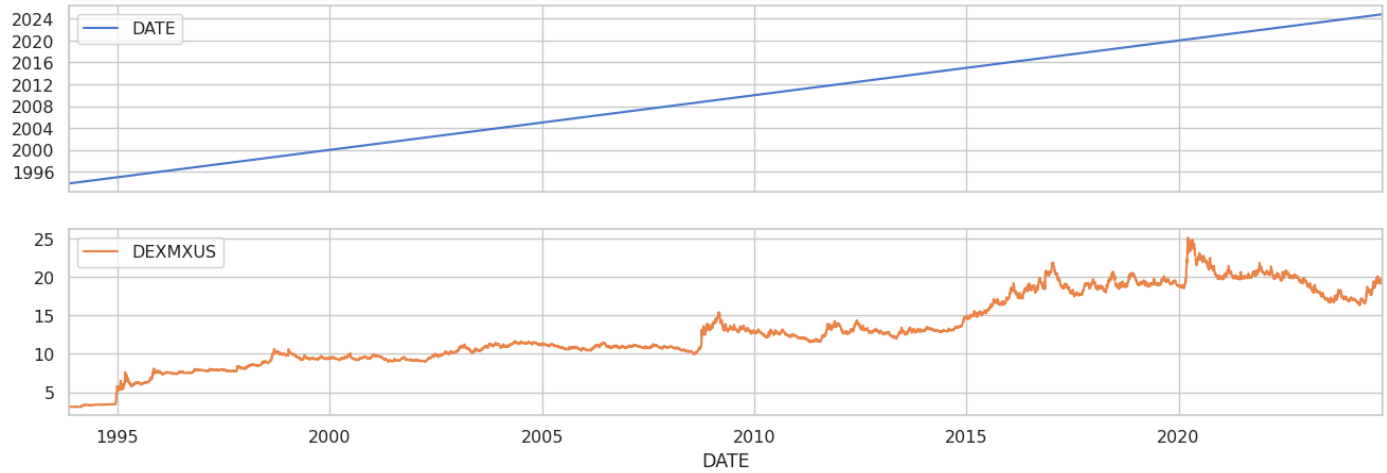
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import seaborn as sns
from pylab import rcParams
import matplotlib.pylab as plt
from matplotlib import rc

sns.set(style="whitegrid", palette="muted", font_scale=1.05)

rcParams["figure.figsize"] = 16,5

RANDOM_SEED = 42

np.random.seed(RANDOM_SEED)

tf.random.set_seed(RANDOM_SEED)


df = pd.read_csv("DEXMXUS (1).csv")
#df.head(50) ## leer los primero 50 datos


df["DEXMXUS"] = pd.to_numeric(df["DEXMXUS"],errors="coerce")## coerce fijara a los valores no valides como NaN
df["DATE"] =pd.to_datetime(df["DATE"])
df.index=df["DATE"]


print(df.dtypes)
```

```
DATE       datetime64[ns]
DEXMXUS           float64
dtype: object
```

```
df.plot(subplots=True)
```

```
array([<Axes: xlabel='DATE'>, <Axes: xlabel='DATE'>], dtype=object)
```




```
df.index=df["DATE"]


features= df["DEXMXUS"]
features=features.to_frame()
features.head(15)
```

DATE	DEXMXUS
1993-11-08	3.1520
1993-11-09	3.2400
1993-11-10	3.2400
1993-11-11	NaN
1993-11-12	3.2400
1993-11-15	3.2150
1993-11-16	NaN
1993-11-17	NaN
1993-11-18	3.1080
1993-11-19	3.1150
1993-11-22	3.1022
1993-11-23	3.1026
1993-11-24	3.1030
1993-11-25	NaN
1993-11-26	3.1140

Próximos pasos: Generar código con features | Ver gráficos recomendados | New interactive sheet

```
features=features.dropna()
features.head(15)
```

DATE	DEXMXUS
1993-11-08	3.1520
1993-11-09	3.2400
1993-11-10	3.2400
1993-11-12	3.2400
1993-11-15	3.2150
1993-11-18	3.1080
1993-11-19	3.1150
1993-11-22	3.1022
1993-11-23	3.1026
1993-11-24	3.1030
1993-11-26	3.1140
1993-11-29	3.1100
1993-11-30	3.1055
1993-12-01	3.1038
1993-12-02	3.1060

Próximos pasos: Generar código con features | Ver gráficos recomendados | New interactive sheet

Haz doble clic (o ingresa) para editar

```
train_size=int(len(features)*0.8)
test_size=len(features)-train_size
train, test= features[0:train_size], features[train_size:len(features)]
print(len(train),len(test))
```

```
6208 1552
```

Vamos a crear una funcion que permita salvar las ventanas temporales

