CODIGO APENDICE 1

```
Importación de librerías
import pandas as pd  # para cargar el Excel
import numpy as np  # para operaciones con vectores
import matplotlib.pyplot as plt  # para gráficos
import matplotlib.dates as mdates  # para formatear y manejar las fechas en los
gráficos
from scipy.stats import zscore, skew, kurtosis
import seaborn as sns  # para gráficos
import xgboost as xgb
from xgboost import XGBRegressor
from sklearn.model_selection import TimeSeriesSplit, RandomizedSearchCV
from itertools import cycle  # Para crear iteradores cíclicos
from sklearn.metrics import mean_squared_error, mean_absolute_error
from statsmodels.tsa.api import ExponentialSmoothing
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
Definición de funciones
# Se genera un vector cíclico, para la estacionalidad del ciclo solar:
def generate_vector(length, n_to_repeat):
values = list(range(1, n_to_repeat+1))
vector = []
cycle_iterator = cycle(values)
for _ in range(length):
vector.append(next(cycle_iterator))
return vector

# Se crean los títulos del DataFrame, mostrando la relación temporal (`t-n`, `t`,
`t+n`):
def Extended_titles(Cn,n_in,n_out):
Total_titles=[]
Tl=n_in+n_out+1
for i in range(Tl-1):
if i<(n_in-1):
letter_to_add = " (t-" + str(n_in-i-1) +")"
elif i==(n_in-1):
letter_to_add = " (t)"
else:
letter_to_add = " (t+" + str(i-n_in+1) +")"
Cn_aux= [word + letter_to_add for word in Cn]
Total_titles=Total_titles+Cn_aux
return(Total_titles)
```

```
# Se convierte la serie temporal en el formato adecuado para el aprendizaje automático:
def series_to_supervised(data, n_in, n_out, dropnan=True):
n_vars = 1 if type(data) is list else data.shape[1]
df = pd.DataFrame(data)
cols = list()
for i in range(n_in, 0, -1):
cols.append(df.shift(i))
for i in range(0, n_out):
cols.append(df.shift(-i))
agg = pd.concat(cols, axis=1)
if dropnan:
agg.dropna(inplace=True)
return agg.values

# Se transforman los datos de la serie temporal en un formato adecuado para el aprendizaje supervisado
def to_supervided2(yiel, n_in, n_out):
values = yiel.values
data = series_to_supervised(values, n_in, n_out)
Cn=list(yiel.columns.values)
Cn_total=Extended_titles(Cn, n_in, n_out)
Data = pd.DataFrame(data, columns=Cn_total)
return Data
Carga de los datos
Df = pd.read_json("C:/Users/Letia/Desktop/TFG_2024/PYTHON/observed-solar-cycle-indices.json")
Cálculo de las estadísticas descriptivas
#Media
mean_values = Df[['ssn','smoothed_ssn']].mean()
print("Media:\n", mean_values)

#Desviación estandar
std_dev = Df[['ssn','smoothed_ssn']].std()
print("Desviación Estándar:\n", std_dev)

#Varianza
variance = Df[['ssn','smoothed_ssn']].var()
print("Varianza:\n", variance)

#Sesgo
skewness = Df[['ssn','smoothed_ssn']].apply(lambda x: skew(x.dropna()))
print("Sesgo (Skewness):\n", skewness)
```

```
#Curtosis: Mide la forma de la distribución, especialmente de las colas
kurtosis_vals = Df[['ssn','smoothed_ssn']].apply(lambda x: kurtosis(x.dropna()))
print("Curtosis (Kurtosis):\n", kurtosis_vals)

#Cuartiles
quartiles = Df[['ssn','smoothed_ssn']].quantile([0.25, 0.5, 0.75])
print("Cuartiles:\n", quartiles)

#Rango intercuartílico
iqr = quartiles.loc[0.75] - quartiles.loc[0.25]
print("Rango Intercuartílico (IQR):\n", iqr)

#Rango
range_values = Df[['ssn','smoothed_ssn']].apply(lambda x: x.max() - x.min())
print("Rango:\n", range_values)
Visualizaciones iniciales
# Histograma del número de manchas solares
plt.figure(figsize=(12, 6))
plt.hist(Df['ssn'], bins=30, edgecolor='k', alpha=0.7)
plt.xlabel('Número de Manchas Solares')
plt.ylabel('Frecuencia')
plt.title('Distribución del Número de Manchas Solares')
plt.grid(True)
plt.show()

# Gráfico de dispersión
Df['Date'] = pd.to_datetime(Df['time-tag'])
