FX_Predictions

April 30, 2024

1 FX Predictions

Creado por:

- **V. D. Betancourt**

1.1 Objetivo

El presente proyecto permite descargar la *evolución histórica* de los siguientes **Tipos de Cambio (Paridades)** usando la librería de Yahoo Finance (`'yfinance'`):

- `"USDMXN"`
- `"EURMXN"`
- `"GBPMXN"`
- `"EURUSD"`
- `"GBPUSD"`

Y crea un **Modelo de Redes Neuronales** para generar sus respectivas *predicciones*.

1.2 Settings

1.2.1 Entorno de Ejecución

El presente proyecto se ha ejecutado en Google Colab usando el siguiente **Entorno de Ejecución**:

- **T4 GPU**

1.2.2 Importar Librerías

[1]:
```
# Importar
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
import yfinance as yf
```

1.2.3 Definir Paridades de Monedas

[2]:
```
# Lista de paridades de monedas
currency_pairs = ["USDMXN=X", "EURMXN=X", "GBPMXN=X", "EURUSD=X", "GBPUSD=X"]
```

1.2.4 Función para Descargar Datos

[3]:
```
# Descargar datos históricos de las paridades de monedas
def download_data(currency_pairs):
    data = {}
    for pair in currency_pairs:
        data[pair] = yf.download(pair, start="2015-01-01", end="2024-04-30")
    return data
```

1.2.5 Función para Graficar Datos Históricos

Matplotlib

[4]:
```
# Dibujar la evolución histórica de las paridades de monedas
def plot_data(data):
    for pair, df in data.items():
        plt.figure(figsize=(14,8))
        sns.lineplot(data=df['Close'])
        plt.title(f'Evolución histórica de {pair}')
        plt.show()
```

Seaborn

[5]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import yfinance as yf

# Función para dibujar evolución histórica de paridades de monedas
def plot_data_sns(data, titles=None, font_scale=1.2, color='blue', figsize=(14,␣
 ↪8)):
    # Establecer el estilo de seaborn y escala de fuentes
    sns.set(style='white', font_scale=font_scale)

    # Número de subplots necesarios
    n_pairs = len(data)

    # Crear una figura con subplots
    fig, axes = plt.subplots(n_pairs, 1, figsize=figsize, squeeze=False)
    axes = axes.flatten()  # Esto maneja el caso de un solo subplot␣
 ↪automáticamente

    # Iterar sobre los pares y sus respectivos DataFrames
```

```
    for ax, (pair, df) in zip(axes, data.items()):
        # Trazar la línea con seaborn, enfocándose en la columna 'Close'
        sns.lineplot(data=df['Close'], ax=ax, color=color, label='Close')

        # Establecer título con control para títulos personalizados
        if titles and pair in titles:
            ax.set_title(titles[pair])
        else:
            ax.set_title(f'Evolución histórica de {pair}')

        # Mejora de visibilidad en los ejes
        ax.set_ylabel('Precio de Cierre')
        ax.set_xlabel('Fecha')

    # Ajustar el layout para evitar superposición de elementos
    plt.tight_layout()
    plt.show()
```

1.2.6 Función para Graficar Predicciones

Matplotlib

[6]:
```
# Función para visualizar las predicciones con Matplotlib
def visualize_predictions_plt(data, models):
    for pair, model in models.items():
        df = data[pair]
        # Preprocesar los datos
        scaler = MinMaxScaler(feature_range=(0,1))
        scaled_data = scaler.fit_transform(df['Close'].values.reshape(-1,1))

        # Crear conjuntos de datos de entrenamiento y prueba
        train_data = scaled_data[0:int(len(scaled_data)*0.8), :]
        test_data = scaled_data[int(len(scaled_data)*0.8):, :]
        total_dataset = np.concatenate((train_data, test_data), axis=0)
        x_test, y_test = [], []
        for i in range(60, len(test_data)):
            x_test.append(total_dataset[i-60:i, 0])
            y_test.append(train_data[i, 0])
        x_test, y_test = np.array(x_test), np.array(y_test)
        x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

        # Realizar predicciones
        predictions = model.predict(x_test)
        predictions = scaler.inverse_transform(predictions)

        # Rellenar las primeras 60 entradas de las predicciones con NaN
        predictions = np.concatenate((np.full((60, 1), np.nan), predictions),␣
 ↪axis=0)
```

```
        # Visualizar los datos
        train = df[:int(len(scaled_data)*0.8)]
        valid = df[int(len(scaled_data)*0.8):]
        valid.insert(1, "Predictions", predictions, True)
        plt.figure(figsize=(16,8))
        plt.title(pair)
        plt.xlabel('Fecha', fontsize=18)
        plt.ylabel('Precio de Cierre', fontsize=18)
        plt.plot(train['Close'])
        plt.plot(valid[['Close', 'Predictions']])
        plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')
        plt.show()

        # Exportar a CSV
        valid.to_csv('datos_fx_predicciones_plt.csv', index=False)

        # Guardar los resultados en un archivo CSV
        #path = input("Indica la ruta donde deseas guardar el fichero en tu␣
 ↪computadora local: ")
        #valid.to_csv(path + '/' + pair + '_predictions.csv')
```

Seaborn

[31]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

# Función para Visualizar las Predicciones con Seaborn
def visualize_predictions_sns(data, models, train_color='#0d3160',␣
 ↪val_color='#45aab8', pred_color='#faab00'):
    for pair, model in models.items():
        df = data[pair]

        # Preprocesar los datos
        scaler = MinMaxScaler(feature_range=(0, 1))
        scaled_data = scaler.fit_transform(df['Close'].values.reshape(-1, 1))

        # Crear conjuntos de datos de entrenamiento y prueba
        train_data = scaled_data[0:int(len(scaled_data) * 0.8), :]
        test_data = scaled_data[int(len(scaled_data) * 0.8):, :]
        total_dataset = np.concatenate((train_data, test_data), axis=0)
        x_test, y_test = [], []
        for i in range(60, len(test_data)):
            x_test.append(total_dataset[i-60:i, 0])
```

```
        x_test = np.array(x_test)
        x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

        # Realizar predicciones
        predictions = model.predict(x_test)
        predictions = scaler.inverse_transform(predictions)

        # Rellenar las primeras 60 entradas de las predicciones con NaN
        predictions = np.concatenate((np.full((60, 1), np.nan), predictions),␣
↪axis=0)

        # Visualizar los datos
        train = df[:int(len(scaled_data) * 0.8)]
        valid = df[int(len(scaled_data) * 0.8):]
        valid.insert(1, "Predictions", predictions.flatten(), True)

        # Configurar Seaborn
        sns.set(style='white')

        plt.figure(figsize=(16, 8))
        plt.title(f'Predicciones para {pair}')
        plt.xlabel('Fecha', fontsize=18)
        plt.ylabel('Precio de Cierre', fontsize=18)

        # Graficar usando Seaborn con colores personalizables
        sns.lineplot(data=train['Close'], label='Train', color=train_color)
        sns.lineplot(data=valid['Close'], label='Validation', color=val_color)
        sns.lineplot(data=valid['Predictions'], label='Prediction',␣
↪color=pred_color)

        plt.legend()
        plt.show()

        # Exportar a CSV
        valid.to_csv(f'datos_fx_predicciones_sns_{pair}.csv', index=False)
```

