

Redes Neuronales aplicado a la astrofísica: Predicción de manchas solares.

Moreno Santiago José Miguel

Zamora Flores Guillermo Armando

Moran Alvarado Luis Andrés

Proyecto desarrollado en una estancia del Programa Delfín, donde se construyó un modelo de red neuronal LSTM para predecir manchas solares.

Los datos fueron recopilados de [SIDC](#).

▼ Carga y Visualización de datos

```
#Montar drive
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

```
#Importamos librerías
import pandas as pd
import plotly.express as px
```

```
# Carga el archivo CSV en un DataFrame
Dataframe = pd.read_csv('/content/drive/MyDrive/Redes neuronales aplicado a ciclos solares/S
Dataframe.head()
```

	No. de manchas solares	Fecha
0	0	1823-01-01
1	0	1823-01-02
2	0	1823-01-03
3	0	1823-01-04
4	0	1823-01-05

```
#Establecer la columna de fecha como el índice del DataFrame:  
Dataframe.set_index('Fecha', inplace=True)
```

```
Dataframe.index = pd.to_datetime(Dataframe.index)
```

```
# Verifica que el índice se ha convertido correctamente  
print(Dataframe.index)
```

```
→ DatetimeIndex(['1823-01-01', '1823-01-02', '1823-01-03', '1823-01-04',  
                 '1823-01-05', '1823-01-06', '1823-01-07', '1823-01-08',  
                 '1823-01-09', '1823-01-10',  
                 ...  
                 '2024-05-22', '2024-05-23', '2024-05-24', '2024-05-25',  
                 '2024-05-26', '2024-05-27', '2024-05-28', '2024-05-29',  
                 '2024-05-30', '2024-05-31'],  
                 dtype='datetime64[ns]', name='Fecha', length=73566, freq=None)
```

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

```
# Cambiar el tipo de letra globalmente (para toda la gráfica)  
rcParams['font.family'] = 'sans-serif' # Puedes probar con 'serif', 'sans-serif', o fuentes
```

```
# Aplicar un estilo más elegante  
sns.set(style="whitegrid")
```

```
# Crear la figura  
plt.figure(figsize=(12, 6))
```

```
# Graficar los datos con una línea roja  
sns.lineplot(x=Dataframe.index, y=Dataframe['No. de manchas solares'], label='Manchas Solares',  
              color="#e17a0d", linewidth=2.5)
```

```
# Personalizar el título principal con el tipo de letra que prefieras y color negro  
plt.title("Número de Manchas Solares de 1823-01-01 a 2024-05-31",  
          fontsize=18,  
          fontweight='bold',  
          pad=20,
```

```
fontdict={'fontname': 'serif', 'color': 'black'}) # Cambiar 'serif' por una fuente
```

```
# Personalizar los ejes
plt.xlabel('Fecha', fontsize=14, labelpad=10)
plt.ylabel('Número de Manchas Solares', fontsize=14, labelpad=10)

# Rotar las etiquetas del eje X si hay muchas fechas
plt.xticks(rotation=45, ha='right')

# Personalizar la cuadrícula
plt.grid(True, linestyle='--', alpha=0.6)

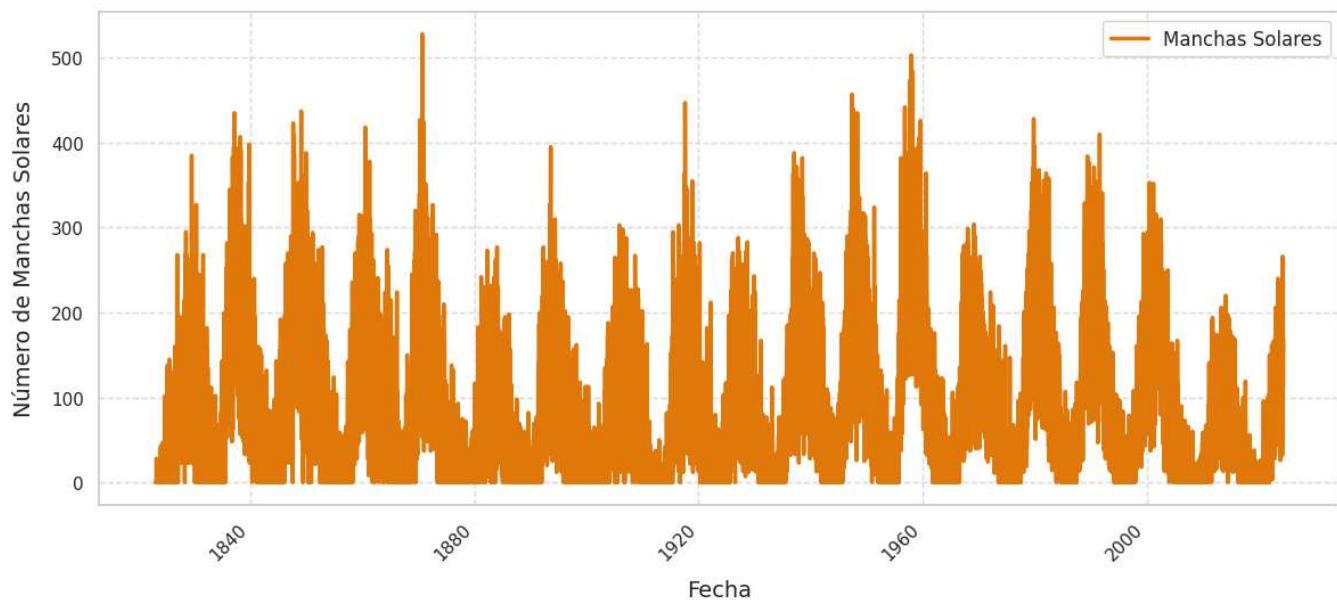
# Leyenda
plt.legend(loc='upper right', fontsize=12)

# Ajustar márgenes
plt.tight_layout()

# Mostrar la gráfica
plt.show()
```



Número de Manchas Solares de 1823-01-01 a 2024-05-31



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

# Cambiar el tipo de letra globalmente (para toda la gráfica)
rcParams['font.family'] = 'sans-serif'

# Aplicar un estilo más elegante
sns.set(style="whitegrid")

# Crear la figura
plt.figure(figsize=(12, 6))

# Graficar los datos como un gráfico de dispersión (scatter plot)
plt.scatter(Dataframe.index, Dataframe['No. de manchas solares'],
            color="#e17a0d", s=30) # 's' es el tamaño de los puntos

# Remover etiquetas de los ejes y título
```

