

# Carga y análisis descriptivo inicial

[ [Volver al índice](#) ]

---

## Importamos archivo loader.py (importa los datos)

[ [Volver al índice](#) ]

---

```
%run loader.py
import matplotlib.pyplot as plt
import numpy as np

display(climate_data)
display(soil_min1_data)
display(soil_mini2_data)

      Fecha  Índice de calor  Humedad relativa \
0  2023-01-01 00:00:00      11.6          51.0
1  2023-01-01 00:15:00      11.2          52.0
2  2023-01-01 00:30:00      11.1          58.0
3  2023-01-01 00:45:00      11.7          54.0
4  2023-01-01 01:00:00      11.3          56.0
...
96996 2025-10-08 22:45:00      15.4          80.0
96997 2025-10-08 23:00:00      15.3          80.0
96998 2025-10-08 23:15:00      15.2          80.0
96999 2025-10-08 23:30:00      15.2          80.0
97000 2025-10-08 23:45:00      15.2          81.0

      Presión de vapor  Radiación solar  Temperatura  Punto de rocío \
0  14.160000          0.0          11.6          1.720000
1  13.810000          0.0          11.2          1.620000
2  13.720000          0.0          11.1          3.070000
```

3	14.250000	0.0	11.7	2.620000
4	13.890000	0.0	11.3	2.760000
...	...	...	...	...
96996	17.955096	0.0	15.4	11.937903
96997	17.844727	0.0	15.3	11.840378
96998	17.734961	0.0	15.2	11.742854
96999	17.734961	0.0	15.2	11.742854
97000	17.734961	0.0	15.2	11.932791
Temperatura del bulbo húmedo \				
0	7.280329		10.0	
1	7.038901		7.0	
2	7.501979		7.0	
3	7.646474		13.0	
4	7.491661		11.0	
...	...		...	
96996	13.471570		6.0	
96997	13.377991		6.0	
96998	13.284236		5.0	
96999	13.284323		6.0	
97000	13.382955		6.0	
Racha de viento máxima \				
0	17.0		180.0	
1	10.0		225.0	
2	12.0		180.0	
3	19.0		180.0	
4	20.0		180.0	
...	...		...	
96996	10.0		22.5	
96997	12.0		315.0	
96998	10.0		45.0	
96999	12.0		0.0	
97000	15.0		90.0	
Humectación en hoja (porcentaje) \				
Precipitaciones			Presión atmosférica	
0	0.0		940.6	
0.0				
1	0.0		940.8	
0.0				
2	0.0		940.8	

0.0		
3	0.0	940.7
0.0		
4	0.0	940.8
0.0		
...	...	...
...		
96996	0.0	939.7
0.0		
96997	0.0	940.0
0.0		
96998	0.0	940.1
0.0		
96999	0.0	940.2
0.0		
97000	0.0	940.1
0.0		

[97001 rows x 14 columns]

	Fecha	Contenido volumétrico (10 cm)	\
0	2023-01-01 00:00:00		14.0
1	2023-01-01 00:15:00		14.0
2	2023-01-01 00:30:00		14.0
3	2023-01-01 00:45:00		14.0
4	2023-01-01 01:00:00		13.9
...	...		...
89883	2025-10-08 22:45:00		28.7
89884	2025-10-08 23:00:00		28.7
89885	2025-10-08 23:15:00		28.6
89886	2025-10-08 23:30:00		28.6
89887	2025-10-08 23:45:00		28.6

	Contenido volumétrico (20 cm)	Contenido volumétrico (40 cm)	\
0	29.5		30.7
1	29.5		30.7
2	29.5		30.7
3	29.5		30.7
4	29.5		30.7
...	...		...
89883	27.6		31.1
89884	27.6		31.1
89885	27.6		31.1
89886	27.6		31.1
89887	27.6		31.1

	Contenido volumétrico (60 cm)	Agua útil	Temperatura en suelo
(10 cm)	\		
0	33.9	66.580874	
10.8			

1	33.9	66.580874
10.7		
2	33.9	66.580874
10.6		
3	33.9	66.580874
10.6		
4	33.9	66.543074
10.6		
...	...	...
...		
89883	32.7	67.305117
18.6		
89884	32.7	67.267317
18.5		
89885	32.7	67.167317
18.4		
89886	32.7	67.167317
18.3		
89887	32.7	67.167317
18.3		

	Temperatura en suelo (20 cm)	Temperatura en suelo (40 cm) \
0	10.8	11.4
1	10.8	11.4
2	10.8	11.4
3	10.8	11.4
4	10.8	11.4
...	...	...
89883	18.8	18.4
89884	18.8	18.4
89885	18.7	18.4
89886	18.6	18.4
89887	18.6	18.4

	Temperatura en suelo (60 cm)
0	11.4
1	11.4
2	11.4
3	11.4
4	11.4
...	...
89883	17.9
89884	17.9
89885	17.9
89886	17.9
89887	17.9

[89888 rows x 10 columns]