1.3 Modelo de Redes Neuronales

1.3.1 Definición del Modelo

[8]:
```
# Crear y entrenar un modelo de red neuronal para cada paridad de monedas
def train_models(data):
    models = {}
    for pair, df in data.items():
        # Preprocesar los datos
        scaler = MinMaxScaler(feature_range=(0,1))
        scaled_data = scaler.fit_transform(df['Close'].values.reshape(-1,1))
```

```
        # Crear conjuntos de datos de entrenamiento y prueba
        train_data = scaled_data[0:int(len(scaled_data)*0.8), :]
        x_train, y_train = [], []
        for i in range(60, len(train_data)):
            x_train.append(train_data[i-60:i, 0])
            y_train.append(train_data[i, 0])
        x_train, y_train = np.array(x_train), np.array(y_train)
        x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))

        # Crear el modelo de red neuronal
        model = Sequential()
        model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.
 ↪shape[1], 1)))
        model.add(LSTM(units=50, return_sequences=False))
        model.add(Dense(units=25))
        model.add(Dense(units=1))

        # Entrenar el modelo
        model.compile(optimizer='adam', loss='mean_squared_error')
        model.fit(x_train, y_train, batch_size=1, epochs=1)

        models[pair] = model
    return models
```

A

1.3.2 Descargar Datos

[9]:
```
# Descargar Datos Históricos y Guardarlos en 'data'
data = download_data(currency_pairs)
```

```
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
```

1.3.3 Exportar CSV (Opcional)

[10]:
```
import pandas as pd

# Guardar Respaldo
bk_data_fx = data.copy()

# Convertir cada valor escalar en una lista
for key in bk_data_fx:
```

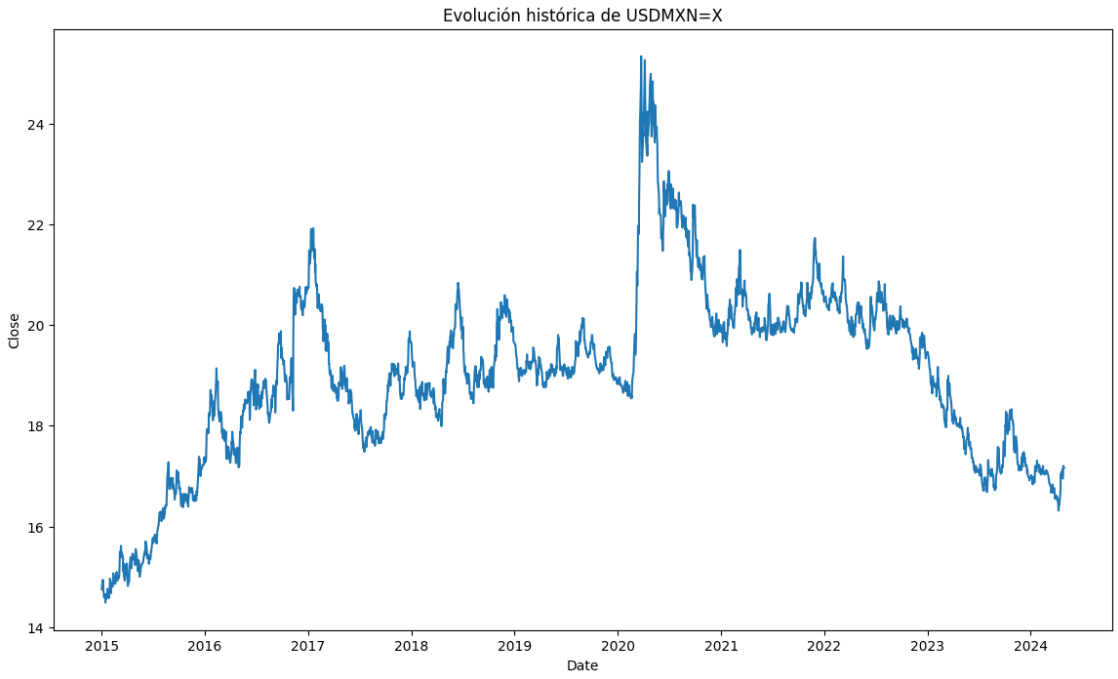
```
    bk_data_fx[key] = [bk_data_fx[key]]

# Convertir la copia del diccionario a DataFrame
bk_data_fx_df = pd.DataFrame(bk_data_fx)
```