Df = Df.sort_values(by='Date')

# Cálculo de la Media Móvil y el Z-Score
Df['SMA'] = Df['ssn'].rolling(window=12, center=True).mean()
Df['Z-Score'] = zscore(Df['ssn'])

# Visualización
plt.figure(figsize=(12, 8))
plt.scatter(Df['Date'], Df['ssn'], color='blue', alpha=0.6, label='Número de
Manchas Solares')
plt.plot(Df['Date'], Df['SMA'], color='red', label='Media Móvil (12 meses)')
plt.scatter(Df['Date'], Df['ssn'], c=abs(Df['Z-Score']), cmap='coolwarm',
label='Z-Score', alpha=0.6, edgecolors='k')
plt.gca().xaxis.set_major_locator(mdates.YearLocator(11))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
plt.xticks(rotation=45)
```

```
plt.xlabel('Fecha')
plt.ylabel('Número de Manchas Solares')
plt.title('Número de Manchas Solares a lo Largo del Tiempo')
plt.colorbar(label='Z-Score')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Matriz de correlación
Df_numeric = Df.select_dtypes(include=[np.number])
Df_numeric = Df_numeric.fillna(Df_numeric.mean())
correlation_matrix = Df_numeric.corr()
print(correlation_matrix)

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1,
vmax=1)
plt.title('Matriz de Correlación')
plt.show()
Comparación por periodos mediante matrices de correlación
# Se ajustan los formatos de las fechas y nos quedamos con las columnas que
queremos
Df['Date'] = pd.to_datetime(Df['Date'], errors='coerce')
Df.set_index('Date', inplace=True)

# Se asegura de que sólo las columnas numéricas se utilicen para la correlación
Df_numeric = Df.select_dtypes(include=[np.number])
Df_numeric = Df_numeric.fillna(Df_numeric.mean())

# Se definen los periodos para dividir la serie temporal
periodos = pd.date_range(start=Df_numeric.index.min(),
end=Df_numeric.index.max(), freq='11Y')
correlation_matrices = {}

for i in range(len(periodos)-1):
start_date = periodos[i]
end_date = periodos[i+1]
Df_periodo = Df_numeric[start_date:end_date]
if not Df_periodo.empty:
corr_matrix = Df_periodo.corr()
correlation_matrices[f"{start_date.year}-{end_date.year}"] = corr_matrix
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
```

```
plt.title(f"Matriz de Correlación {start_date.year}-{end_date.year}")
plt.show()

Df = Df.rename(columns={'ssn': 'Sunspot_Number', 'smoothed_ssn':
'Smoothed_Sunspot_Number'})
Df['Date'] = pd.to_datetime(Df['time-tag'])
Df.insert(0, 'Date', Df.pop('Date'))
Df = Df[["Date", "Sunspot_Number", "Smoothed_Sunspot_Number"]]
Df.replace(-1, np.nan, inplace=True)
Visualización inicial de los datos: Sunspot Number vs Smoothed Sunspot
Number
fig, ax = plt.subplots(figsize=(15, 6))
ax.plot(Df['Date'], Df['Sunspot_Number'], color='blue', linestyle='-', linewidth=0.75,
label='Sunspot Number')
ax.plot(Df['Date'], Df['Smoothed_Sunspot_Number'], color='red', linestyle='-',
linewidth=1, label='Smoothed Sunspot Number')
ax.set_title('Número de manchas solares a lo largo del tiempo')
ax.set_xlabel('Fecha')
ax.set_ylabel('Número de manchas solares')
plt.xticks(rotation=45)
ax.grid(True)
ax.legend()
plt.show()
Preparación de datos para el modelado
# Transformación de los datos para el aprendizaje supervisado
n_in=24 # número de observaciones anteriores utilizadas para predecir
n_out=12 # número de observaciones anteriores utilizadas a predecir

Df=to_supervided2(Df,n_in,n_out) #Df listo para el aprendizaje supervisado
Df = Df.dropna(subset=[f'Date (t-{n_in-1})']) #Se eliminan las filas que no nos
valen
Df.set_index('Date (t)', inplace=True)



# Se revisa el rango de fechas después de la transformación
print("Rango de fechas en el DataFrame supervisado:")
print(Df.index.min(), "a", Df.index.max())



n_to_repeat=11*12 # 11 años y 12 meses tiene cada ciclo aproximadamente