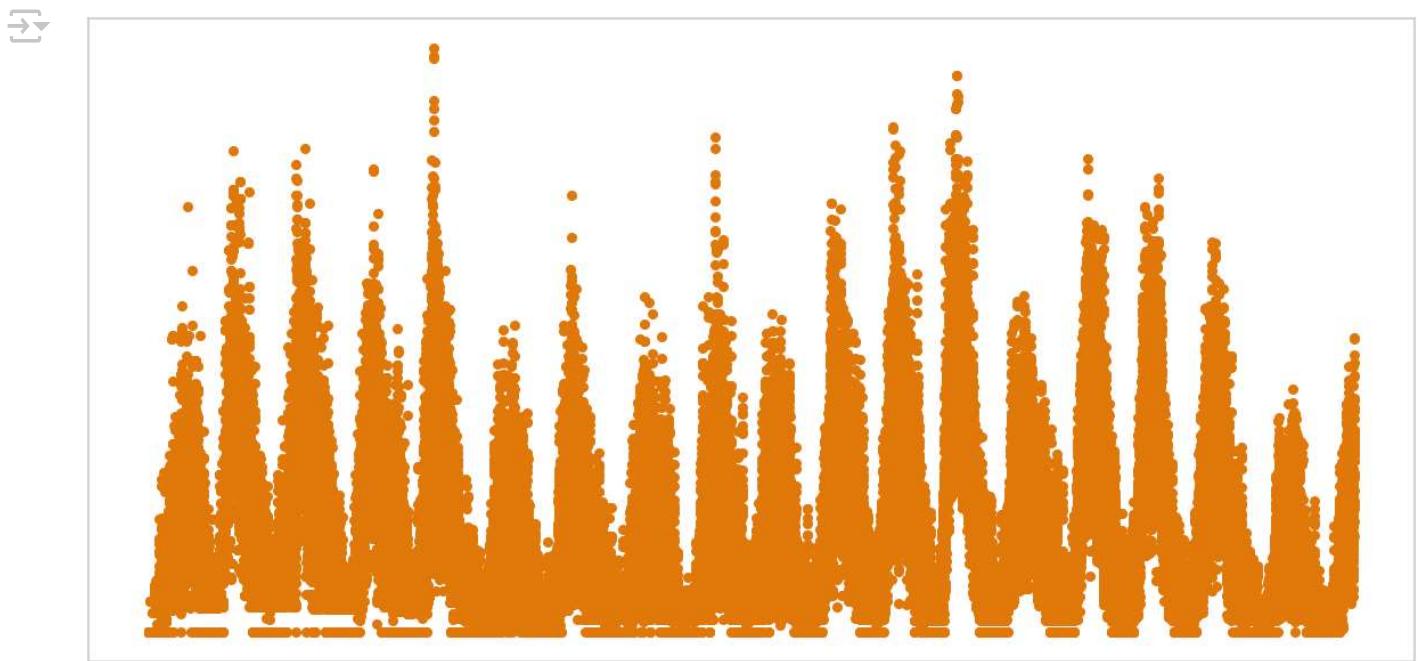
```
plt.xlabel('')
plt.ylabel('')
plt.title('')

# Ocultar las marcas del eje X e Y
plt.xticks([])
plt.yticks([])

# Personalizar la cuadrícula
plt.grid(True, linestyle='--', alpha=0.6)

# Ajustar márgenes
plt.tight_layout()

# Mostrar la gráfica
plt.show()
```



▼ Instalación de Optuna

```
!pip install optuna
```

→ Mostrar el resultado oculto

```
!pip install mlflow --ignore-installed blinker --quiet
```

```
[██████████] 128.2/128.2 kB 18.8 MB/s eta 0:00:00
[██████████] 105.4/105.4 kB 16.4 MB/s eta 0:00:00
[██████████] 8.3/8.3 MB 98.0 MB/s eta 0:00:00
[██████████] 18.2/18.2 MB 79.4 MB/s eta 0:00:00
[██████████] 59.9/59.9 kB 8.3 MB/s eta 0:00:00
[██████████] 107.0/107.0 kB 16.1 MB/s eta 0:00:00
[██████████] 54.0/54.0 kB 8.5 MB/s eta 0:00:00
[██████████] 13.0/13.0 MB 99.2 MB/s eta 0:00:00
[██████████] 294.6/294.6 kB 2.3 MB/s eta 0:00:00
[██████████] 38.3/38.3 MB 41.6 MB/s eta 0:00:00
[██████████] 505.5/505.5 kB 41.1 MB/s eta 0:00:00
[██████████] 705.5/705.5 kB 44.3 MB/s eta 0:00:00
[██████████] 64.9/64.9 kB 8.3 MB/s eta 0:00:00
[██████████] 13.4/13.4 MB 88.6 MB/s eta 0:00:00
[██████████] 41.1/41.1 MB 18.5 MB/s eta 0:00:00
[██████████] 3.1/3.1 MB 57.1 MB/s eta 0:00:00
[██████████] 44.0/44.0 kB 5.5 MB/s eta 0:00:00
[██████████] 133.3/133.3 kB 15.0 MB/s eta 0:00:00
[██████████] 84.4/84.4 kB 10.3 MB/s eta 0:00:00
[██████████] 121.4/121.4 kB 641.6 kB/s eta 0:00:00
[██████████] 227.3/227.3 kB 25.2 MB/s eta 0:00:00
[██████████] 62.7/62.7 kB 9.4 MB/s eta 0:00:00
[██████████] 202.9/202.9 kB 28.0 MB/s eta 0:00:00
[██████████] 52.8/52.8 kB 8.1 MB/s eta 0:00:00
[██████████] 305.2/305.2 kB 37.1 MB/s eta 0:00:00
[██████████] 4.6/4.6 MB 106.1 MB/s eta 0:00:00
[██████████] 1.6/1.6 MB 88.6 MB/s eta 0:00:00
[██████████] 4.5/4.5 MB 109.0 MB/s eta 0:00:00
[██████████] 103.2/103.2 kB 15.9 MB/s eta 0:00:00
[██████████] 229.9/229.9 kB 31.1 MB/s eta 0:00:00
[██████████] 130.5/130.5 kB 19.5 MB/s eta 0:00:00
[██████████] 345.4/345.4 kB 43.1 MB/s eta 0:00:00
[██████████] 142.1/142.1 kB 20.7 MB/s eta 0:00:00
[██████████] 66.8/66.8 kB 10.4 MB/s eta 0:00:00
[██████████] 163.0/163.0 kB 24.5 MB/s eta 0:00:00
[██████████] 301.8/301.8 kB 37.9 MB/s eta 0:00:00
[██████████] 616.0/616.0 kB 57.0 MB/s eta 0:00:00
[██████████] 80.3/80.3 kB 13.7 MB/s eta 0:00:00
```

```
torch 2.3.0+cu121 requires nvidia-cuann-cu12==8.9.2.26; platform_system == "Linux" and torch 2.3.0+cu121 requires nvidia-cufft-cu12==11.0.2.54; platform_system == "Linux" and torch 2.3.0+cu121 requires nvidia-curand-cu12==10.3.2.106; platform_system == "Linux" and torch 2.3.0+cu121 requires nvidia-cusolver-cu12==11.4.5.107; platform_system == "Linux" and torch 2.3.0+cu121 requires nvidia-cusparse-cu12==12.1.0.106; platform_system == "Linux" and torch 2.3.0+cu121 requires nvidia-nccl-cu12==2.20.5; platform_system == "Linux" and torch 2.3.0+cu121 requires nvidia-nvtx-cu12==12.1.105; platform_system == "Linux" and cudf-cu12 24.4.1 requires pandas<2.2.2dev0,>=2.0, but you have pandas 2.2.2 which is cudf-cu12 24.4.1 requires pyarrow<15.0.0a0,>=14.0.1, but you have pyarrow 15.0.2 which is google-colab 1.0.0 requires pandas==2.0.3, but you have pandas 2.2.2 which is incompatible with pyarrow 15.0.2 google-colab 1.0.0 requires requests==2.31.0, but you have requests 2.32.3 which is incompatible with requests 2.31.0 imageio 2.31.6 requires pillow<10.1.0,>=8.3.2, but you have pillow 10.4.0 which is incompatible with pillow 8.3.2 tensorflow 2.15.0 requires wrapt<1.15,>=1.11.0, but you have wrapt 1.16.0 which is incompatible with wrapt 1.11.0 tensorflow-metadata 1.15.0 requires protobuf<4.21,>=3.20.3; python version < "3.11", but you have protobuf 3.20.3 which is incompatible with tensorflow-metadata 1.15.0
```

```
REPO_NAME= "Ciclo_solar"
REPO_OWNER= "chichastark14"
USER_NAME = "chichastark14"
```

```
import mlflow
import os
from getpass import getpass

os.environ['MLFLOW_TRACKING_USERNAME'] = USER_NAME
os.environ['MLFLOW_TRACKING_PASSWORD'] = getpass('Enter your DAGsHub access token or password')

mlflow.set_tracking_uri(f'https://dagshub.com/{REPO_OWNER}/{REPO_NAME}.mlflow')
```