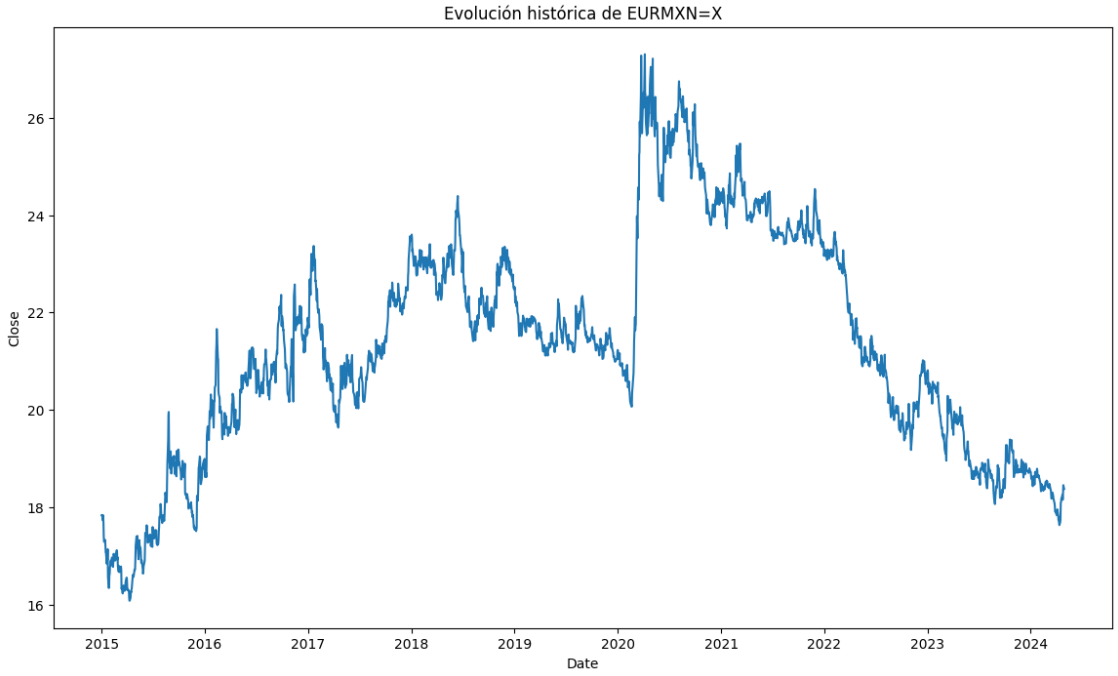
[11]:
```
# Exportar a CSV
bk_data_fx_df.to_csv('datos_fx_hist.csv', index=False)
```