		Fecha	Contenido volumétrico (10 cm)	\
0	2023-01-01	00:00:00	20.3	
1	2023-01-01	00:15:00	20.3	
2	2023-01-01	00:30:00	20.3	
3	2023-01-01	00:45:00	20.3	
4	2023-01-01	01:00:00	20.3	
...	...	...	...	...
83359	2025-10-08	22:45:00	25.4	
83360	2025-10-08	23:00:00	25.4	
83361	2025-10-08	23:15:00	25.4	
83362	2025-10-08	23:30:00	25.3	
83363	2025-10-08	23:45:00	25.3	
		Contenido volumétrico (20 cm)	Contenido volumétrico (40 cm)	\
0		25.0	21.9	
1		25.0	21.9	
2		25.0	21.9	
3		25.0	21.9	
4		25.0	21.9	
...	...	...	...	...
83359		24.4	25.8	
83360		24.4	25.8	
83361		24.4	25.8	
83362		24.4	25.8	
83363		24.4	25.8	
		Contenido volumétrico (60 cm)	Aqua útil	Temperatura en suelo
(10 cm)	\			
0		21.7	47.413496	
9.9				
1		21.7	47.413496	
9.9				
2		21.7	47.413496	
9.9				
3		21.7	47.413496	
9.9				
4		21.7	47.413496	
9.9				
...	...	...	...	...
...	...	...	...	...
83359		18.8	53.913496	
18.7				
83360		18.8	53.913496	
18.6				
83361		18.8	53.913496	
18.5				
83362		18.8	53.813496	
18.4				
83363		18.8	53.813496	
18.4				

```
Temperatura en suelo (20 cm) Temperatura en suelo (40 cm) \
0 10.6 11.4
1 10.6 11.4
2 10.6 11.4
3 10.6 11.4
4 10.6 11.4
...
83359 19.2 19.2
83360 19.1 19.2
83361 19.1 19.2
83362 19.1 19.2
83363 19.0 19.2

Temperatura en suelo (60 cm)
0 11.6
1 11.6
2 11.6
3 11.6
4 11.6
...
83359 18.7
83360 18.7
83361 18.7
83362 18.7
83363 18.7

[83364 rows x 10 columns]
```

---

## Análisis estadístico descriptivo del conjunto de datos

[\[ Volver al índice \]](#)

---

```
climate_data.info()
soil_minil_data.info()
soil_mini2_data.info()

display(climate_data.describe())
display(soil_minil_data.describe())
display(soil_mini2_data.describe())
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 97001 entries, 0 to 97000
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Fecha            97001 non-null   datetime64[ns]
 1   Índice de calor  97001 non-null   float64 
 2   Humedad relativa 96993 non-null   float64 
 3   Presión de vapor 96993 non-null   float64 
 4   Radiación solar  96993 non-null   float64 
 5   Temperatura      96993 non-null   float64 
 6   Punto de rocío   96993 non-null   float64 
 7   Temperatura del bulbo húmedo 97001 non-null   float64 
 8   Velocidad del viento 96993 non-null   float64 
 9   Racha de viento máxima 96993 non-null   float64 
 10  Dirección del viento 96993 non-null   float64 
 11  Humectación en hoja (porcentaje) 96993 non-null   float64 
 12  Presión atmosférica 96993 non-null   float64 
 13  Precipitaciones   96993 non-null   float64 
dtypes: datetime64[ns](1), float64(13)
memory usage: 10.4 MB

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 89888 entries, 0 to 89887
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Fecha            89888 non-null   datetime64[ns]
 1   Contenido volumétrico (10 cm) 89880 non-null   float64 
 2   Contenido volumétrico (20 cm) 89880 non-null   float64 
 3   Contenido volumétrico (40 cm) 89880 non-null   float64 
 4   Contenido volumétrico (60 cm) 89880 non-null   float64 
 5   Agua útil        89880 non-null   float64 
 6   Temperatura en suelo (10 cm) 89880 non-null   float64 
 7   Temperatura en suelo (20 cm) 89880 non-null   float64 
 8   Temperatura en suelo (40 cm) 89880 non-null   float64 
 9   Temperatura en suelo (60 cm) 89880 non-null   float64 
dtypes: datetime64[ns](1), float64(9)
memory usage: 6.9 MB

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 83364 entries, 0 to 83363
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Fecha            83364 non-null   datetime64[ns]
 1   Contenido volumétrico (10 cm) 83356 non-null   float64 
 2   Contenido volumétrico (20 cm) 83356 non-null   float64 
 3   Contenido volumétrico (40 cm) 83356 non-null   float64 
 4   Contenido volumétrico (60 cm) 83356 non-null   float64 
 5   Agua útil        83356 non-null   float64 
 6   Temperatura en suelo (10 cm) 83356 non-null   float64 

```

```

7  Temperatura en suelo (20 cm)    83356 non-null  float64
8  Temperatura en suelo (40 cm)    83356 non-null  float64
9  Temperatura en suelo (60 cm)    83356 non-null  float64
dtypes: datetime64[ns](1), float64(9)
memory usage: 6.4 MB

```

	Fecha	Índice de calor	Humedad
relativa \			
count	97001	97001.000000	
96993.000000			
mean	2024-05-20 21:54:30.232265472	13.406654	
72.922036			
min	2023-01-01 00:00:00	-8.200000	
7.000000			
25%	2023-09-10 17:00:00	7.200000	
58.000000			
50%	2024-05-20 23:00:00	12.600000	
79.000000			
75%	2025-01-29 03:45:00	19.000000	
91.000000			
max	2025-10-08 23:45:00	40.421924	
100.000000			
std	NaN	8.602919	
21.177359			

	Presión de vapor	Radiación solar	Temperatura	Punto de rocío
\				
count	96993.000000	96993.000000	96993.000000	96993.000000
mean	18.185691	151.018929	13.456638	7.688319
min	3.680000	0.000000	-8.200000	-16.910000
25%	10.670000	0.000000	7.200000	3.970000
50%	15.080000	6.000000	12.600000	8.110000
75%	22.350000	216.000000	19.000000	11.800000
max	74.140000	1212.000000	40.400000	21.680000
std	10.735552	238.772730	8.704900	5.299786

	Temperatura del bulbo húmedo	Velocidad del viento	\
count	97001.000000	96993.000000	
mean	10.328257	4.829936	
min	-8.492907	0.000000	
25%	6.097173	1.000000	
50%	10.565531	3.000000	