v_sta = generate_vector(len(Df), n_to_repeat) #se genera el vector a repetir

Df.insert(0, 'Index_for_seasonality', v_sta)
Separación entre "target" y "features"
```

```
Div_index=Df.columns.get_loc("Smoothed_Sunspot_Number (t)")

features=Df.iloc[:,:Div_index+1].columns.to_list()
features=[element for element in features if "Date" not in element]

targets=Df.iloc[:,Div_index:].columns.to_list()
targets=[element for element in targets if "Sunspot_Number" in element]
targets=[element for element in targets if "Smoothed" not in element]


X=Df[features]
y=Df[targets]

X = X.apply(pd.to_numeric, errors='coerce')
y = y.apply(pd.to_numeric, errors='coerce')


Cut_Date=pd.Timestamp(1971,1,1)

X_train=X[X.index<Cut_Date]
y_train=y[y.index<Cut_Date]


X_test=X[X.index>=Cut_Date]
y_test=y[y.index>=Cut_Date]
Validación cruzada y entrenamiento del modelo XGBoost
#Se inicializa el modelo XGBoost
XGB_model = XGBRegressor()

# Se definen los hiperparámetros a buscar en RandomizedSearchCV
param_grid = {
        'learning_rate': [0.001, 0.01],    # Tasa de aprendizaje
        'max_depth': [3, 5],               # Profundidad máxima del árbol
        'min_child_weight': [1, 3],        # Peso mínimo de un nodo hijo
        'n_estimators': [100, 300],        # Número de árboles (estimadores)
        'lambda': [0.1, 1],               # Regularización L2
        'gamma': [0, 0.1]                  # Reducción mínima de pérdida
}

# Se define el esquema de validación cruzada
tscv = TimeSeriesSplit(n_splits=3)

# Configuración de RandomizedSearchCV con la validación cruzada
random_search = RandomizedSearchCV(
```

```
        estimator=XGB_model,
        param_distributions=param_grid,
        n_iter=50,                    # Número de combinaciones a probar
        cv=tscv,                      # Validación cruzada con TimeSeriesSplit
        scoring="neg_mean_squared_error",     # Métrica de evaluación
        random_state=42,
        n_jobs=-1,
        verbose=2
        )

        # Se buscan de los mejores hiperparámetros
        random_result = random_search.fit(X_train, y_train)

        # Se obtienen los mejores hiperparámetros
        best_params = random_result.best_params_

        # Se entrena el modelo con los mejores parámetros
        XGB_model = XGBRegressor(**best_params)
        XGB_model.fit(X_train, y_train)
        Predicción del Sunspot Number y posterior comparación datos históricos
        # Se realizan las predicciones
        y_test_pred = XGB_model.predict(X_test)

        # Se convierte y_test_pred en un DataFrame con la fecha y la predicción

        # La primera columna de y_test_pred es la predicción para 'Sunspot Number'
        sunspot_number_pred = y_test_pred[:, 0]  # Se selecciona la primera columna

        # X_test tiene el índice de fechas que se quiere usar
        prediction_dates = X_test.index + pd.DateOffset(months=12) # Se asume que el
índice de X_test es la fecha

        #El índice de X_test es la fecha real (t), no (t+12)
        prediction_dates = X_test.index  # Las fechas no se desplazan, ya que ya
corresponden a t+12 en X_test

        # Se crea un DataFrame con las fechas y las predicciones
        y_test_pred_Df = pd.DataFrame({
                'Date (t+12)': prediction_dates,
                'Sunspot_Number (t+12)_pred': sunspot_number_pred
        })

        # Se crea la figura y los ejes
        plt.figure(figsize=(12, 6))
```

```
		# Se grafican los valores reales usando la fecha
		plt.plot(y_test.index, y_test['Sunspot_Number (t+12)'], label='Valor Real', color='blue', marker='o')

		# Se grafican las predicciones usando la fecha