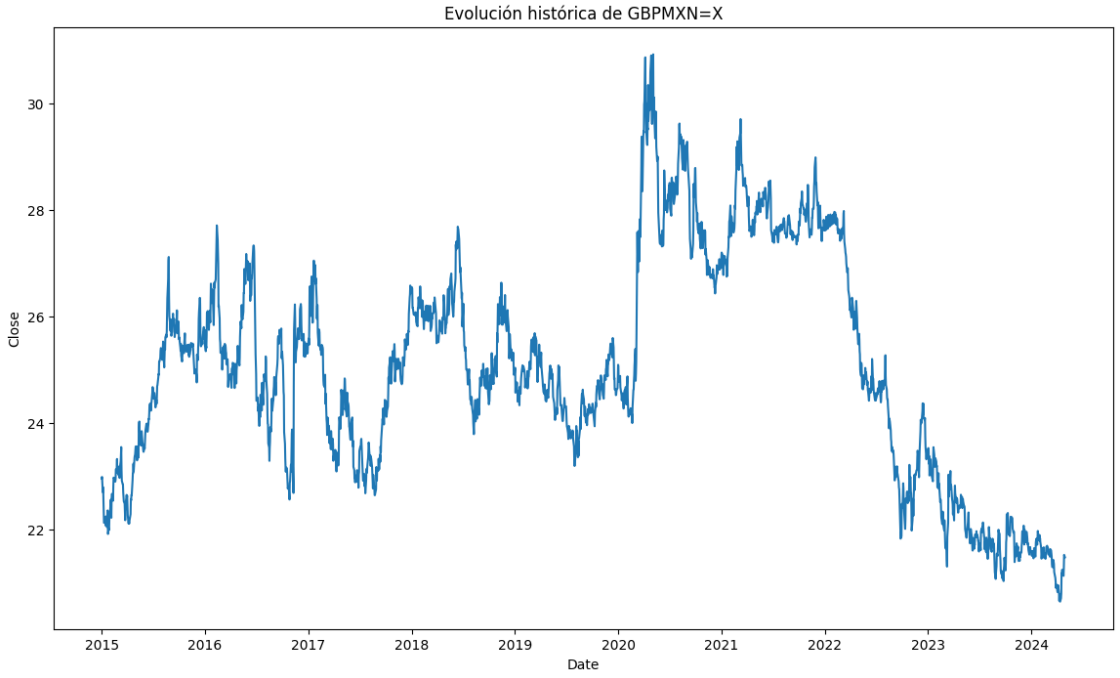
1.3.4 Graficar Datos Históricos

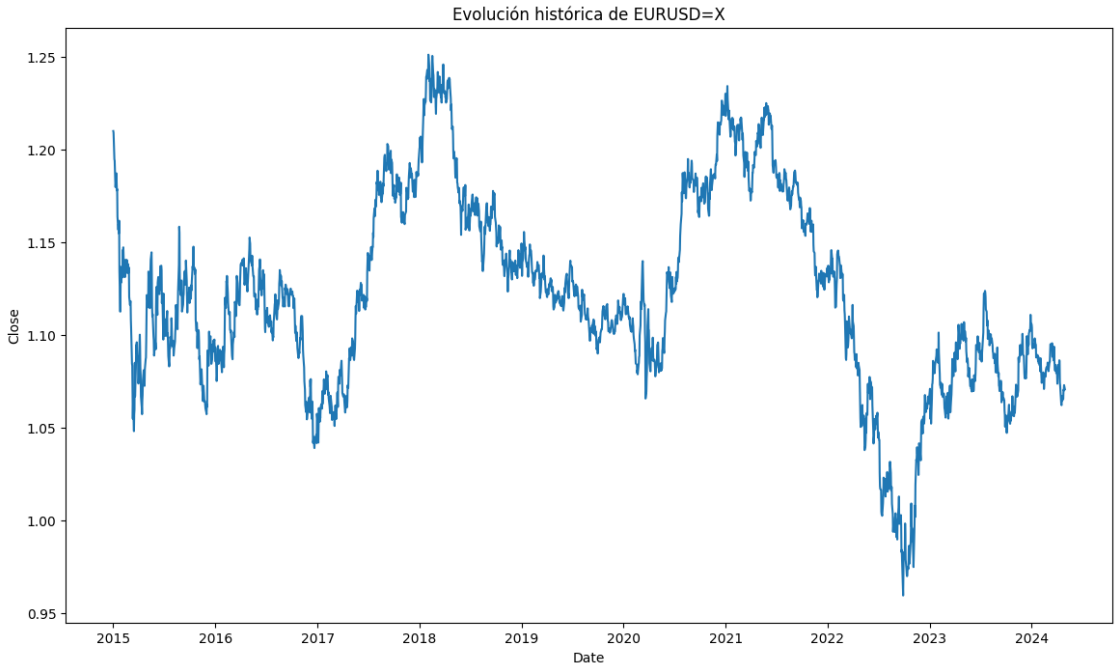
Matplotlib

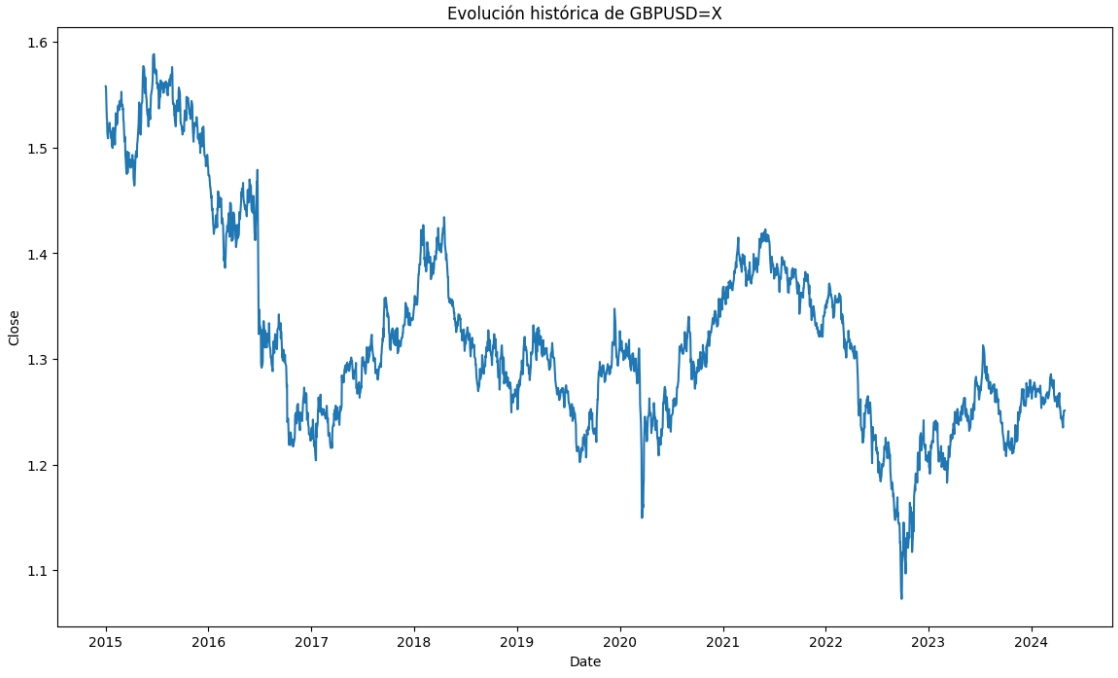
[12]:
```
# Graficar Datos Históricos
plot_data(data)
```




Evolución histórica de EURMXN=X
Close
Date
16
18
20
22
24
26
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024


Evolución histórica de GBPMXN=X
Close
Date
22
24
26
28
30
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024


Evolución histórica de EURUSD=X
Close
Date
1.25
1.20
1.15
1.10
1.05
1.00
0.95
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024


Evolución histórica de GBPUSD=X
Close
Date
1.6
1.5
1.4
1.3
1.2
1.1
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024

Seaborn

[33]:
```
# Graficar Predicciones con Seaborn
currency_pairs = ["USDMXN=X", "EURMXN=X", "GBPMXN=X", "EURUSD=X", "GBPUSD=X"]
data = download_data(currency_pairs)
plot_data_sns(data, font_scale=0.9, color='#0d3160')
```

```
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
[**********************100%%**********************]  1 of 1 completed
```

1.3.5 Entrenar el Modelo

[14]:
```
# Entrenar el Modelo
models = train_models(data)
```

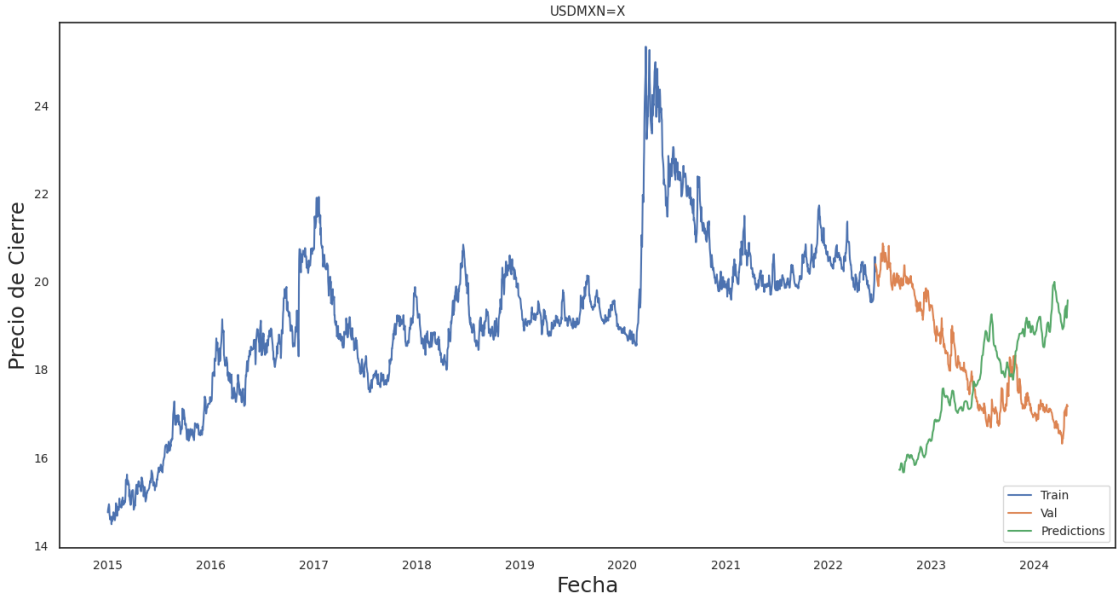
```
1885/1885 [==============================] - 25s 8ms/step - loss: 0.0019
1884/1884 [==============================] - 18s 8ms/step - loss: 0.0024
1884/1884 [==============================] - 18s 8ms/step - loss: 0.0027
1884/1884 [==============================] - 18s 8ms/step - loss: 0.0027
1884/1884 [==============================] - 20s 9ms/step - loss: 0.0022
```