75%	15.034132	7.000000
max	24.578836	36.000000
std	6.068678	4.851804
	Racha de viento máxima	Dirección del viento
count	96993.000000	96993.000000
mean	8.689256	121.355665
min	0.000000	0.000000
25%	3.000000	22.500000
50%	7.000000	135.000000
75%	12.000000	202.500000
max	57.000000	337.500000
std	7.127678	101.671012
	Humectación en hoja (porcentaje)	Presión atmosférica
Precipitaciones		
count	96993.000000	96993.000000
	96993.000000	
mean	2.236409	936.781740
0.012320		
min	0.000000	904.700000
0.000000		
25%	0.000000	933.400000
0.000000		
50%	0.000000	936.800000
0.000000		
75%	0.000000	940.500000
0.000000		
max	99.000000	957.000000
10.600000		
std	6.106364	6.666035
0.131626		
	Fecha	Contenido volumétrico (10 cm)
count	89888	89880.000000
mean	2024-05-10 05:27:10.198691584	20.378814
min	2023-01-01 00:00:00	9.000000
25%	2023-08-23 11:26:15	13.700000
50%	2024-05-04 22:07:30	20.500000
75%	2025-02-15 23:18:45	27.200000
max	2025-10-08 23:45:00	35.900000
std	Nan	6.929012
	Contenido volumétrico (20 cm)	Contenido volumétrico (40 cm)
count	89880.000000	89880.000000
mean	25.749672	30.305247
min	17.300000	21.100000
25%	23.600000	30.000000
50%	26.200000	30.500000
75%	27.800000	31.000000

max	36.200000	34.000000
std	2.971332	1.574829
	Contenido volumétrico (60 cm)	Agua útil \
count	89880.000000	89880.000000
mean	32.274536	59.782176
min	21.300000	18.260968
25%	32.100000	55.209274
50%	32.800000	59.799275
75%	33.300000	64.846317
max	34.500000	87.593377
std	2.021060	9.513411
	Temperatura en suelo (10 cm)	Temperatura en suelo (20 cm) \
count	89880.000000	89880.000000
mean	16.931110	16.704238
min	2.300000	3.200000
25%	11.300000	11.100000
50%	17.100000	16.800000
75%	22.500000	22.100000
max	34.800000	30.300000
std	6.850257	6.473244
	Temperatura en suelo (40 cm)	Temperatura en suelo (60 cm)
count	89880.000000	89880.000000
mean	16.894686	16.415595
min	5.300000	6.600000
25%	11.600000	11.600000
50%	16.900000	16.400000
75%	21.900000	21.100000
max	28.000000	26.200000
std	5.976503	5.379677
	Fecha	Contenido volumétrico (10 cm) \
count	83364	83356.000000
mean	2024-04-10 19:51:38.470563072	16.281201
min	2023-01-01 00:00:00	10.400000
25%	2023-08-06 08:41:15	13.000000
50%	2024-03-11 23:37:30	16.400000
75%	2024-10-18 04:03:45	17.800000
max	2025-10-08 23:45:00	32.700000
std	NaN	3.761337
	Contenido volumétrico (20 cm)	Contenido volumétrico (40 cm) \
count	83356.000000	83356.000000
mean	21.435584	20.780391
min	16.100000	7.300000
25%	19.700000	18.100000
50%	21.500000	21.400000
75%	22.600000	22.600000

max	31.900000	30.400000
std	2.304119	3.259983
Contenido volumétrico (60 cm)      Agua útil \		
count	83356.000000	83356.000000
mean	16.881269	28.925583
min	11.500000	2.139396
25%	14.800000	18.100364
50%	17.700000	30.613496
75%	18.400000	36.413496
max	23.100000	79.957696
std	2.583366	14.066059
Temperatura en suelo (10 cm)      Temperatura en suelo (20 cm) \		
count	83356.000000	83356.000000
mean	17.750100	18.083385
min	-7.500000	3.500000
25%	12.100000	12.600000
50%	18.300000	18.600000
75%	23.500000	23.800000
max	34.600000	32.800000
std	7.099417	6.845997
Temperatura en suelo (40 cm)      Temperatura en suelo (60 cm)		
count	83356.000000	83356.000000
mean	18.502097	18.176117
min	5.300000	-2.500000
25%	13.300000	13.400000
50%	18.900000	18.700000
75%	24.000000	23.300000
max	30.900000	42.000000
std	6.492916	6.019553

---

## Análisis y limpieza de instancias con nulos en el DataFrame

[\[ Volver al índice \]](#)

---

```
nulos_por_columna = soil_mini1_data.isnull().sum()
nulos_por_columna2 = soil_mini2_data.isnull().sum()
nulos_por_columna3 = climate_data.isnull().sum()
print(nulos_por_columna)
```

```

print(nulos_por_columna2)
print(nulos_por_columna3)

Fecha 0
Contenido volumétrico (10 cm) 8
Contenido volumétrico (20 cm) 8
Contenido volumétrico (40 cm) 8
Contenido volumétrico (60 cm) 8
Agua útil 8
Temperatura en suelo (10 cm) 8
Temperatura en suelo (20 cm) 8
Temperatura en suelo (40 cm) 8
Temperatura en suelo (60 cm) 8
dtype: int64
Fecha 0
Contenido volumétrico (10 cm) 8
Contenido volumétrico (20 cm) 8
Contenido volumétrico (40 cm) 8
Contenido volumétrico (60 cm) 8
Agua útil 8
Temperatura en suelo (10 cm) 8
Temperatura en suelo (20 cm) 8
Temperatura en suelo (40 cm) 8
Temperatura en suelo (60 cm) 8
dtype: int64
Fecha 0
Índice de calor 0
Humedad relativa 8
Presión de vapor 8
Radiación solar 8
Temperatura 8
Punto de rocío 8
Temperatura del bulbo húmedo 0
Velocidad del viento 8
Racha de viento máxima 8
Dirección del viento 8
Humectación en hoja (porcentaje) 8
Presión atmosférica 8
Precipitaciones 8
dtype: int64

```

Mostramos las instancias en las que hay nulos

```

display(soil_min1_data[soil_min1_data.isnull().any(axis=1)])
display(soil_min2_data[soil_min2_data.isnull().any(axis=1)])
display(climate_data[climate_data.isnull().any(axis=1)])

```

	Fecha	Contenido volumétrico (10 cm)	\
28854	2023-10-29 02:00:00	NaN	
28855	2023-10-29 02:15:00	NaN	