→ Enter your DAGsHub access token or password:

▼ Multiples pruebas Optuna

Optuna es una biblioteca de optimización de hiperparámetros que permite buscar de manera eficiente los mejores valores para mejorar el rendimiento de modelos de machine learning.

En este proyecto, **Optuna** se utilizó para encontrar la mejor configuración de la red LSTM, ajustando parámetros como el número de neuronas, la tasa de aprendizaje y el dropout, optimizando así la precisión de la predicción de manchas solares.

```
pip install optuna-integration
```

→ Collecting optuna-integration
 Downloading optuna_integration-3.6.0-py3-none-any.whl (93 kB)
 ━━━━━━━━━━ 93.4/93.4 kB 3.0 MB/s eta 0:00:00
 Collecting optuna (from optuna-integration)
 Downloading optuna-3.6.1-py3-none-any.whl (380 kB)
 ━━━━━━━━━━ 380.1/380.1 kB 19.3 MB/s eta 0:00:00
 Collecting alembic>=1.5.0 (from optuna->optuna-integration)

```
  Downloading alembic-1.13.2-py3-none-any.whl (232 kB)
  _____ 233.0/233.0 kB 32.1 MB/s eta 0:00:00
Collecting colorlog (from optuna->optuna-integration)
  Downloading colorlog-6.8.2-py3-none-any.whl (11 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from optuna->optuna-integration)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from optuna->optuna-integration)
Requirement already satisfied: sqlalchemy>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from optuna->optuna-integration)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from optuna->optuna-integration)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.10/dist-packages (from optuna->optuna-integration)
Collecting Mako (from alembic>=1.5.0->optuna->optuna-integration)
  Downloading Mako-1.3.5-py3-none-any.whl (78 kB)
  _____ 78.6/78.6 kB 12.8 MB/s eta 0:00:00
Requirement already satisfied: typing-extensions>=4 in /usr/local/lib/python3.10/dist-packages (from alembic>=1.5.0->optuna->optuna-integration)
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.10/dist-packages (from alembic>=1.5.0->optuna->optuna-integration)
Requirement already satisfied: MarkupSafe>=0.9.2 in /usr/local/lib/python3.10/dist-packages (from alembic>=1.5.0->optuna->optuna-integration)
Installing collected packages: Mako, colorlog, alembic, optuna, optuna-integration
Successfully installed Mako-1.3.5 alembic-1.13.2 colorlog-6.8.2 optuna-3.6.1 optuna-integration
```

▼ Preprocesamiento de los datos

```
# Importar librerías
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import SimpleRNN, Dense, TimeDistributed, LSTM, Dropout
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras import models, layers, regularizers, optimizers
import os
from google.colab import files
from tensorflow.keras.callbacks import ModelCheckpoint, Callback, EarlyStopping
from keras.preprocessing.sequence import TimeseriesGenerator
import optuna
from optuna import Trial
# Importar BatchNormalization desde tf.keras.layers
from tensorflow.keras.layers import BatchNormalization
from optuna.integration import TFKerasPruningCallback
from sklearn.metrics import mean_squared_error

# Seleccionar solo la columna 'No. de manchas solares' del DataFrame original
df = Dataframe[['No. de manchas solares']]

# Escalar los datos a un rango de 0 a 1 para mejorar el rendimiento del modelo
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df)
```

```
# Convertir los datos escalados de nuevo a un DataFrame, manteniendo el mismo índice y nombre
df_scaled = pd.DataFrame(scaled_data, index=df.index, columns=['numero de manchas solares'])
```

```
# Determinar el tamaño del conjunto de entrenamiento y validación
train_size = int(len(df_scaled) * 0.6)
test_size = int(len(df_scaled) * 0.40)
```

```
# Separar los datos en conjuntos de entrenamiento y prueba
train_data = df_scaled[:train_size]
test_data = df_scaled[train_size:train_size + test_size]
```

```
# Verificar las longitudes
print("Longitud del conjunto total:", len(df_scaled))
print("Longitud del conjunto de entrenamiento:", len(train_data))
print("Longitud del conjunto de prueba:", len(test_data))
```