1.3.6 Graficar Predicciones

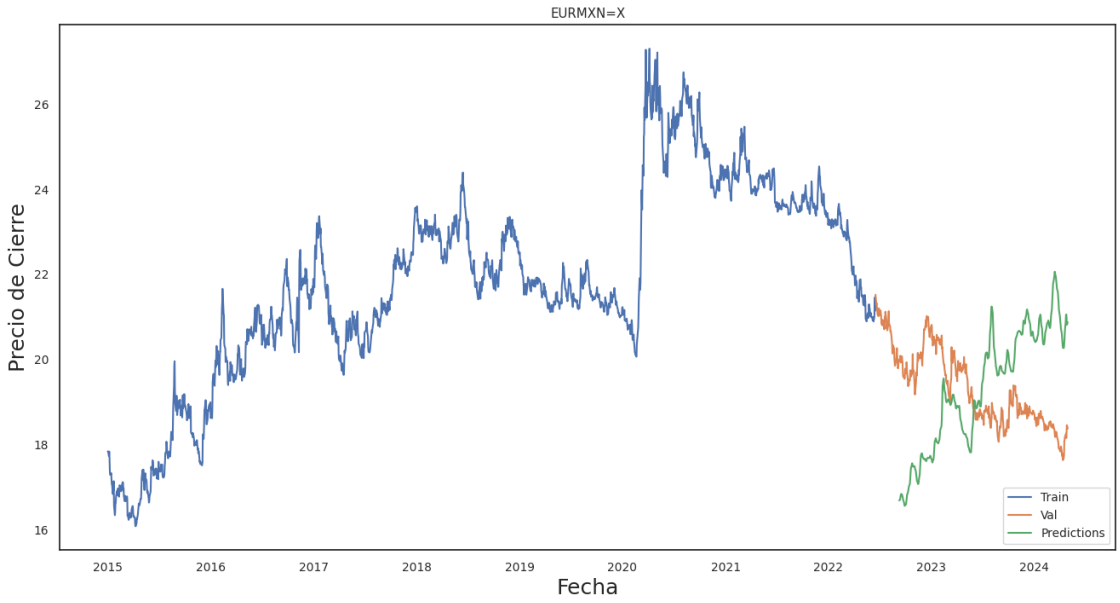
Matplotlib

[15]:
```
# Visualizar las Predicciones con Matplotlib
visualize_predictions_plt(data, models)
```

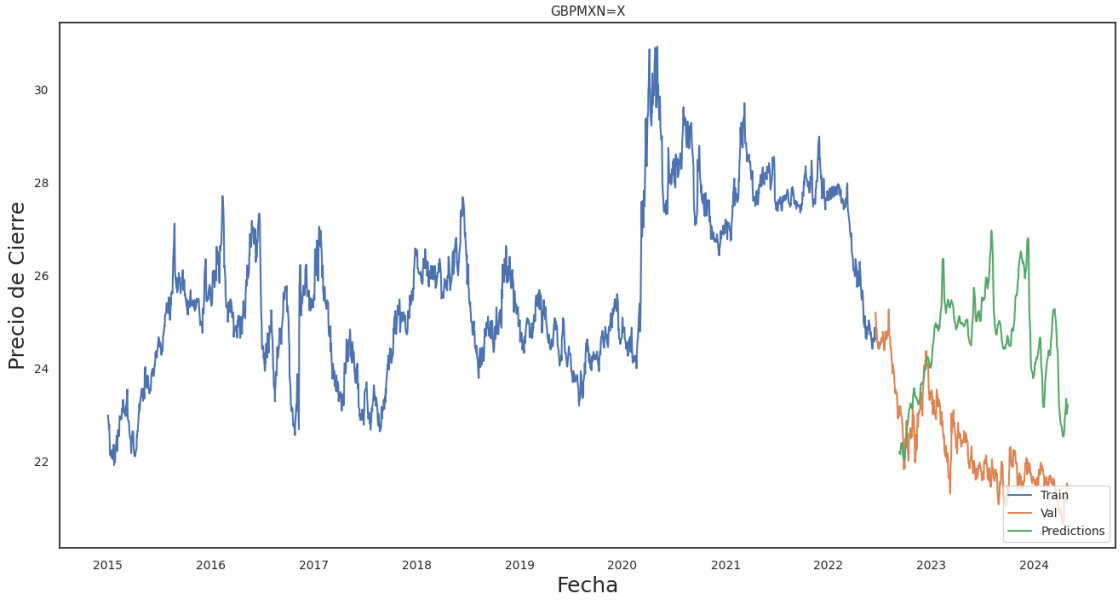
```
14/14 [==============================] - 1s 7ms/step
```



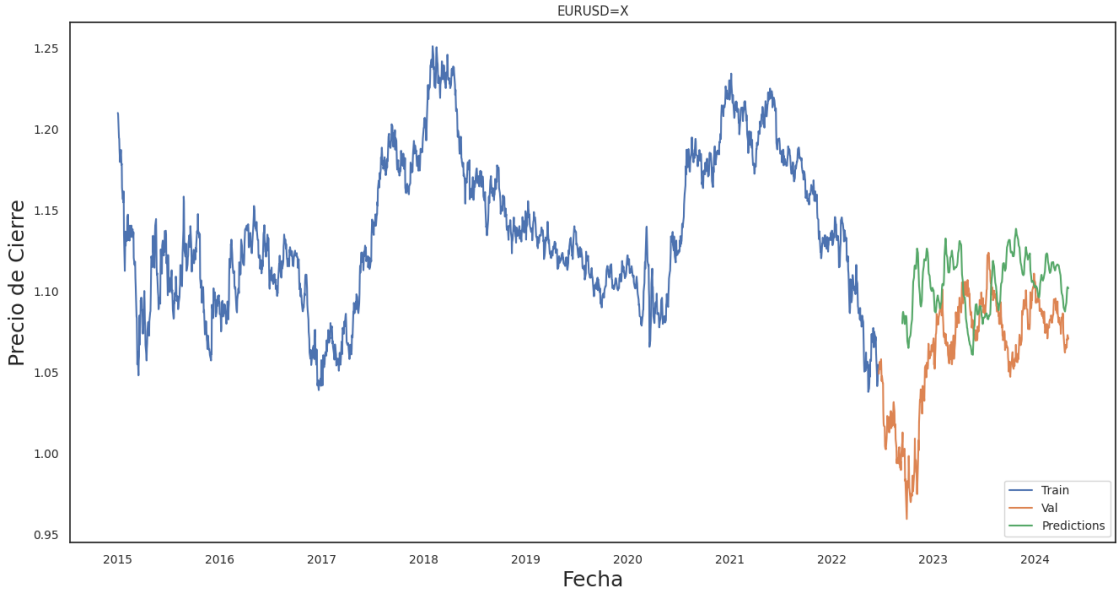

```
14/14 [==============================] - 1s 5ms/step
```



