIDERAR LAS COLUMNAS DE PROVINCIA, DISTRITO Y CORREGIMIENTO

#COMO PARTE DE NUESTRO MODELO, DEBEMOS TRANSFORMAR SUS DATOS A NUMÉRICOS

dfbt = pd.read_csv('C:/Users/HP/OneDrive - Universidad Tecnológica de Panamá/MAESTRIA EN ANALITICA DE DATOS/INTRODUCCION A CIENCIA DE DATOS/PROYECTO FINAL INTRODUCCION A CIENCIA DE DATOS/dataset_bocas_del_toro.csv')


# Aplicamos One-Hot Encoding a las columnas categóricas 'PROVINCIA', 'DISTRITO', y 'CORREGIMIENTO'

df_dummies = pd.get_dummies(dfbt, columns=['PROVINCIA', 'DISTRITO', 'CORREGIMIENTO'])


# Guardar el nuevo dataframe en un archivo CSV

df_dummies.to_csv('Dataset Bocas del Toro con Columnas Numéricas.csv', index=False)


# Mostrar las primeras filas del dataframe procesado (opcional)

print(df_dummies.head())


#ENTRENAMIENTO DEL MODELO PARA BOCAS DEL TORO
```

```
# Traemos el dataset para realizar el entrenamiento

df_BT = pd.read_csv('C:/Users/HP/OneDrive - Universidad Tecnológica de Panamá/MAESTRIA EN ANALITICA DE DATOS/INTRODUCCION A CIENCIA DE DATOS/PROYECTO FINAL INTRODUCCION A CIENCIA DE DATOS/Dataset Bocas del Toro con Columnas Numéricas.csv')


# DEFINIMOS NUESTRAS VARIABLES INDEPENDIENTES (X) Y NUESTRA VARIABLE OBJETIVO (Y)

X = df_BT.drop(['CANTIDAD','GlobalID'], axis=1)  # features (características)

y = df_BT['CANTIDAD']  # target (variable objetivo)


# Dividir los datos en conjunto de entrenamiento y prueba

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


# Modelo de Regresión Lineal

model = LinearRegression()


# VALIDACIÓN CRUZADA de 5-FOLDS

k = 5

kf = KFold(n_splits=k, shuffle=True, random_state=42)


mae_scores = []

mse_scores = []

rmse_scores = []
```

```
#Se dividen los datos en 5 folds. Se usan 4 para entrenar el modelo, y uno para probarlo. Repetimos el proceso 5 veces
for train_index, val_index in kf.split(X_train):

    X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[val_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]



    model.fit(X_train_fold, y_train_fold)



    y_val_pred = model.predict(X_val_fold)

    # Métricas de error después de cada fold
    MAE = mean_absolute_error(y_val_fold, y_val_pred)
    MSE = mean_squared_error(y_val_fold, y_val_pred)
    RMSE = np.sqrt(MSE)



    mae_scores.append(MAE)
    mse_scores.append(MSE)
    rmse_scores.append(RMSE)

#Resultados de la validación cruzada
mae_scores = np.array(mae_scores)
mse_scores = np.array(mse_scores)
```

```
rmse_scores = np.array(rmse_scores)



print("MAE Scores: ", mae_scores)

print("MAE Promedio: %0.2f (+/- %0.2f)" % (mae_scores.mean(), mae_scores.std()))

print("MSE Scores: ", mse_scores)

print("MSE Promedio: %0.2f (+/- %0.2f)" % (mse_scores.mean(), mse_scores.std()))

print("RMSE Scores: ", rmse_scores)

print("RMSE Promedio: %0.2f (+/- %0.2f)" % (rmse_scores.mean(), rmse_scores.std()))


# Entrenamos el modelo final de regresión lineal con los datos de entrenamiento y prueba que dividimos nosotros

model.fit(X_train, y_train)


#Predicciones para el conjunto de prueba del modelo de regresión lineal

y_pred = model.predict(X_test)


# Métricas para el conjunto de prueba

MAE_test = mean_absolute_error(y_test, y_pred)

MSE_test = mean_squared_error(y_test, y_pred)

RMSE_test = np.sqrt(MSE_test)

R2_test = r2_score(y_test, y_pred)