28856	2023-10-29	02:30:00		NaN
28857	2023-10-29	02:45:00		NaN
61505	2024-10-27	02:00:00		NaN
61506	2024-10-27	02:15:00		NaN
61507	2024-10-27	02:30:00		NaN
61508	2024-10-27	02:45:00		NaN
Contenido volumétrico (20 cm)		Contenido volumétrico (40 cm)		\
28854		NaN		NaN
28855		NaN		NaN
28856		NaN		NaN
28857		NaN		NaN
61505		NaN		NaN
61506		NaN		NaN
61507		NaN		NaN
61508		NaN		NaN
Contenido volumétrico (60 cm)		Agua útil	Temperatura en suelo	
(10 cm) \				
28854		NaN	NaN	
Nan				
28855		NaN	NaN	
Nan				
28856		NaN	NaN	
Nan				
28857		NaN	NaN	
Nan				
61505		NaN	NaN	
Nan				
61506		NaN	NaN	
Nan				
61507		NaN	NaN	
Nan				
61508		NaN	NaN	
Nan				
Temperatura en suelo (20 cm)		Temperatura en suelo (40 cm)		\
28854		NaN		NaN
28855		NaN		NaN
28856		NaN		NaN
28857		NaN		NaN
61505		NaN		NaN
61506		NaN		NaN
61507		NaN		NaN
61508		NaN		NaN
Temperatura en suelo (60 cm)				
28854		NaN		
28855		NaN		
28856		NaN		

28857		NaN	
61505		NaN	
61506		NaN	
61507		NaN	
61508		NaN	
	Fecha	Contenido volumétrico (10 cm)	\
28869	2023-10-29 02:00:00	NaN	
28870	2023-10-29 02:15:00	NaN	
28871	2023-10-29 02:30:00	NaN	
28872	2023-10-29 02:45:00	NaN	
63277	2024-10-27 02:00:00	NaN	
63278	2024-10-27 02:15:00	NaN	
63279	2024-10-27 02:30:00	NaN	
63280	2024-10-27 02:45:00	NaN	
	Contenido volumétrico (20 cm)	Contenido volumétrico (40 cm)	\
28869		NaN	NaN
28870		NaN	NaN
28871		NaN	NaN
28872		NaN	NaN
63277		NaN	NaN
63278		NaN	NaN
63279		NaN	NaN
63280		NaN	NaN
	Contenido volumétrico (60 cm)	Agua útil	Temperatura en suelo
(10 cm)	\		
28869		NaN	NaN
NaN			
28870		NaN	NaN
NaN			
28871		NaN	NaN
NaN			
28872		NaN	NaN
NaN			
63277		NaN	NaN
NaN			
63278		NaN	NaN
NaN			
63279		NaN	NaN
NaN			
63280		NaN	NaN
NaN			
	Temperatura en suelo (20 cm)	Temperatura en suelo (40 cm)	\
28869		NaN	NaN
28870		NaN	NaN
28871		NaN	NaN
28872		NaN	NaN

63277		NaN		NaN
63278		NaN		NaN
63279		NaN		NaN
63280		NaN		NaN
	Temperatura en suelo (60 cm)			
28869		NaN		
28870		NaN		
28871		NaN		
28872		NaN		
63277		NaN		
63278		NaN		
63279		NaN		
63280		NaN		
	Fecha	Índice de calor	Humedad relativa	\
28893	2023-10-29 02:00:00	13.5	NaN	
28894	2023-10-29 02:15:00	13.6	NaN	
28895	2023-10-29 02:30:00	13.5	NaN	
28896	2023-10-29 02:45:00	13.4	NaN	
63736	2024-10-27 02:00:00	6.3	NaN	
63737	2024-10-27 02:15:00	6.3	NaN	
63738	2024-10-27 02:30:00	6.2	NaN	
63739	2024-10-27 02:45:00	6.3	NaN	
	Presión de vapor	Radiación solar	Temperatura	Punto de rocío
\				
28893	NaN	NaN	NaN	NaN
28894	NaN	NaN	NaN	NaN
28895	NaN	NaN	NaN	NaN
28896	NaN	NaN	NaN	NaN
63736	NaN	NaN	NaN	NaN
63737	NaN	NaN	NaN	NaN
63738	NaN	NaN	NaN	NaN
63739	NaN	NaN	NaN	NaN
	Temperatura del bulbo húmedo	Velocidad del viento	\	
28893	12.140945	NaN		
28894	12.143126	NaN		
28895	11.954894	NaN		
28896	11.673305	NaN		
63736	5.890469	NaN		
63737	5.890493	NaN		

63738	5.792191	NaN
63739	5.959094	NaN
Racha de viento máxima Dirección del viento \		
28893	NaN	NaN
28894	NaN	NaN
28895	NaN	NaN
28896	NaN	NaN
63736	NaN	NaN
63737	NaN	NaN
63738	NaN	NaN
63739	NaN	NaN
Humectación en hoja (porcentaje) Presión atmosférica		
Precipitaciones		
28893	NaN	NaN
NaN		
28894	NaN	NaN
NaN		
28895	NaN	NaN
NaN		
28896	NaN	NaN
NaN		
63736	NaN	NaN
NaN		
63737	NaN	NaN
NaN		
63738	NaN	NaN
NaN		
63739	NaN	NaN
NaN		