		plt.plot(y_test_pred_Df['Date (t+12)'], y_test_pred_Df['Sunspot_Number (t+12)_pred'], label='Predicción', color='red', marker='o')

		# Se agrega el título y las etiquetas a los ejes
		plt.title('Comparación de Valores Reales y Predicciones de Manchas Solares')
		plt.xlabel('Fecha')
		plt.ylabel('Número de Manchas Solares')

		#Se muestra la leyenda
		plt.legend()

		#Se muestra la gráfica
		plt.grid(True)
		plt.show()

		# Esto supone que la primera predicción corresponde a la primera fecha en y_test_pred_Df
		y_test_pred_Df.index = y_test.index

		#Se crea un DataFrame combinando los valores reales y las predicciones
		comparison_df = pd.DataFrame({
			'Date (t+12)': y_test.index,
			'Sunspot_Number (t+12)_Real': y_test['Sunspot_Number (t+12)'],
			'Sunspot_Number (t+12)_Pred': y_test_pred_Df['Sunspot_Number (t+12)_pred']
		})

		#Se define la ruta donde se quiere guardar el archivo
		file_path = 'comparacion_predicciones.xlsx'

		#Se exporta el DataFrame a un archivo Excel
		comparison_df.to_excel(file_path, index=False)
		Evaluación del rendimiento del modelo
		#TÉCNICAS DE EVALUACION DEL RENDIMIENTO DEL MODELO

		#Error Absoluto Medio

		# Paso 1: Se calcula el MAE del modelo
```

```
                mae = mean_absolute_error(y_test['Sunspot_Number (t+12)'],
y_test_pred_Df['Sunspot_Number (t+12)_pred'])

                # Paso 2: Se calcula la media de Sunspot_Number (t+12) en el conjunto de
entrenamiento
                mean_sunspot_number = y_train['Sunspot_Number (t+12)'].mean()

                # Paso 3: Se crean predicciones del baseline utilizando la media calculada
                baseline_predictions = [mean_sunspot_number] * len(y_test)

                # Paso 4: Se calcula el MAE del baseline
                baseline_mae = mean_absolute_error(y_test['Sunspot_Number (t+12)'],
baseline_predictions)

                # Se compara con el MAE del modelo
                print(f'MAE del modelo: {mae}')
                print(f'MAE del baseline: {baseline_mae}')

                # Se calcula la mejora porcentual del modelo sobre el baseline
                improvement = (baseline_mae - mae) / baseline_mae
                print(f'Mejoría del modelo sobre el baseline: {improvement:.2%}')

                #Error Cuadrático Medio

                # Paso 1: Se calcula el MSE del modelo
                mse_model = mean_squared_error(y_test['Sunspot_Number (t+12)'],
y_test_pred_Df['Sunspot_Number (t+12)_pred'])

                # Paso 2: Se calcula la media de la variable objetivo en el conjunto de prueba
                mean_sunspot_number = y_test['Sunspot_Number (t+12)'].mean()

                # Paso 3: Se crea predicciones del baseline utilizando la media calculada
                baseline_predictions = [mean_sunspot_number] * len(y_test)

                # Paso 4: Se calcula el MSE del baseline
                mse_baseline = mean_squared_error(y_test['Sunspot_Number (t+12)'],
baseline_predictions)

                # Se compara con el MSE del modelo
                print(f'MSE del modelo: {mse_model}')
                print(f'MSE del baseline: {mse_baseline}')

                # Se calcula la mejora porcentual del modelo sobre el baseline
                improvement = (mse_baseline - mse_model) / mse_baseline
```

```
print(f'Mejoría del modelo sobre el baseline: {improvement:.2%}')

#Raiz del Error Cuadrático Medio

# Paso 1: Se calcula el RMSE del modelo
rmse = mean_squared_error(y_test['Sunspot_Number (t+12)'],
y_test_pred_Df['Sunspot_Number (t+12)_pred'], squared=False)

# Paso 2: Se calcula el RMSE del baseline
baseline_predictions = [y_test['Sunspot_Number (t+12)'].mean()] * len(y_test)
rmse_baseline = mean_squared_error(y_test['Sunspot_Number (t+12)'],