# Resultados finales del modelo

print("Intercept: %f, Coeficiente(s): %s" % (model.intercept_, model.coef_))

print("MAE en el conjunto de prueba: %f" % MAE_test)
```

```
print("MSE en el conjunto de prueba: %f" % MSE_test)

print("RMSE en el conjunto de prueba: %f" % RMSE_test)

print("R² en el conjunto de prueba: %f" % R2_test)

# Modelo de Random Forest Regressor

random_forest_model = RandomForestRegressor(n_estimators=100, random_state=42)


# VALIDACIÓN CRUZADA de 5-FOLDS

k = 5

kf = KFold(n_splits=k, shuffle=True, random_state=42)



mae_scores_rf = []

mse_scores_rf = []

rmse_scores_rf = []


#Se dividen los datos en 5 folds. Se usan 4 para entrenar el modelo, y uno para probarlo. Repetimos el proceso 5 veces


for train_index, val_index in kf.split(X_train):


    X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[val_index]

    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]



    random_forest_model.fit(X_train_fold, y_train_fold)
```

```
    y_val_pred_rf = random_forest_model.predict(X_val_fold)

    # Métricas de error después de cada fold
    mae_val_rf = mean_absolute_error(y_val_fold, y_val_pred_rf)
    mse_val_rf = mean_squared_error(y_val_fold, y_val_pred_rf)
    rmse_val_rf = np.sqrt(mse_val_rf)



    mae_scores_rf.append(mae_val_rf)
    mse_scores_rf.append(mse_val_rf)
    rmse_scores_rf.append(rmse_val_rf)

#Resultados de la validación cruzada
mae_scores_rf = np.array(mae_scores_rf)
mse_scores_rf = np.array(mse_scores_rf)
rmse_scores_rf = np.array(rmse_scores_rf)



print("MAE Scores por fold: ", mae_scores_rf)
print("MAE Promedio: %0.2f (+/- %0.2f)" % (mae_scores_rf.mean(), mae_scores_rf.std()))
print("MSE Scores por fold: ", mse_scores_rf)
print("MSE Promedio: %0.2f (+/- %0.2f)" % (mse_scores_rf.mean(), mse_scores_rf.std()))
print("RMSE Scores por fold: ", rmse_scores_rf)
print("RMSE Promedio: %0.2f (+/- %0.2f)" % (rmse_scores_rf.mean(), rmse_scores_rf.std()))
```

```
# Entrenamos el modelo final de random forest con los datos de entrenamiento y prueba que dividimos nosotros

random_forest_model.fit(X_train, y_train)

#Predicciones para el conjunto de prueba del modelo de random forest
y_pred_rf = random_forest_model.predict(X_test)

# Métricas para el conjunto de prueba
mae_rf_test = mean_absolute_error(y_test, y_pred_rf)
mse_rf_test = mean_squared_error(y_test, y_pred_rf)
rmse_rf_test = np.sqrt(mse_rf_test)
r2_rf_test = r2_score(y_test, y_pred_rf)

# Resultados finales del modelo
print(f"MAE (Random Forest en conjunto de prueba): {mae_rf_test}")
print(f"MSE (Random Forest en conjunto de prueba): {mse_rf_test}")
print(f"RMSE (Random Forest en conjunto de prueba): {rmse_rf_test}")
print(f"R² (Random Forest en conjunto de prueba): {r2_rf_test}")

#CHIRIQUI
#PARA PODER CONSIDERAR LAS COLUMNAS DE PROVINCIA, DISTRITO Y CORREGIMIENTO
#COMO PARTE DE NUESTRO MODELO, DEBEMOS TRANSFORMAR SUS DATOS A NUMÉRICOS
dfbt = pd.read_csv('C:/Users/HP/OneDrive - Universidad Tecnológica de Panamá/MAESTRIA EN ANALITICA DE DATOS/INTRODUCCION A CIENCIA DE DATOS/PROYECTO FINAL INTRODUCCION A CIENCIA DE DATOS/dataset_chiriqui.csv')
```

```
# Aplicamos One-Hot Encoding a las columnas categóricas 'PROVINCIA', 'DISTRITO', y 'CORREGIMIENTO'
df_dummies = pd.get_dummies(dfbt, columns=['PROVINCIA', 'DISTRITO', 'CORREGIMIENTO'])

# Guardar el nuevo dataframe en un archivo CSV
df_dummies.to_csv('Dataset Chiriqui con Columnas Numéricas.csv', index=False)