Eliminamos dichas instancias y comprobamos que no quedan instancias con valores nulos en nuestro DataFrame

```
soil_minil_data.dropna(inplace=True)
soil_mini2_data.dropna(inplace=True)
climate_data.dropna(inplace=True)

nulos_por_columna = soil_minil_data.isnull().sum()
nulos_por_columna2 = soil_mini2_data.isnull().sum()
nulos_por_columna3 = climate_data.isnull().sum()

print(nulos_por_columna)
print(nulos_por_columna2)
print(nulos_por_columna3)

Fecha 0
Contenido volumétrico (10 cm) 0
```

```

Contenido volumétrico (20 cm)    0
Contenido volumétrico (40 cm)    0
Contenido volumétrico (60 cm)    0
Agua útil                         0
Temperatura en suelo (10 cm)    0
Temperatura en suelo (20 cm)    0
Temperatura en suelo (40 cm)    0
Temperatura en suelo (60 cm)    0
dtype: int64
Fecha                             0
Contenido volumétrico (10 cm)    0
Contenido volumétrico (20 cm)    0
Contenido volumétrico (40 cm)    0
Contenido volumétrico (60 cm)    0
Agua útil                         0
Temperatura en suelo (10 cm)    0
Temperatura en suelo (20 cm)    0
Temperatura en suelo (40 cm)    0
Temperatura en suelo (60 cm)    0
dtype: int64
Fecha                             0
Índice de calor                   0
Humedad relativa                 0
Presión de vapor                  0
Radiación solar                   0
Temperatura                       0
Punto de rocío                   0
Temperatura del bulbo húmedo     0
Velocidad del viento              0
Racha de viento máxima            0
Dirección del viento              0
Humectación en hoja (porcentaje) 0
Presión atmosférica                0
Precipitaciones                   0
dtype: int64

import seaborn as sns

# Seleccionar solo columnas numéricas (excluyendo datetime)
numeric_cols =
soil_minil_data.select_dtypes(include=['number']).columns

# Calcular medias y varianzas para ambos datasets
medias1 = soil_minil_data[numeric_cols].mean(axis=0)
varianzas1 = soil_minil_data[numeric_cols].var(axis=0)

medias2 = soil_mini2_data[numeric_cols].mean(axis=0)
varianzas2 = soil_mini2_data[numeric_cols].var(axis=0)

#lo mostramos en forma de grafico para su mejor visualización

```

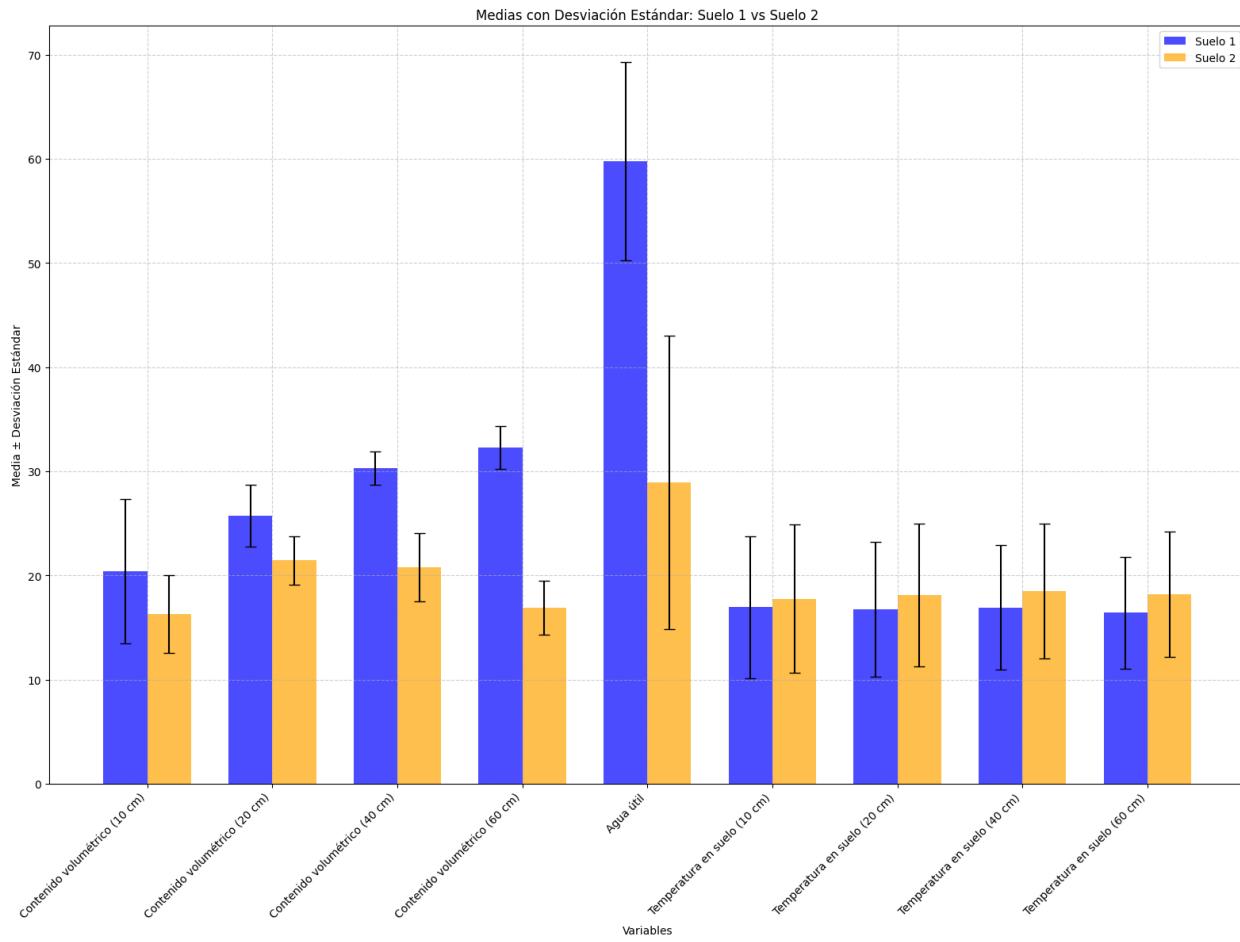
```
plt.figure(figsize=(16, 12))

# Posiciones de las barras
x = np.arange(len(numeric_cols))
width = 0.35

# Calcular desviaciones estándar
std1 = np.sqrt(varianzas1)
std2 = np.sqrt(varianzas2)

plt.bar(x - width/2, medias1, width, yerr=std1, label='Suelo 1',
color='blue', alpha=0.7, capsize=5)
plt.bar(x + width/2, medias2, width, yerr=std2, label='Suelo 2',
color='orange', alpha=0.7, capsize=5)

plt.xlabel('Variables')
plt.ylabel('Media ± Desviación Estándar')
plt.title('Medias con Desviación Estándar: Suelo 1 vs Suelo 2')
plt.xticks(x, numeric_cols, rotation=45, ha='right')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



## Agregación diaria de datos de suelo y clima

[ [Volver al índice](#) ]

En esta celda se combinan los datos de los sensores de suelo **mini1** y **mini2** calculando la **media por fecha**, obteniendo un único DataFrame de suelo representativo.

Posteriormente, este DataFrame de suelo se **fusiona con el DataFrame climático** usando las fechas comunes.

A continuación, los datos —originalmente con una granularidad de **15 minutos**— se agrupan a **escala diaria**:

- Las **variables acumulativas** (radiación solar y precipitaciones) se **suman por día**.
- El resto de variables se **promedian por día**.

Además, se crea la columna **humedad relativa mínima diaria**, derivada de la humedad relativa, necesaria para el **cálculo posterior de la evapotranspiración del cultivo**.