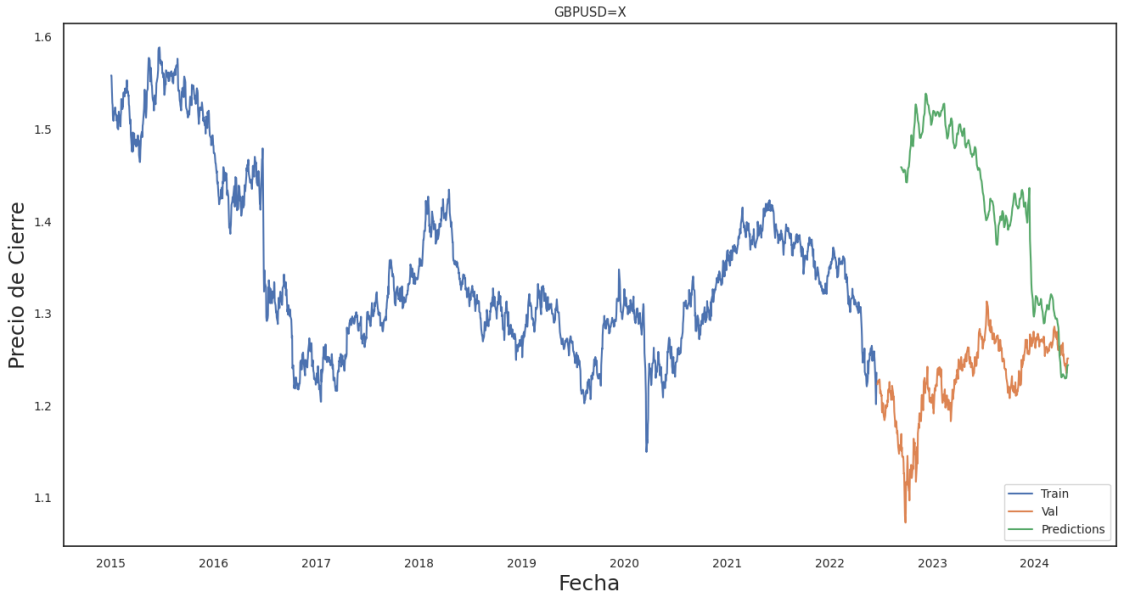

```
14/14 [==============================] - 1s 5ms/step
```




```
14/14 [==============================] - 1s 5ms/step
```



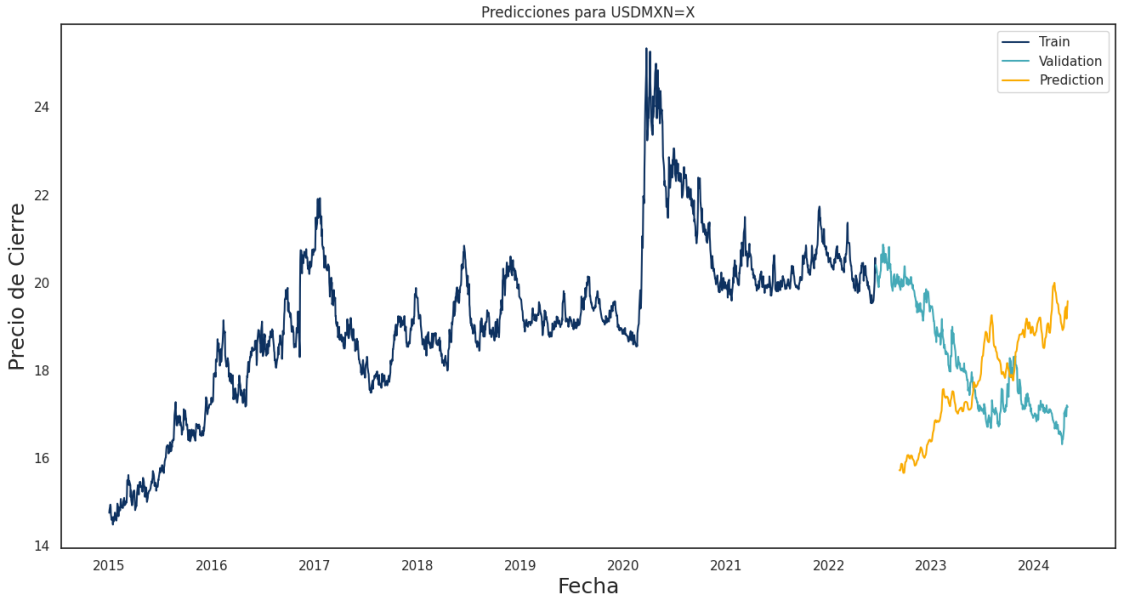

```
14/14 [==============================] - 1s 5ms/step
```