→ Longitud del conjunto total: 73566
Longitud del conjunto de entrenamiento: 44139
Longitud del conjunto de prueba: 29426

```
# Función para crear datos de secuencia
```

```
def create_sequence_data(data, seq_length):
    X = []
    y = []
    for i in range(len(data) - seq_length):
        X.append(data.iloc[i:i + seq_length].values)
        y.append(data.iloc[i + seq_length].values)
    return np.array(X), np.array(y)
```

```
# Función para definir y crear el modelo
```

```
def create_model(trial, seq_length):
    model = models.Sequential()
    n_lstm_layers = trial.suggest_int("n_lstm_layers", 1, 3)
    for i in range(n_lstm_layers):
        units = trial.suggest_int(f"lstm_units_{i}", 10, 500)
        return_sequences = True if i < n_lstm_layers - 1 else False
        if i == 0:
            model.add(layers.LSTM(units=units, return_sequences=return_sequences, input_shape=(None, 1)))
        else:
            model.add(layers.LSTM(units=units, return_sequences=return_sequences))

        dropout_rate = trial.suggest_uniform(f"dropout_{i}", 0.1, 0.7)
        if i < n_lstm_layers - 1:
            model.add(layers.Dropout(dropout_rate))

    model.add(layers.Flatten())
```

```
n_dense_layers = trial.suggest_int("n_dense_layers", 1, 3)
for i in range(n_dense_layers):
    units = trial.suggest_int(f"dense_units_{i}", 32, 256)
```

```
model.add(layers.Dense(units, activation='relu'))
```

```
model.add(layers.Dense(1))
```

```
return model
```

› Función objetivo para Optuna

[] ↴ 7 celdas ocultas

▼ Crear modelo con los hiperparámetros de optuna

En lugar de seleccionar directamente el mejor modelo sugerido por Optuna, se crearon y evaluaron múltiples modelos con diferentes hiperparámetros para comparar su rendimiento y determinar cuál predice mejor.

› Modelo 1

[] ↴ 8 celdas ocultas

› Modelo 2

[] ↴ 9 celdas ocultas

› Modelo 3

[] ↴ 9 celdas ocultas

› Modelo 4

[] ↴ 11 celdas ocultas

› Modelo 5

[] ↴ 9 celdas ocultas

> Modelo 6

[] ↴ 9 celdas ocultas

> Modelo 7

[] ↴ 11 celdas ocultas

> Modelo 8

[] ↴ 9 celdas ocultas

> Modelo 9

[] ↴ 9 celdas ocultas

> Modelo 10

[] ↴ 11 celdas ocultas

> Modelo 11

[] ↴ 11 celdas ocultas

> Modelo 12

[] ↴ 11 celdas ocultas

> Modelo 13

[] ↴ 11 celdas ocultas

> Modelo 14

[] ↴ 9 celdas ocultas

> Modelo 15

[] ↴ 11 celdas ocultas

> Modelo 16

[] ↴ 9 celdas ocultas

> Modelo 17

[] ↴ 9 celdas ocultas

> Modelo 18

[] ↴ 11 celdas ocultas

> Modelo 19

[] ↴ 9 celdas ocultas

> Modelo 20

[] ↴ 11 celdas ocultas

> Modelo 21

[] ↴ 9 celdas ocultas

> Modelo 22

[] ↴ 11 celdas ocultas

> Modelo 23

[] ↴ 11 celdas ocultas

> Modelo 24

[] ↴ 11 celdas ocultas

> Modelo 25

[] ↴ 9 celdas ocultas

> Modelo 26

[] ↴ 11 celdas ocultas

> Modelo 27

[] ↴ 9 celdas ocultas

> Modelo 28

[] ↴ 11 celdas ocultas

> Modelo 29

[] ↴ 9 celdas ocultas

> Modelo 30

▶ ↴ 11 celdas ocultas

▼ Pruebas Finales

▼ Procesamiento de Manchas solares de Junio

```
# Cargar el archivo CSV en un DataFrame usando `;` como delimitador
df_junio = pd.read_csv('/content/drive/MyDrive/Redes neuronales aplicado a ciclos solares/Ma
```

```
# Mostrar las primeras filas del DataFrame y los nombres de las columnas
print(df_junio.head())
print("Nombres de columnas:", df_junio.columns)
```

```
→      Fecha  No. de manchas solares
0  2024-06-01          181
1  2024-06-02          207
2  2024-06-03          212
3  2024-06-04          211
4  2024-06-05          182
Nombres de columnas: Index(['Fecha', 'No. de manchas solares'], dtype='object')
```

```
# Establecer la columna de fecha como el índice del DataFrame
df_junio.set_index('Fecha', inplace=True)
```

```
# Convertir el índice a formato de fecha
df_junio.index = pd.to_datetime(df_junio.index, format='%Y-%m-%d', errors='coerce')
```

```
print(df_junio.head())
```

```
→      No. de manchas solares
Fecha
2024-06-01          181
2024-06-02          207
2024-06-03          212
2024-06-04          211
2024-06-05          182
```

Cargar Manchas solares de Junio

▼ Cargar modelo y predecir 30 días

```
# Seleccionar solo la columna 'No. de manchas solares' del DataFrame original
df = Dataframe[['No. de manchas solares']]
```

```
# Escalar los datos a un rango de 0 a 1 para mejorar el rendimiento del modelo
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df)
```

```
# Convertir los datos escalados de nuevo a un DataFrame, manteniendo el mismo índice y nombr
```

```
df_scaled = pd.DataFrame(scaled_data, index=df.index, columns=['numero de manchas solares'])

# Determinar el tamaño del conjunto de entrenamiento y validación
train_size = int(len(df_scaled) * 0.6)
test_size = int(len(df_scaled) * 0.40)

# Separar los datos en conjuntos de entrenamiento y prueba
train_data = df_scaled[:train_size]
test_data = df_scaled[train_size:train_size + test_size]

# Verificar las longitudes
print("Longitud del conjunto total:", len(df_scaled))
print("Longitud del conjunto de entrenamiento:", len(train_data))
print("Longitud del conjunto de prueba:", len(test_data))

# Función para crear datos de secuencia
def create_sequence_data(data, seq_length):
    X = []
    y = []
    for i in range(len(data) - seq_length):
        X.append(data.iloc[i:i + seq_length].values)
        y.append(data.iloc[i + seq_length].values)
    return np.array(X), np.array(y)

seq_length = 2500

# Crear datos de entrenamiento y prueba
X_train, y_train = create_sequence_data(train_data, seq_length)
X_test, y_test = create_sequence_data(test_data, seq_length)

# Ajustar las dimensiones de las secuencias para que sean compatibles con el modelo
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

# Cargar el modelo desde un archivo HDF5
modelo = load_model('ruta/al/modelo.h5')

# Verifica que el modelo se ha cargado correctamente
modelo.summary()

# Crear una nueva secuencia de datos a partir de los últimos datos conocidos
last_known_sequence = test_data[-seq_length:].values # Última secuencia conocida del conjur
future_predictions = []

futuro = 30

# Realizar las predicciones paso a paso
for _ in range(futuro):
    # Preparar los datos de entrada para el modelo
    input_sequence = last_known_sequence.reshape((1, seq_length, 1))
```

```
# Realizar la predicción
predicted_value = loaded_model.predict(input_sequence)

# Almacenar la predicción
future_predictions.append(predicted_value[0, 0])

# Actualizar la secuencia de entrada añadiendo la nueva predicción y descartando el primero
last_known_sequence = np.append(last_known_sequence, predicted_value)[-seq_length:]

# Revertir la escala de las predicciones y redondear a enteros
future_predictions = scaler.inverse_transform(np.array(future_predictions).reshape(-1, 1))
future_predictions = np.round(future_predictions).astype(int)

# Crear un índice de tiempo para las predicciones futuras
last_date = df.index[-1]
future_dates = [last_date + pd.Timedelta(days=i) for i in range(1, futuro+1)]

# Crear un DataFrame de predicciones
df_predictions = pd.DataFrame(data=future_predictions, index=future_dates, columns=['numero de mancha solar'])
```