```
#Usaremos time_step para el tamalo de la ventana temporal
def create_dataset(X,y,time_steps=1):
  Xs,ys=[],[] #Las declaramos como listas vacías donde se almacenaran las
                  #listas de las ventanas temporales
  for i in range(len(X)-time_steps):
    v=X.iloc[i:(i + time_steps)].values
    Xs.append(v)  #append: finaliza la lista
    ys.append(y.iloc[i + time_steps])  #append: finaliza la lista
  return np.array(Xs), np.array(ys)
```

crea un arreglo de listas de ventanas de tamaño 20

```
time_steps=20
X_train,y_train=create_dataset(train,train["DEXMXUS"],time_steps)
X_test,y_test=create_dataset(test,test["DEXMXUS"],time_steps)
```

Definimos la red neuronal

Definimos el proceso de modelaje y a la red resultado se manda modelo existe otras definidas con memorias a largo plazo

```
#Red multicapa o de capas apiladas
model=keras.Sequential()
#Apilamos la capa densa o totalmente conectada a una sola red neuronal (salida)
model.add(keras.layers.LSTM(64,input_shape=(X_train.shape[1],X_train.shape[2])))
model.add(keras.layers.Dense(1))
#Compilamos la red con un hyperparámetro, es decir una función de pérdida, optimzador, razón de aprendizaje, etc
#"mae" MeanAbsoluteError class: Calcula la media de la diferencia absoluta entre etiquetas y predicciones.
#                              loss = abs(y_true - y_pred)
model.compile(loss="mae",optimizer=keras.optimizers.RMSprop(clipvalue=1.0))
```

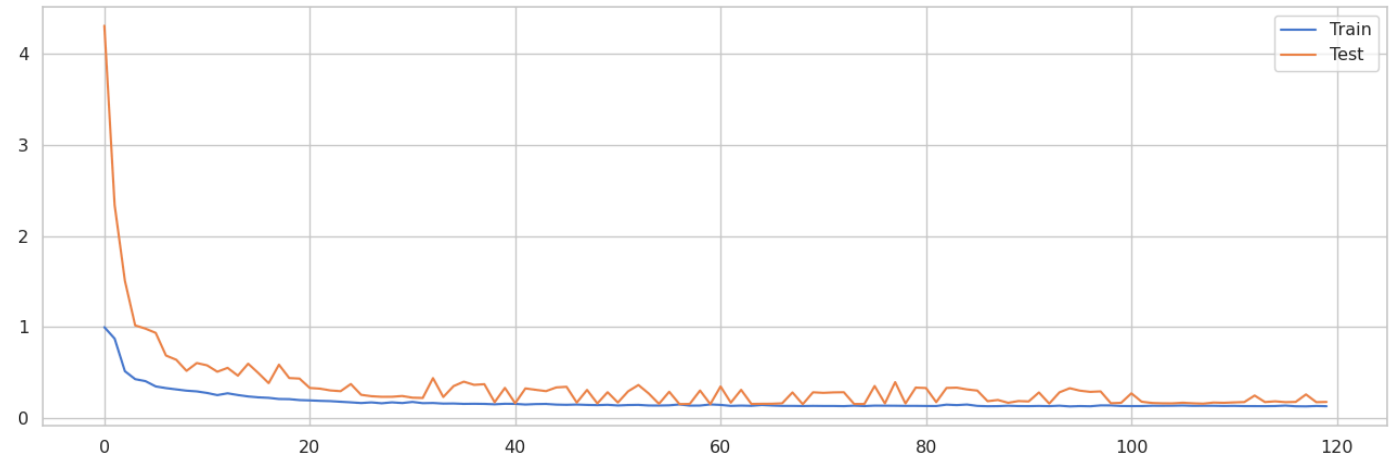
```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input_shape`/`input_dim` argument
  super().__init__(**kwargs)
```

```
history=model.fit(
    X_train,
    y_train,
    epochs=120, #hiperparámetro que define el número de veces que el algoritmo
                #de aprendizaje funcionará en todo el conjunto de datos de entrenamiento.
    batch_size=32, #lotes de agrupamiento
    verbose=1, #Datos del proceso en pantalla
    validation_split=0.01,
    shuffle=False
)
```

```
192/192 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - loss: 0.1116 - val_loss: 0.1591
Epoch 106/120
192/192 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - loss: 0.1112 - val_loss: 0.1658
Epoch 107/120
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 0.1089 - val_loss: 0.1597
Epoch 108/120
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 0.1104 - val_loss: 0.1559
Epoch 109/120
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.1110 - val_loss: 0.1678
Epoch 110/120
192/192 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - loss: 0.1092 - val_loss: 0.1648
Epoch 111/120
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - loss: 0.1104 - val_loss: 0.1696
Epoch 112/120
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.1086 - val_loss: 0.1733
Epoch 113/120
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.1074 - val_loss: 0.2452
Epoch 114/120
192/192 ━━━━━━━━━━━━━━━━━━━━ 4s 20ms/step - loss: 0.1065 - val_loss: 0.1733
Epoch 115/120
192/192 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - loss: 0.1093 - val_loss: 0.1812
Epoch 116/120
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.1181 - val_loss: 0.1723
Epoch 117/120
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.1073 - val_loss: 0.1750
Epoch 118/120
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.1072 - val_loss: 0.2573
Epoch 119/120
192/192 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 0.1087 - val_loss: 0.1724
Epoch 120/120
192/192 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.1108 - val_loss: 0.1746
```