baseline_predictions, squared=False)

# Se compara con el MSE del modelo
print(f'RMSE del modelo: {rmse}')
print(f'RMSE del baseline: {rmse_baseline}')

#Se calcula el rango de valores
range_y_test = y_test['Sunspot_Number (t+12)'].max() - y_test['Sunspot_Number
(t+12)'].min()
print(f'Rango de los valores: {range_y_test}')


#CREACIÓN TABLA ESTADÍSTICAS DESCRIPTIVAS

#Se crea un DataFrame con los resultados
stats_Df = pd.DataFrame({
    'Estadística': [
    'Media',
    'Desviación Estándar',
    'Varianza',
    'Sesgo (Skewness)',
    'Curtosis (Kurtosis)',
    'Cuartil Q1',
    'Mediana (Q2)',
    'Cuartil Q3',
    'Rango Intercuartílico (IQR)',
    'Rango'
    ],
    'Valor': [
    mean_values,
    std_dev,
    variance,
    skewness,
```

```
            kurtosis_vals,
            quartiles.loc[0.25],  # Cuartil Q1
            quartiles.loc[0.5],   # Mediana (Q2)
            quartiles.loc[0.75],  # Cuartil Q3
            iqr,
            range_values
            ]
})

# Se convierte el diccionario a un DataFrame
stats_Df = pd.DataFrame(stats_Df)

# Se convierte el diccionario a un DataFrame
stats_Df = pd.DataFrame(stats_Df)

# Se guarda el DataFrame en un archivo Excel
file_path = 'estadisticas_descriptivas.xlsx'
stats_Df.to_excel(file_path, index=False)
Comparativas modelo XGBoost con otros modelos
# Se vuelven a cargar los datos
Df_comparativa = pd.read_json('observed-solar-cycle-indices.json')
Df_comparativa = Df_comparativa.sort_values('time-tag')

# Se define el tamaño del conjunto de prueba
test_size = int(len(Df_comparativa) * 0.2)  # 20% para prueba

# Se calculan los índices
train_indices = list(range(len(Df_comparativa) - test_size))
test_indices = list(range(len(Df_comparativa) - test_size, len(Df_comparativa)))

# Se dividen los datos en conjuntos de entrenamiento y prueba
train_data = Df_comparativa.iloc[train_indices]
test_data = Df_comparativa.iloc[test_indices]

# Se define el tamaño de la ventana para modelos comparativos
window_size = 12

# Se define y_test (valores reales de prueba)
y_test = test_data['ssn']

# MODELO DE LA MEDIA MÓVIL
y_train_ma = Df_comparativa['ssn'].iloc[train_indices]
y_test_ma = Df_comparativa['ssn'].iloc[test_indices]
```

```
        # MEDIA MÓVIL SIMPLE
        y_pred_ma =
y_test_ma.rolling(window=window_size).mean().shift(-window_size+1)
        y_pred_ma = y_pred_ma.dropna()

        # Se ajusta y_test_ma para que coincida con la longitud de y_pred_ma
        y_test_ma = y_test_ma.loc[y_pred_ma.index]
        mse_ma = mean_squared_error(y_test_ma, y_pred_ma)
        rmse_ma = mean_squared_error(y_test_ma, y_pred_ma, squared=False)
        print(f'MSE del modelo de Media Móvil: {mse_ma}')
        print(f'RMSE del modelo de Media Móvil: {rmse_ma}')

        # MODELO DE PROMEDIO EXPONENCIAL
        y_train_es = Df_comparativa['ssn'].iloc[train_indices]
        y_test_es = Df_comparativa['ssn'].iloc[test_indices]

        # Ajuste del modelo Exponential Smoothing
        es_model = ExponentialSmoothing(y_train_es, trend='add', seasonal='add',
seasonal_periods=window_size)
        es_fit = es_model.fit()
        y_pred_es = es_fit.forecast(len(y_test_es))

        # Se asegura que y_test_es coincida con la longitud de y_pred_es
        y_test_es = y_test_es.loc[y_pred_es.index]
        mse_es = mean_squared_error(y_test_es, y_pred_es)
        rmse_es = mean_squared_error(y_test_es, y_pred_es, squared=False)
        print(f'MSE del modelo de Promedio Exponencial: {mse_es}')
        print(f'RMSE del modelo de Promedio Exponencial: {rmse_es}')

        print(f'MSE del modelo XGBoost: {mse}')
        print(f'RMSE del modelo XGBoost: {rmse}')
```