▼ Comparamos resultados

```
import pandas as pd
# Función para procesar el DataFrame
def procesar_dataframe(df):
    # Cambiar el nombre de la columna 'Unnamed: 0' a 'Fecha'
    df.rename(columns={'Unnamed: 0': 'Fecha'}, inplace=True)

    # Establecer la columna de fecha como el índice del DataFrame
    df.set_index('Fecha', inplace=True)

    # Convertir el índice a formato de fecha
    df.index = pd.to_datetime(df.index, format='%Y-%m-%d', errors='coerce')

    return df

# Cargar los DataFrames desde los archivos CSV
df_junio_4 = pd.read_csv('/content/drive/MyDrive/Redes neuronales aplicado a ciclos solares/junio_4.csv')
df_junio_7 = pd.read_csv('/content/drive/MyDrive/Redes neuronales aplicado a ciclos solares/junio_7.csv')
df_junio_10 = pd.read_csv('/content/drive/MyDrive/Redes neuronales aplicado a ciclos solares/junio_10.csv')
df_junio_11 = pd.read_csv('/content/drive/MyDrive/Redes neuronales aplicado a ciclos solares/junio_11.csv')
df_junio_12 = pd.read_csv('/content/drive/MyDrive/Redes neuronales aplicado a ciclos solares/junio_12.csv')
df_junio_13 = pd.read_csv('/content/drive/MyDrive/Redes neuronales aplicado a ciclos solares/junio_13.csv')
df_junio_15 = pd.read_csv('/content/drive/MyDrive/Redes neuronales aplicado a ciclos solares/junio_15.csv')
df_junio_18 = pd.read_csv('/content/drive/MyDrive/Redes neuronales aplicado a ciclos solares/junio_18.csv')
df_junio_20 = pd.read_csv('/content/drive/MyDrive/Redes neuronales aplicado a ciclos solares/junio_20.csv')
df_junio_22 = pd.read_csv('/content/drive/MyDrive/Redes neuronales aplicado a ciclos solares/junio_22.csv')
df_junio_23 = pd.read_csv('/content/drive/MyDrive/Redes neuronales aplicado a ciclos solares/junio_23.csv')
```

```
df_junio_24 = pd.read_csv('/content/drive/MyDrive/Redes neuronales aplicado a ciclos solares'
df_junio_26 = pd.read_csv('/content/drive/MyDrive/Redes neuronales aplicado a ciclos solares'
df_junio_28 = pd.read_csv('/content/drive/MyDrive/Redes neuronales aplicado a ciclos solares'
df_junio_30 = pd.read_csv('/content/drive/MyDrive/Redes neuronales aplicado a ciclos solares'

# Aplicar la función a todos los DataFrames
df_junio_4 = procesar_dataframe(df_junio_4)
df_junio_7 = procesar_dataframe(df_junio_7)
df_junio_10 = procesar_dataframe(df_junio_10)
df_junio_11 = procesar_dataframe(df_junio_11)
df_junio_12 = procesar_dataframe(df_junio_12)
df_junio_13 = procesar_dataframe(df_junio_13)
df_junio_15 = procesar_dataframe(df_junio_15)
df_junio_18 = procesar_dataframe(df_junio_18)
df_junio_20 = procesar_dataframe(df_junio_20)
df_junio_22 = procesar_dataframe(df_junio_22)
df_junio_23 = procesar_dataframe(df_junio_23)
df_junio_24 = procesar_dataframe(df_junio_24)
df_junio_26 = procesar_dataframe(df_junio_26)
df_junio_28 = procesar_dataframe(df_junio_28)
df_junio_30 = procesar_dataframe(df_junio_30)

import matplotlib.pyplot as plt

# Lista de DataFrames y nombres para las leyendas, incluyendo los datos reales
dataframes = [df_junio, df_junio_4, df_junio_7, df_junio_10, df_junio_11, df_junio_12, df_junio_13, df_junio_15, df_junio_18, df_junio_20, df_junio_22, df_junio_23, df_junio_24, df_junio_26, df_junio_28, df_junio_30]
nombres = ['Datos Reales', 'Modelo 4', 'Modelo 7', 'Modelo 10', 'Modelo 11', 'Modelo 12', 'Modelo 13', 'Modelo 15', 'Modelo 18', 'Modelo 20', 'Modelo 22', 'Modelo 23', 'Modelo 24', 'Modelo 26', 'Modelo 28', 'Modelo 30']

# Crear una figura y un eje
plt.figure(figsize=(15, 10))

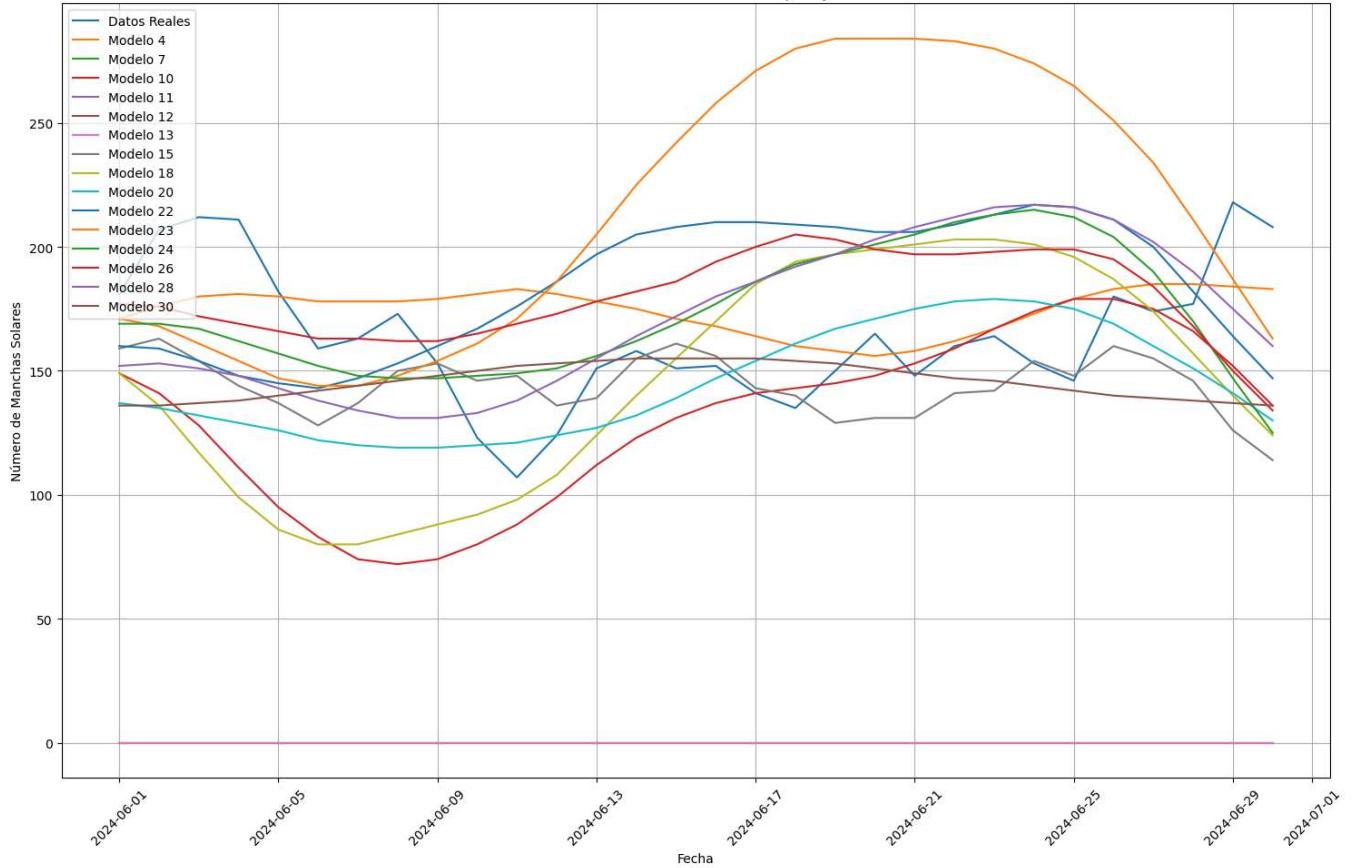
# Graficar los primeros 30 datos de cada DataFrame
for df, nombre in zip(dataframes, nombres):
    plt.plot(df.index[:30], df.iloc[:30, 0], label=nombre)

# Añadir títulos y etiquetas
plt.title('Predicciones de Manchas Solares para Junio')
plt.xlabel('Fecha')
plt.ylabel('Número de Manchas Solares')
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)

# Mostrar la gráfica
plt.tight_layout()
plt.show()
```