```

for df in [climate_data, soil_minil_data, soil_mini2_data]:
    if "Fecha" in df.columns:
        df.set_index("Fecha", inplace=True)
        df.index = pd.to_datetime(df.index)

# --- COMBINAR SUELO ---
soil_combined = pd.concat([soil_minil_data,
soil_mini2_data]).groupby("Fecha").mean()

# --- UNIFICAR SUELO + CLIMA ---
combined_df = pd.concat([soil_combined, climate_data], axis=1,
join="inner")

# --- DEFINIR QUÉ COLUMNAS SON ACUMULATIVAS ---
cols_acumulativas = [
    "Radiación solar",
    "Precipitaciones"
]

# --- AGRUPAR SOLO DÍAS EXISTENTES ---
daily_sum =
combined_df[cols_acumulativas].groupby(pd.Grouper(freq="D")).sum()
daily_mean =
combined_df.drop(columns=cols_acumulativas).groupby(pd.Grouper(freq="D")).mean()

# --- AÑADIR HUMEDAD RELATIVA MÍNIMA ---
# Asumiendo que la columna se llama "Humedad relativa"
daily_min_humedad = combined_df[["Humedad
relativa"]].groupby(pd.Grouper(freq="D")).min()
daily_min_humedad = daily_min_humedad.rename(columns={"Humedad
relativa": "Humedad relativa mínima"})

# --- COMBINAR RESULTADOS ---
daily_df = pd.concat([daily_mean, daily_sum, daily_min_humedad],
axis=1).sort_index()

daily_df = daily_df.interpolate()

# --- RESULTADO FINAL ---
# pd.set_option('display.max_columns', None, 'display.max_rows', None)
display(daily_df)

daily_df.shape

```

cm) \ Fecha	Contenido volumétrico (10 cm)	Contenido volumétrico (20
2023-01-01 27.232812	17.048438	
2023-01-02 27.236458	17.048958	
2023-01-03 27.044149	16.934574	
2023-01-04 26.938021	16.839063	
2023-01-05 26.856771	16.848437	
...	...	
.. 2025-10-04 26.424479	27.030729	
2025-10-05 25.438542	25.577083	
2025-10-06 26.796875	27.211979	
2025-10-07 28.032813	29.270833	
2025-10-08 26.519271	27.375521	
cm) \ Fecha	Contenido volumétrico (40 cm)	Contenido volumétrico (60
2023-01-01 27.794271	26.251563	
2023-01-02 27.770833	26.228646	
2023-01-03 27.701596	26.117553	
2023-01-04 27.729688	26.095312	
2023-01-05 27.647917	25.992708	
...	...	
.. 2025-10-04 25.901042	28.725521	
2025-10-05 25.587500	28.291667	
2025-10-06 25.647396	28.729167	
2025-10-07	29.786979	

26.306250  
2025-10-08  
25.921875

28.738542

Fecha	Agua útil	Temperatura en suelo (10 cm)	\
2023-01-01	56.814366	11.004167	
2023-01-02	56.926758	10.421875	
2023-01-03	56.284717	7.348936	
2023-01-04	55.993025	6.418229	
2023-01-05	55.659726	6.455729	
...	...	...	
2025-10-04	61.980537	18.506771	
2025-10-05	57.713626	17.705729	
2025-10-06	62.337520	16.959375	
2025-10-07	69.298230	17.343750	
2025-10-08	62.613956	18.143750	
\	Temperatura en suelo (20 cm)	Temperatura en suelo (40 cm)	
Fecha			
2023-01-01	10.955208	11.408333	
2023-01-02	10.869271	11.564062	
2023-01-03	8.566489	10.463830	
2023-01-04	7.303646	9.194271	
2023-01-05	7.159896	8.735417	
...	...	...	
2025-10-04	18.428125	18.663542	
2025-10-05	18.268750	18.910937	
2025-10-06	17.371354	18.278646	
2025-10-07	17.500521	18.148958	
2025-10-08	18.136458	18.436458	
\	Temperatura en suelo (60 cm)	Índice de calor	...
Fecha			\
2023-01-01	11.480208	12.891667	...
2023-01-02	11.553646	6.855208	...
2023-01-03	11.201064	0.775532	...
2023-01-04	10.352604	0.633333	...

2023-01-05	9.795312	0.539583	...
...	...	...	...
2025-10-04	18.423438	18.060538	...
2025-10-05	18.635417	13.093750	...
2025-10-06	18.408333	13.386458	...
2025-10-07	18.187500	14.866887	...
2025-10-08	18.234896	16.794792	...
Punto de rocío Temperatura del bulbo húmedo \			
Fecha			
2023-01-01	4.675417	9.020438	
2023-01-02	4.751562	5.908868	
2023-01-03	0.316596	0.594564	
2023-01-04	0.521042	0.594022	
2023-01-05	0.527917	0.535606	
...	...	...	
2025-10-04	10.628101	13.796026	
2025-10-05	7.322281	10.147863	
2025-10-06	6.435258	9.772663	
2025-10-07	7.364555	10.796727	
2025-10-08	9.803297	12.894810	
Velocidad del viento Racha de viento máxima \			
Fecha			
2023-01-01	13.854167	20.531250	
2023-01-02	4.093750	7.322917	
2023-01-03	0.691489	1.797872	
2023-01-04	1.729167	3.333333	
2023-01-05	1.333333	2.760417	
...	...	...	
2025-10-04	1.927083	4.687500	
2025-10-05	3.895833	7.802083	
2025-10-06	1.625000	4.062500	
2025-10-07	0.520833	1.718750	
2025-10-08	2.145833	4.739583	
Dirección del viento Humectación en hoja (porcentaje) \			
Fecha			
2023-01-01	176.484375	2.177083	
2023-01-02	186.093750	0.500000	
2023-01-03	165.638298	0.595745	
2023-01-04	174.609375	2.052083	
2023-01-05	91.875000	5.562500	
...	...	...	
2025-10-04	181.406250	0.000000	
2025-10-05	78.750000	0.104167	
2025-10-06	110.390625	0.114583	
2025-10-07	81.328125	0.000000	
2025-10-08	72.890625	0.958333	