# Mostrar las primeras filas del dataframe procesado (opcional)
print(df_dummies.head())

#ENTRENAMIENTO DEL MODELO PARA CHIRIQUÍ


# Traemos el dataset para realizar el entrenamiento
df_BT = pd.read_csv('C:/Users/HP/OneDrive - Universidad Tecnológica de Panamá/MAESTRIA EN ANALITICA DE DATOS/INTRODUCCION A CIENCIA DE DATOS/PROYECTO FINAL INTRODUCCION A CIENCIA DE DATOS/Dataset Chiriqui con Columnas Numéricas.csv')

# DEFINIMOS NUESTRAS VARIABLES INDEPENDIENTES (X) Y NUESTRA VARIABLE OBJETIVO (Y)
X = df_BT.drop(['CANTIDAD','GlobalID'], axis=1)  # features (características)
y = df_BT['CANTIDAD']  # target (variable objetivo)

# Dividir los datos en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Modelo de Regresión Lineal
model = LinearRegression()

# VALIDACIÓN CRUZADA de 5-FOLDS
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=42)

mae_scores = []
mse_scores = []
rmse_scores = []

#Se dividen los datos en 5 folds. Se usan 4 para entrenar el modelo, y uno para probarlo. Repetimos el proceso 5 veces

for train_index, val_index in kf.split(X_train):

    X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[val_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]


    model.fit(X_train_fold, y_train_fold)


    y_val_pred = model.predict(X_val_fold)

    # Métricas de error después de cada fold
```

```
    MAE = mean_absolute_error(y_val_fold, y_val_pred)
    MSE = mean_squared_error(y_val_fold, y_val_pred)
    RMSE = np.sqrt(MSE)


    mae_scores.append(MAE)
    mse_scores.append(MSE)
    rmse_scores.append(RMSE)

#Resultados de la validación cruzada
mae_scores = np.array(mae_scores)
mse_scores = np.array(mse_scores)
rmse_scores = np.array(rmse_scores)


print("MAE Scores: ", mae_scores)
print("MAE Promedio: %0.2f (+/- %0.2f)" % (mae_scores.mean(), mae_scores.std()))
print("MSE Scores: ", mse_scores)
print("MSE Promedio: %0.2f (+/- %0.2f)" % (mse_scores.mean(), mse_scores.std()))
print("RMSE Scores: ", rmse_scores)
print("RMSE Promedio: %0.2f (+/- %0.2f)" % (rmse_scores.mean(), rmse_scores.std()))

# Entrenamos el modelo final de regresión lineal con los datos de entrenamiento y prueba que dividimos nosotros

model.fit(X_train, y_train)
```

```
#Predicciones para el conjunto de prueba del modelo de regresión lineal

y_pred = model.predict(X_test)

# Métricas para el conjunto de prueba

MAE_test = mean_absolute_error(y_test, y_pred)
MSE_test = mean_squared_error(y_test, y_pred)
RMSE_test = np.sqrt(MSE_test)
R2_test = r2_score(y_test, y_pred)

# Resultados finales del modelo

print("Intercept: %f, Coeficiente(s): %s" % (model.intercept_, model.coef_))
print("MAE en el conjunto de prueba: %f" % MAE_test)
print("MSE en el conjunto de prueba: %f" % MSE_test)
print("RMSE en el conjunto de prueba: %f" % RMSE_test)
print("R² en el conjunto de prueba: %f" % R2_test)

# Modelo de Regresión Lineal
model = LinearRegression()

# VALIDACIÓN CRUZADA de 5-FOLDS
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=42)
```

```
mae_scores = []
mse_scores = []
rmse_scores = []

#Se dividen los datos en 5 folds. Se usan 4 para entrenar el modelo, y uno para probarlo. Repetimos el proceso 5 veces

for train_index, val_index in kf.split(X_train):

    X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[val_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]


    model.fit(X_train_fold, y_train_fold)


    y_val_pred = model.predict(X_val_fold)

    # Métricas de error después de cada fold
    MAE = mean_absolute_error(y_val_fold, y_val_pred)
    MSE = mean_squared_error(y_val_fold, y_val_pred)
    RMSE = np.sqrt(MSE)


    mae_scores.append(MAE)
```

```
    mse_scores.append(MSE)
    rmse_scores.append(RMSE)