Predicciones de Manchas Solares para Junio



```
import matplotlib.pyplot as plt

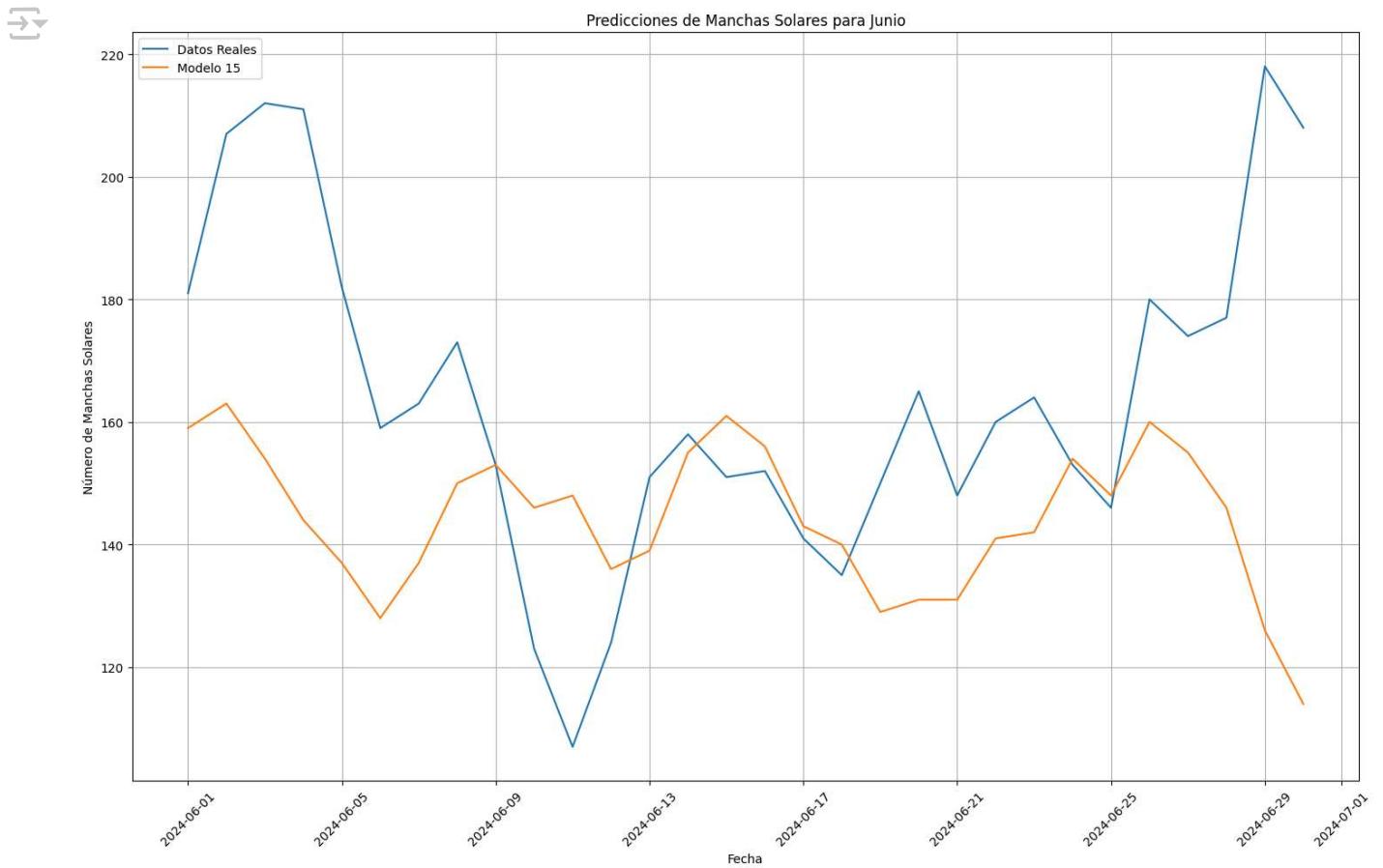
# Lista de DataFrames y nombres para las leyendas, incluyendo solo los modelos seleccionados
dataframes = [df_junio, df_junio_15]
nombres = ['Datos Reales', 'Modelo 15']

# Crear una figura y un eje
plt.figure(figsize=(15, 10))

# Graficar los primeros 30 datos de cada DataFrame
for df, nombre in zip(dataframes, nombres):
    plt.plot(df.index[:30], df.iloc[:30, 0], label=nombre)

# Añadir títulos y etiquetas
plt.title('Predicciones de Manchas Solares para Junio')
plt.xlabel('Fecha')
plt.ylabel('Número de Manchas Solares')
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)

# Mostrar la gráfica
plt.tight_layout()
plt.show()
```



```
# Lista de DataFrames y nombres para las leyendas, incluyendo los datos reales
dataframes = [df_junio, df_junio_4, df_junio_7, df_junio_10, df_junio_11, df_junio_12, df_junio_13]
nombres = ['Datos Reales', 'Modelo 4', 'Modelo 7', 'Modelo 10', 'Modelo 11', 'Modelo 12', 'Modelo 13']

# Crear una figura y un eje
plt.figure(figsize=(15, 10))

# Graficar los primeros 30 datos de cada DataFrame
for df, nombre in zip(dataframes, nombres):
    if nombre == 'Datos Reales':
        plt.plot(df.index[:30], df.iloc[:30, 0], label=nombre, color='black', linewidth=2)
    else:
        plt.plot(df.index[:30], df.iloc[:30, 0], linestyle='--', marker='o', label=nombre)

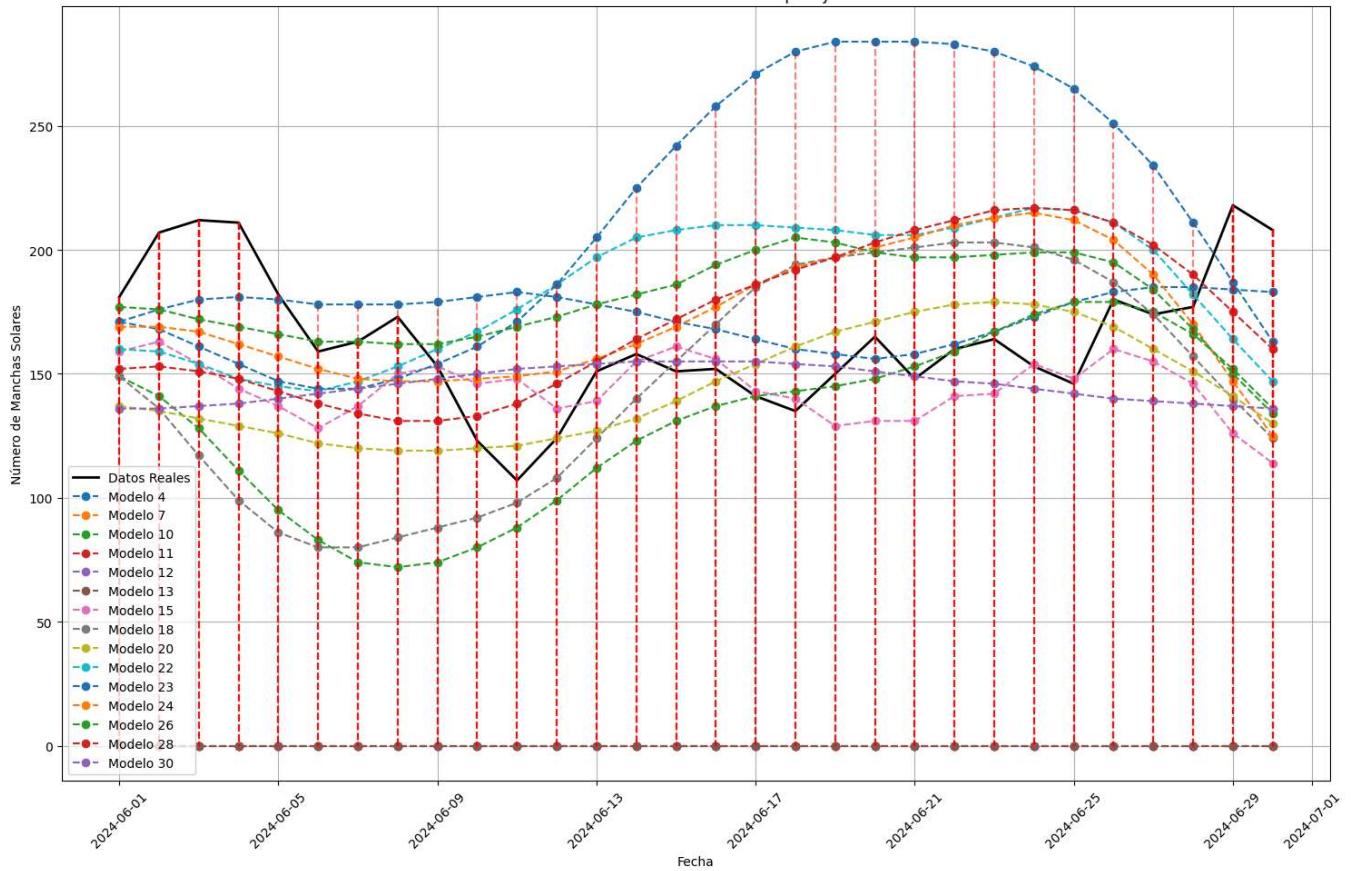
# Añadir líneas verticales desde los datos reales a cada predicción
for i in range(30):
    real_value = df_junio.iloc[i, 0]
    date = df_junio.index[i]
    for df in dataframes[1:]:
        pred_value = df.iloc[i, 0]
        plt.plot([date, date], [real_value, pred_value], 'r--', alpha=0.5)

# Añadir títulos y etiquetas
plt.title('Predicciones de Manchas Solares para Junio')
plt.xlabel('Fecha')
plt.ylabel('Número de Manchas Solares')
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)

# Mostrar la gráfica
plt.tight_layout()
plt.show()
```