```

Presión atmosférica Radiación solar Precipitaciones \
Fecha
2023-01-01      937.696875      5826.0      3.0
2023-01-02      942.617708      4620.0      1.0
2023-01-03      949.270213      3837.0      0.0
2023-01-04      952.208333      3282.0      0.2
2023-01-05      948.176042      3054.0      0.0
...
2025-10-04      940.113542      14664.0      0.0
2025-10-05      942.435417      15558.0      0.0
2025-10-06      942.463542      15573.0      0.0
2025-10-07      941.246875      14898.0      0.0
2025-10-08      939.396875      13752.0      1.2

Humedad relativa mínima
Fecha
2023-01-01      42.0
2023-01-02      68.0
2023-01-03      83.0
2023-01-04      97.0
2023-01-05      99.0
...
2025-10-04      44.0
2025-10-05      47.0
2025-10-06      28.0
2025-10-07      37.0
2025-10-08      39.0

[1012 rows x 23 columns]
(1012, 23)

```

Comprobamos que la solución de interpolar para crear datos sintéticos en los días vacíos funciona correctamente

```

# Ver si hay missing dates
print(f"Rango de fechas: {daily_df.index.min()} to {daily_df.index.max()}")
print(f"Número de registros: {len(daily_df)}")

# Calcular la frecuencia más común
freq_diff = pd.Series(np.diff(daily_df.index)).value_counts()
print("\nFrecuencias encontradas:")
print(freq_diff)

```

Rango de fechas: 2023-01-01 00:00:00 to 2025-10-08 00:00:00  
 Número de registros: 1012

Frecuencias encontradas:

```
1 days    1011
Name: count, dtype: int64
```

---

# Análisis exploratorio de datos

[ [Volver al índice](#) ]

---

---

## Visualizaciones exploratorias

- Visualizaciones exploratorias:
  - Diagramas de caja para detectar la distribución y posibles valores atípicos en variables numéricas.
  - Graficos con la evolución de las variables a lo largo del tiempo para visualizar posibles tendencias.

```
numeric_cols = daily_df.select_dtypes(include=['number']).columns
numeric_cols = numeric_cols.drop(["Radiación solar", "Presión
atmosférica"])

# Diagramas de caja para detectar distribucion y valores atípicos en
variables numéricas

plt.figure(figsize=(18, 18))
boxplot = sns.boxplot(data=daily_df[numeric_cols])
plt.xticks(rotation=30, ha='right', fontsize=10) # Reduce tamaño de
fuente
boxplot.figure.subplots_adjust(bottom=0.3) # Aumenta espacio inferior
plt.show()