#Resultados de la validación cruzada
mae_scores = np.array(mae_scores)
mse_scores = np.array(mse_scores)
rmse_scores = np.array(rmse_scores)


print("MAE Scores: ", mae_scores)
print("MAE Promedio: %0.2f (+/- %0.2f)" % (mae_scores.mean(), mae_scores.std()))
print("MSE Scores: ", mse_scores)
print("MSE Promedio: %0.2f (+/- %0.2f)" % (mse_scores.mean(), mse_scores.std()))
print("RMSE Scores: ", rmse_scores)
print("RMSE Promedio: %0.2f (+/- %0.2f)" % (rmse_scores.mean(), rmse_scores.std()))

# Entrenamos el modelo final de regresión lineal con los datos de entrenamiento y prueba
que dividimos nosotros

model.fit(X_train, y_train)

#Predicciones para el conjunto de prueba del modelo de regresión lineal

y_pred = model.predict(X_test)

# Métricas para el conjunto de prueba
```

```
MAE_test = mean_absolute_error(y_test, y_pred)

MSE_test = mean_squared_error(y_test, y_pred)

RMSE_test = np.sqrt(MSE_test)

R2_test = r2_score(y_test, y_pred)


# Resultados finales del modelo


print("Intercept: %f, Coeficiente(s): %s" % (model.intercept_, model.coef_))

print("MAE en el conjunto de prueba: %f" % MAE_test)

print("MSE en el conjunto de prueba: %f" % MSE_test)

print("RMSE en el conjunto de prueba: %f" % RMSE_test)

print("R² en el conjunto de prueba: %f" % R2_test)


# Modelo de Random Forest Regressor

random_forest_model = RandomForestRegressor(n_estimators=100, random_state=42)


# VALIDACIÓN CRUZADA de 5-FOLDS

k = 5

kf = KFold(n_splits=k, shuffle=True, random_state=42)



mae_scores_rf = []

mse_scores_rf = []

rmse_scores_rf = []
```

```
#Se dividen los datos en 5 folds. Se usan 4 para entrenar el modelo, y uno para probarlo. Repetimos el proceso 5 veces

for train_index, val_index in kf.split(X_train):

    X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[val_index]
    y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index]


    random_forest_model.fit(X_train_fold, y_train_fold)


    y_val_pred_rf = random_forest_model.predict(X_val_fold)

    # Métricas de error después de cada fold
    mae_val_rf = mean_absolute_error(y_val_fold, y_val_pred_rf)
    mse_val_rf = mean_squared_error(y_val_fold, y_val_pred_rf)
    rmse_val_rf = np.sqrt(mse_val_rf)


    mae_scores_rf.append(mae_val_rf)
    mse_scores_rf.append(mse_val_rf)
    rmse_scores_rf.append(rmse_val_rf)

#Resultados de la validación cruzada
mae_scores_rf = np.array(mae_scores_rf)
```

```
mse_scores_rf = np.array(mse_scores_rf)
rmse_scores_rf = np.array(rmse_scores_rf)



print("MAE Scores por fold: ", mae_scores_rf)
print("MAE Promedio: %0.2f (+/- %0.2f)" % (mae_scores_rf.mean(), mae_scores_rf.std()))
print("MSE Scores por fold: ", mse_scores_rf)
print("MSE Promedio: %0.2f (+/- %0.2f)" % (mse_scores_rf.mean(), mse_scores_rf.std()))
print("RMSE Scores por fold: ", rmse_scores_rf)
print("RMSE Promedio: %0.2f (+/- %0.2f)" % (rmse_scores_rf.mean(), rmse_scores_rf.std()))

# Entrenamos el modelo final de random forest con los datos de entrenamiento y prueba
que dividimos nosotros

random_forest_model.fit(X_train, y_train)

#Predicciones para el conjunto de prueba del modelo de random forest

y_pred_rf = random_forest_model.predict(X_test)

# Métricas para el conjunto de prueba

mae_rf_test = mean_absolute_error(y_test, y_pred_rf)
mse_rf_test = mean_squared_error(y_test, y_pred_rf)
rmse_rf_test = np.sqrt(mse_rf_test)
r2_rf_test = r2_score(y_test, y_pred_rf)
```

```
#Resultados finales del modelo
print(f"MAE (Random Forest en conjunto de prueba): {mae_rf_test}")
print(f"MSE (Random Forest en conjunto de prueba): {mse_rf_test}")
print(f"RMSE (Random Forest en conjunto de prueba): {rmse_rf_test}")
print(f"R² (Random Forest en conjunto de prueba): {r2_rf_test}")
```