```
plt.plot(history.history["loss"],label="Train")
plt.plot(history.history["val_loss"],label="Test")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7ef636ed61d0>
```

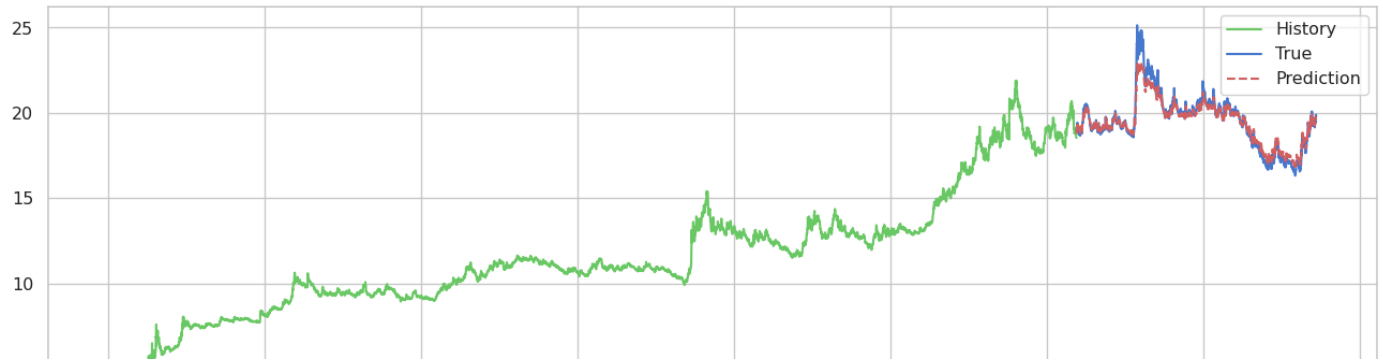



```
y_pred=model.predict(X_test)
```

```
48/48 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step
```

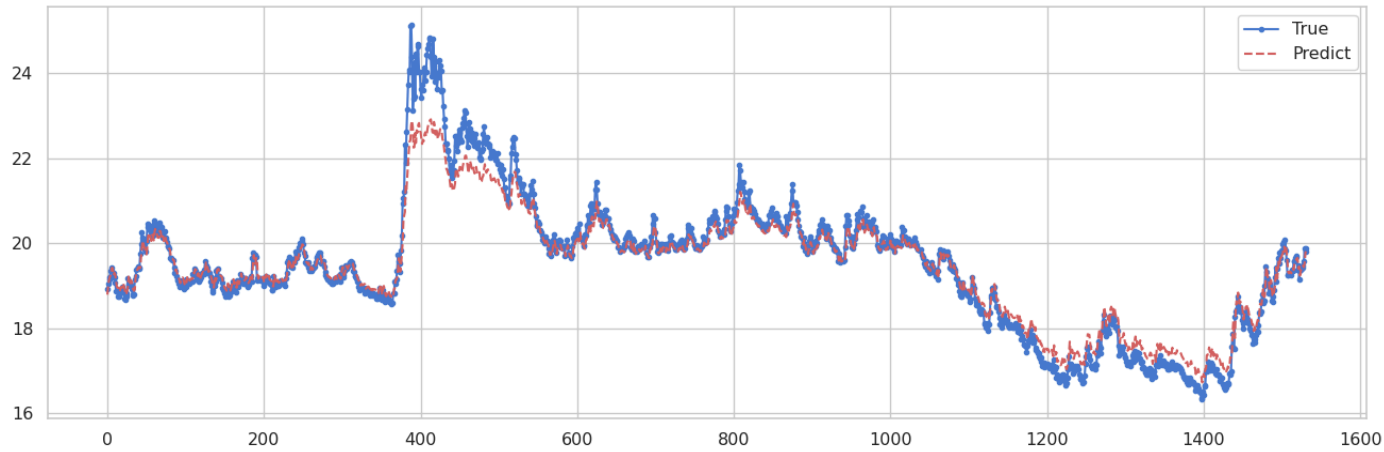
```
plt.plot(np.arange(0,len(y_train)),y_train,"g",label="History")
plt.plot(np.arange(len(y_train),len(y_train)+len(y_test)),y_test,label="True")
plt.plot(np.arange(len(y_train),len(y_train)+len(y_test)),y_pred,"r--",label="Prediction")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7ef6c27b9690>
```




```
plt.plot(y_test,marker=".",label="True")
plt.plot(y_pred,"r--",label="Predict")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7ef62dee27d0>
```




```
from keras.models import model_from_json
model.save("Tipo_cambio_octubre2024.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is consi
```