Predicciones de Manchas Solares para junio



```
import pandas as pd
from sklearn.metrics import mean_absolute_error
import numpy as np

# Función para calcular el MAPE
def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

# Lista de DataFrames y nombres para las leyendas, incluyendo los datos reales
dataframes = [df_junio_4, df_junio_7, df_junio_10, df_junio_11, df_junio_12, df_junio_13, df_junio_14]
nombres = ['Modelo 4', 'Modelo 7', 'Modelo 10', 'Modelo 11', 'Modelo 12', 'Modelo 13', 'Modelo 14']

# Calcular MAE, MAPE, precisión y contar valores correctos para cada modelo
mae_scores = {}
mape_scores = {}
accuracy_scores = {}
correct_predictions_count = {}
correct_dates = {}

for df, nombre in zip(dataframes, nombres):
    # Calcular MAE
    mae = mean_absolute_error(df_junio.iloc[:30, 0], df.iloc[:30, 0])
    mae_scores[nombre] = mae

    # Calcular MAPE
    mape = mean_absolute_percentage_error(df_junio.iloc[:30, 0], df.iloc[:30, 0])
    mape_scores[nombre] = mape

    # Calcular precisión
    accuracy = 100 - mape
    accuracy_scores[nombre] = accuracy

    # Contar valores correctos
    correct_predictions = np.round(df.iloc[:30, 0]) == np.round(df_junio.iloc[:30, 0])
    correct_count = np.sum(correct_predictions)
    correct_predictions_count[nombre] = correct_count

    if correct_count > 0:
        # Almacenar las fechas y valores correctos
        correct_dates[nombre] = df_junio.index[:30][correct_predictions].strftime('%Y-%m-%d')
        correct_values = df_junio.iloc[:30, 0][correct_predictions].tolist()

# Mostrar resultados
print("MAE de cada modelo:")
for nombre, mae in mae_scores.items():
    print(f"{nombre}: MAE = {mae:.4f}")

print("\nMAPE de cada modelo:")
for nombre, mape in mape_scores.items():
    print(f"{nombre}: MAPE = {mape:.2f}%")
```

```
print("\nPorcentaje de acierto de cada modelo:")
for nombre, accuracy in accuracy_scores.items():
    print(f"{nombre}: Precisión = {accuracy:.2f}%")

print("\nNúmero de valores correctos de cada modelo:")
for nombre, correct_count in correct_predictions_count.items():
    print(f"{nombre}: Valores correctos = {correct_count}")

print("\nModelos con al menos un valor correcto y las fechas correspondientes:")
for nombre, dates in correct_dates.items():
    print(f"\n{nombre} acertó en las siguientes fechas:")
    for date in dates:
        value = df_junio.loc[date].values[0] # Obtener el valor de manchas solares en esa fecha
        print(f" - {date}: {value} manchas solares")

# Encontrar el mejor modelo basado en MAE y MAPE
mejor_modelo_mae = min(mae_scores, key=mae_scores.get)
mejor_modelo_mape = min(mape_scores, key=mape_scores.get)

print(f"\nEl modelo con el MAE más bajo es {mejor_modelo_mae} con un MAE de {mae_scores[mejor_modelo_mae]}")
print(f"El modelo con el MAPE más bajo es {mejor_modelo_mape} con un MAPE de {mape_scores[mejor_modelo_mape]}")
```