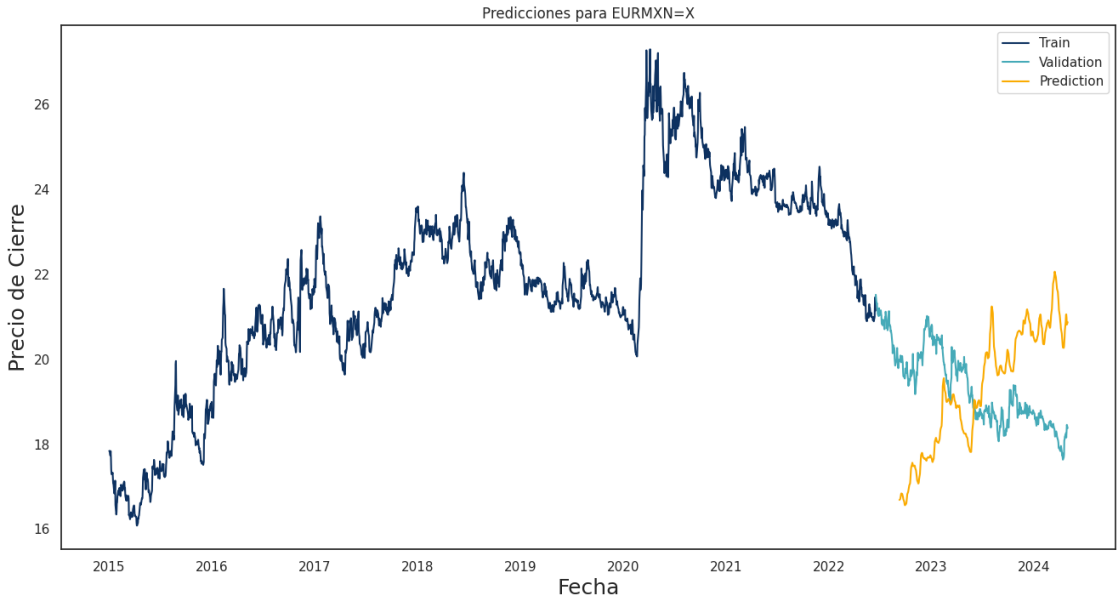
Seaborn

[32]:
```
# Visualizar las Predicciones con Seaborn
visualize_predictions_sns(data, models)
```

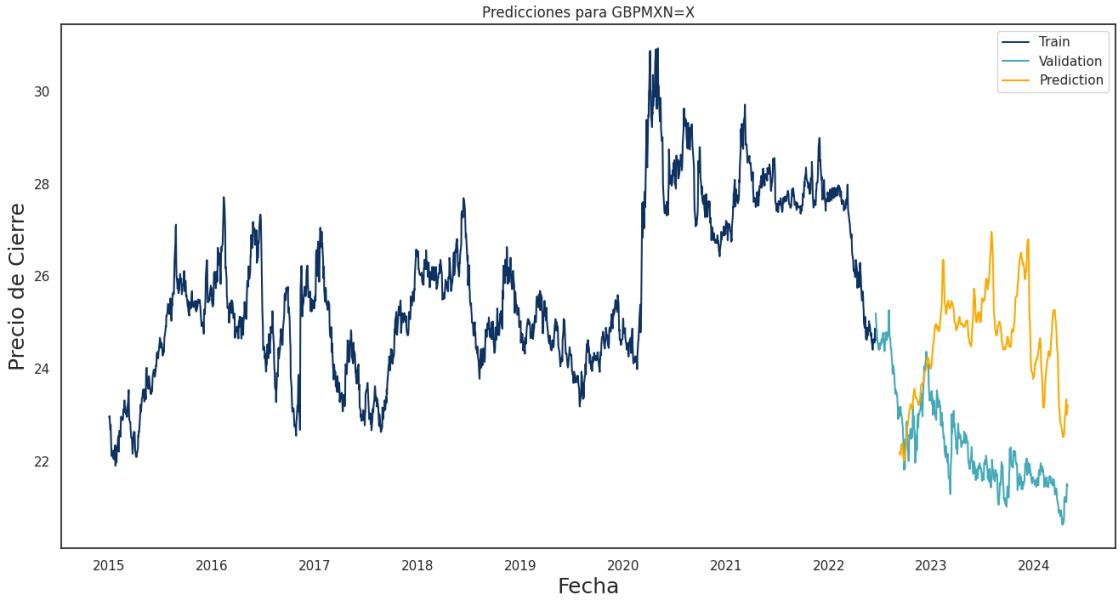
```
14/14 [==============================] - 0s 4ms/step
```



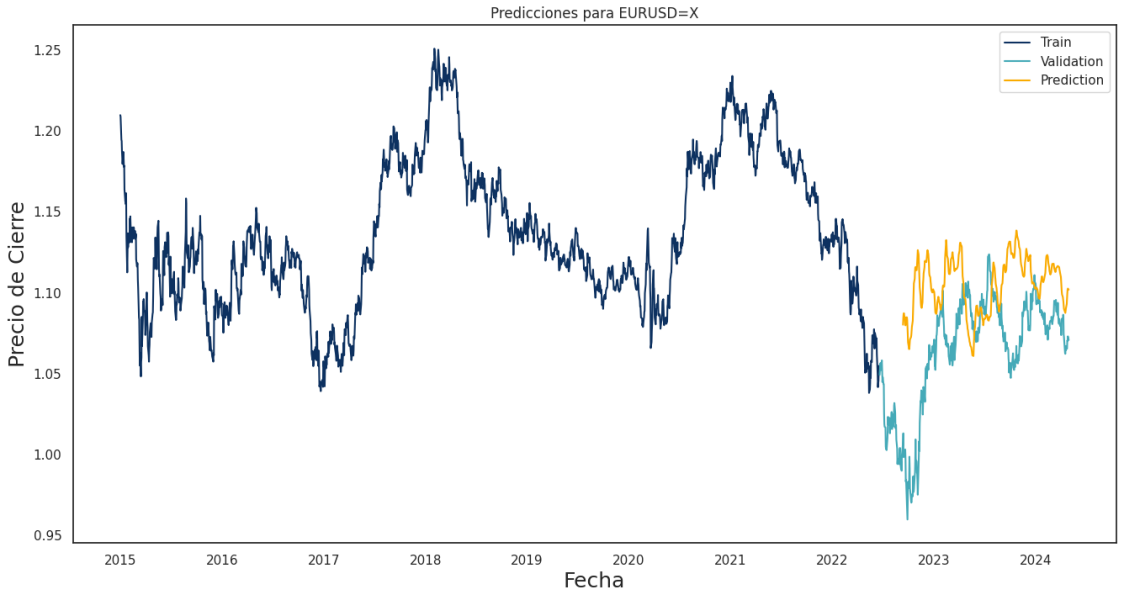

```
14/14 [==============================] - 0s 5ms/step
```