→ MAE de cada modelo:

Modelo 4: MAE = 21.8333
Modelo 7: MAE = 164.2000
Modelo 10: MAE = 38.4667
Modelo 11: MAE = 164.2000
Modelo 12: MAE = 164.2000
Modelo 13: MAE = 164.2000
Modelo 15: MAE = 26.6667
Modelo 18: MAE = 48.3667
Modelo 20: MAE = 32.2667
Modelo 22: MAE = 45.9333
Modelo 23: MAE = 70.6000
Modelo 24: MAE = 34.6667
Modelo 26: MAE = 34.9667
Modelo 28: MAE = 37.6667
Modelo 30: MAE = 28.3333

MAPE de cada modelo:

Modelo 4: MAPE = 14.66%
Modelo 7: MAPE = 100.00%
Modelo 10: MAPE = 22.49%
Modelo 11: MAPE = 100.00%
Modelo 12: MAPE = 100.00%
Modelo 13: MAPE = 100.00%
Modelo 15: MAPE = 15.24%
Modelo 18: MAPE = 28.59%
Modelo 20: MAPE = 18.29%
Modelo 22: MAPE = 29.35%
Modelo 23: MAPE = 45.66%
Modelo 24: MAPE = 21.45%

Modelo 26: MAPE = 22.46%
Modelo 28: MAPE = 23.06%
Modelo 30: MAPE = 16.28%

Porcentaje de acierto de cada modelo:

Modelo 4: Precisión = 85.34%
Modelo 7: Precisión = 0.00%
Modelo 10: Precisión = 77.51%
Modelo 11: Precisión = 0.00%
Modelo 12: Precisión = 0.00%
Modelo 13: Precisión = 0.00%
Modelo 15: Precisión = 84.76%
Modelo 18: Precisión = 71.41%
Modelo 20: Precisión = 81.71%
Modelo 22: Precisión = 70.65%
Modelo 23: Precisión = 54.34%
Modelo 24: Precisión = 78.55%
Modelo 26: Precisión = 77.54%
Modelo 28: Precisión = 76.94%
Modelo 30: Precisión = 83.72%

Número de valores correctos de cada modelo:

Modelo 4: Valores correctos = 0
Modelo 7: Valores correctos = 0
Modelo 10: Valores correctos = 1
Modelo 11: Valores correctos = 0
Modelo 12: Valores correctos = 0
Modelo 13: Valores correctos = 0

```
from sklearn.metrics import mean_absolute_error

# Lista de DataFrames y nombres para las leyendas, incluyendo los datos reales
dataframes = [df_junio_4, df_junio_7, df_junio_10, df_junio_11, df_junio_12, df_junio_13, df
nombres = ['Modelo 4', 'Modelo 7', 'Modelo 10', 'Modelo 11', 'Modelo 12', 'Modelo 13', 'Mode

# Calcular el MAE para cada modelo
mae_scores = {}
for df, nombre in zip(dataframes, nombres):
    mae = mean_absolute_error(df_junio.iloc[:30, 0], df.iloc[:30, 0])
    mae_scores[nombre] = mae

# Mostrar los resultados
for nombre, mae in mae_scores.items():
    print(f"{nombre}: MAE = {mae}")

# Encontrar el modelo con el MAE más bajo
mejor_modelo = min(mae_scores, key=mae_scores.get)
print(f"\nEl modelo con las predicciones más cercanas a los datos reales es {mejor_modelo} c
```

→ Modelo 4: MAE = 21.833333333333332
Modelo 7: MAE = 164.2
Modelo 10: MAE = 38.466666666666667
Modelo 11: MAE = 164.2

```
Modelo 12: MAE = 164.2
Modelo 13: MAE = 164.2
Modelo 15: MAE = 26.666666666666668
Modelo 18: MAE = 48.36666666666667
Modelo 20: MAE = 32.266666666666666
Modelo 22: MAE = 45.9333333333333
Modelo 23: MAE = 70.6
Modelo 24: MAE = 34.666666666666664
Modelo 26: MAE = 34.966666666666667
Modelo 28: MAE = 37.666666666666664
Modelo 30: MAE = 28.33333333333332
```

El modelo con las predicciones más cercanas a los datos reales es Modelo 4 con un MAE de

```
import pandas as pd
from sklearn.metrics import mean_absolute_error
import numpy as np

# Función para calcular el MAPE
def mean_absolute_percentage_error(y_true, y_pred):
```