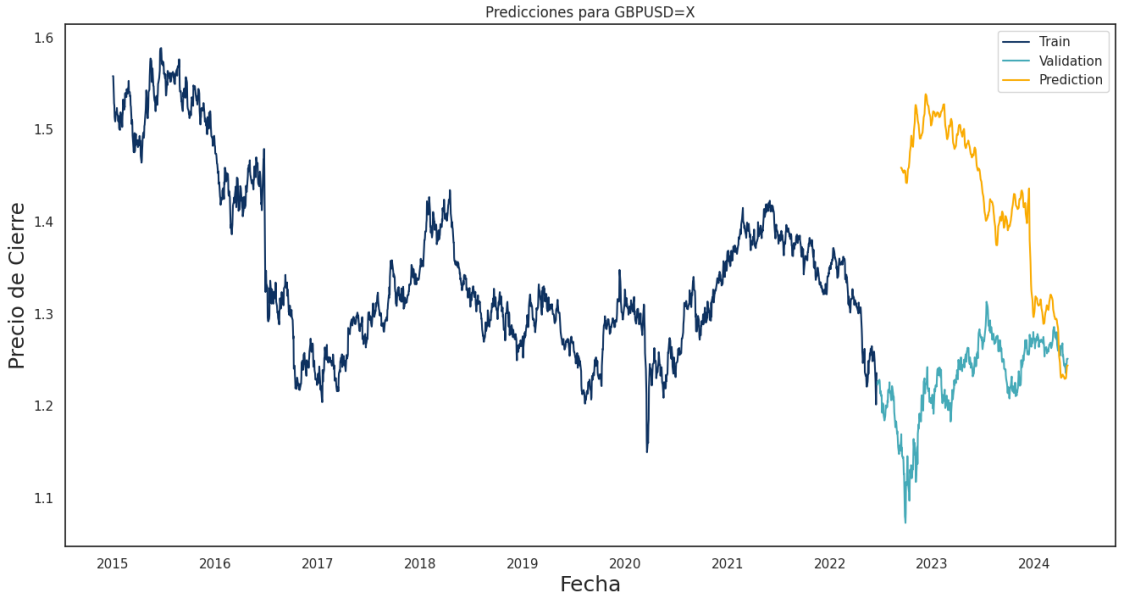

```
14/14 [==============================] - 0s 5ms/step
```




```
14/14 [==============================] - 0s 5ms/step
```




```
14/14 [==============================] - 0s 5ms/step
```



1.3.7 Mostrar Tablas con Datos de Paridades y Predicciones

[30]:
```
import pandas as pd
```

```
# Lista de paridades de monedas que se analizaron y para las cuales se␣
 ↪generaron predicciones
#currency_pairs = ["USDMXN=X", "EURMXN=X", "GBPMXN=X", "EURUSD=X", "GBPUSD=X"]

# Recorrer la lista de paridades y cargar cada CSV
for pair in currency_pairs:
    file_name = f'datos_fx_predicciones_sns_{pair}.csv'
    df = pd.read_csv(file_name)

    # Mostrar el nombre del par de divisas y la información del DataFrame
    print(f'Datos para {pair}:')
    print(f'Shape del DataFrame: {df.shape}')
    print(df.head())
    print('\n' + '-'*50)  # Imprime una línea separadora para cada par de␣
 ↪divisas
```

```
Datos para USDMXN=X:
Shape del DataFrame: (487, 7)
        Open  Predictions       High        Low      Close  Adj Close  Volume
0  20.397200          NaN  20.571800  20.295500  20.397200  20.397200       0
1  20.294310          NaN  20.372700  20.218700  20.294310  20.294310       0
2  20.268499          NaN  20.281111  20.116301  20.268499  20.268499       0
3  20.136999          NaN  20.245119  19.997801  20.136999  20.136999       0
4  20.049500          NaN  20.138670  19.993200  20.049500  20.049500       0

--------------------------------------------------
Datos para EURMXN=X:
Shape del DataFrame: (487, 7)
        Open  Predictions     High        Low      Close  Adj Close  Volume
0  21.527800          NaN  21.5541  21.376699  21.520100  21.520100       0
1  21.288799          NaN  21.3836  21.274500  21.290199  21.290199       0
2  21.323601          NaN  21.3631  21.231199  21.321699  21.321699       0
3  21.221399          NaN  21.2533  21.099501  21.219240  21.219240       0
4  21.194700          NaN  21.2103  21.020800  21.190001  21.190001       0

--------------------------------------------------
Datos para GBPMXN=X:
Shape del DataFrame: (487, 7)
        Open  Predictions       High        Low      Close  Adj Close  Volume
0  25.206200          NaN  25.211300  24.920481  25.197594  25.197594       0
1  24.810801          NaN  24.868225  24.755381  24.814827  24.814827       0
2  24.833599          NaN  24.879709  24.680401  24.829700  24.829700       0
3  24.713190          NaN  24.755768  24.548071  24.713499  24.713499       0
4  24.580700          NaN  24.653601  24.417038  24.574400  24.574400       0

--------------------------------------------------
Datos para EURUSD=X:
```

```
Shape del DataFrame: (486, 7)
       Open  Predictions      High       Low     Close  Adj Close  Volume
0  1.049142          NaN  1.054519  1.048636  1.049142   1.049142       0
1  1.052078          NaN  1.058190  1.051425  1.052078   1.052078       0
2  1.053264          NaN  1.060288  1.047186  1.053264   1.053264       0
3  1.056412          NaN  1.058067  1.048449  1.056412   1.056412       0
4  1.052011          NaN  1.056915  1.051370  1.052011   1.052011       0

--------------------------------------------------
Datos para GBPUSD=X:
Shape del DataFrame: (486, 7)
       Open  Predictions      High       Low     Close  Adj Close  Volume
0  1.222823          NaN  1.227777  1.220063  1.222823   1.222823       0
1  1.225010          NaN  1.231922  1.224920  1.225175   1.225175       0
2  1.226407          NaN  1.231467  1.216856  1.226663   1.226663       0
3  1.225325          NaN  1.229468  1.217182  1.225085   1.225085       0
4  1.226196          NaN  1.231679  1.224350  1.226076   1.226076       0

--------------------------------------------------
```

1.4 Fin