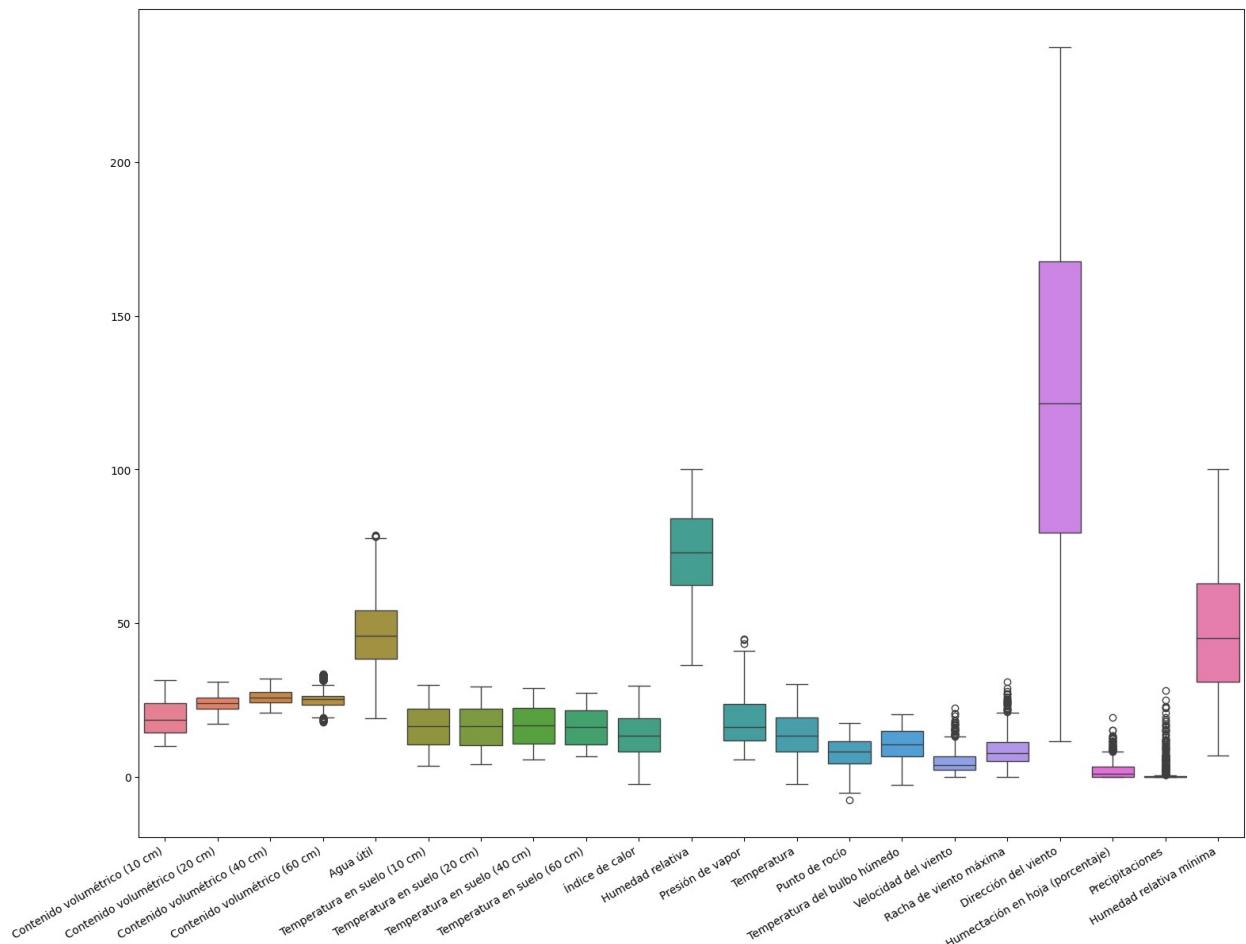
# Variables a monitorizar en el tiempo
vars_clave = ['Presión atmosférica', 'Humedad relativa', 'Humedad
```

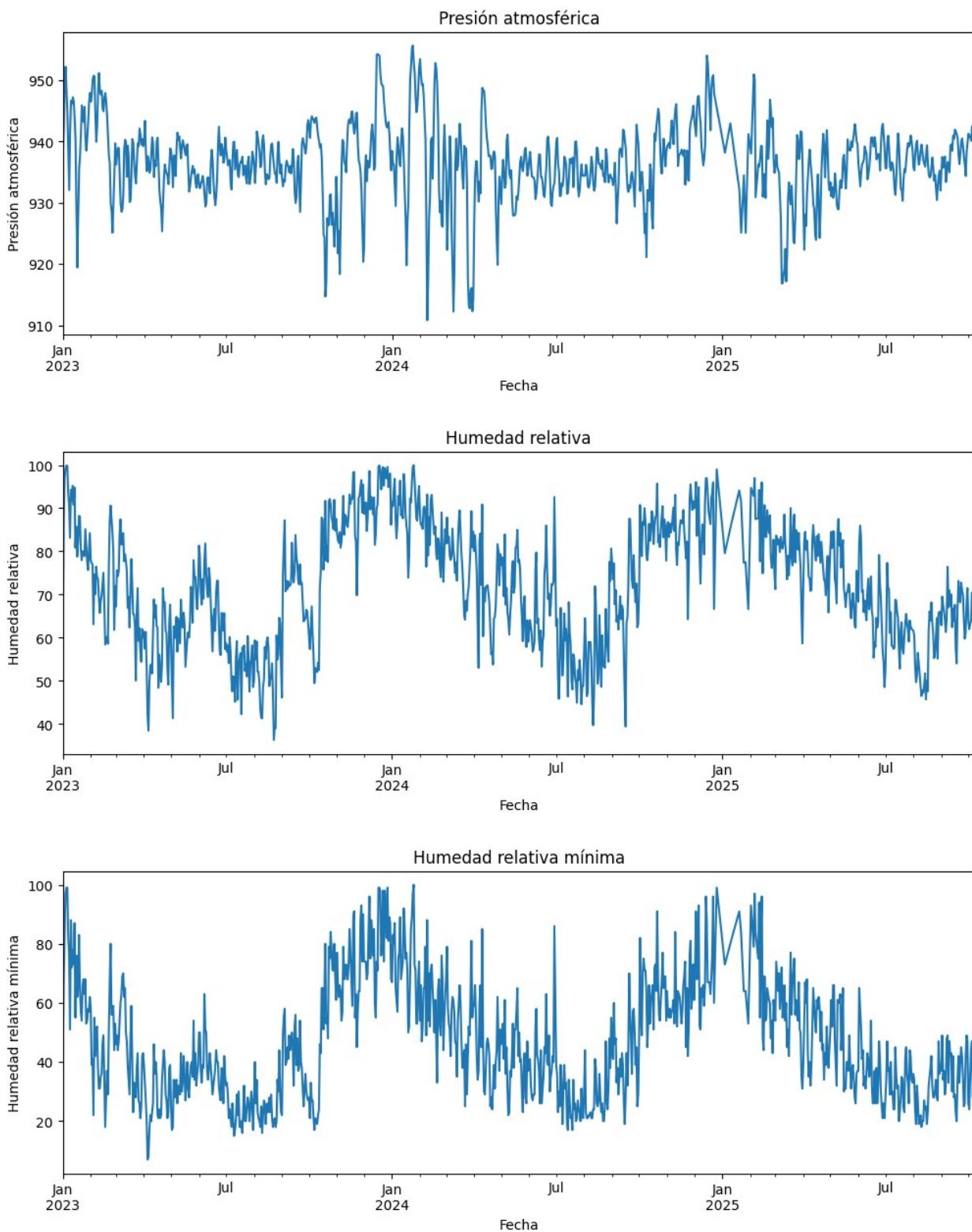
```

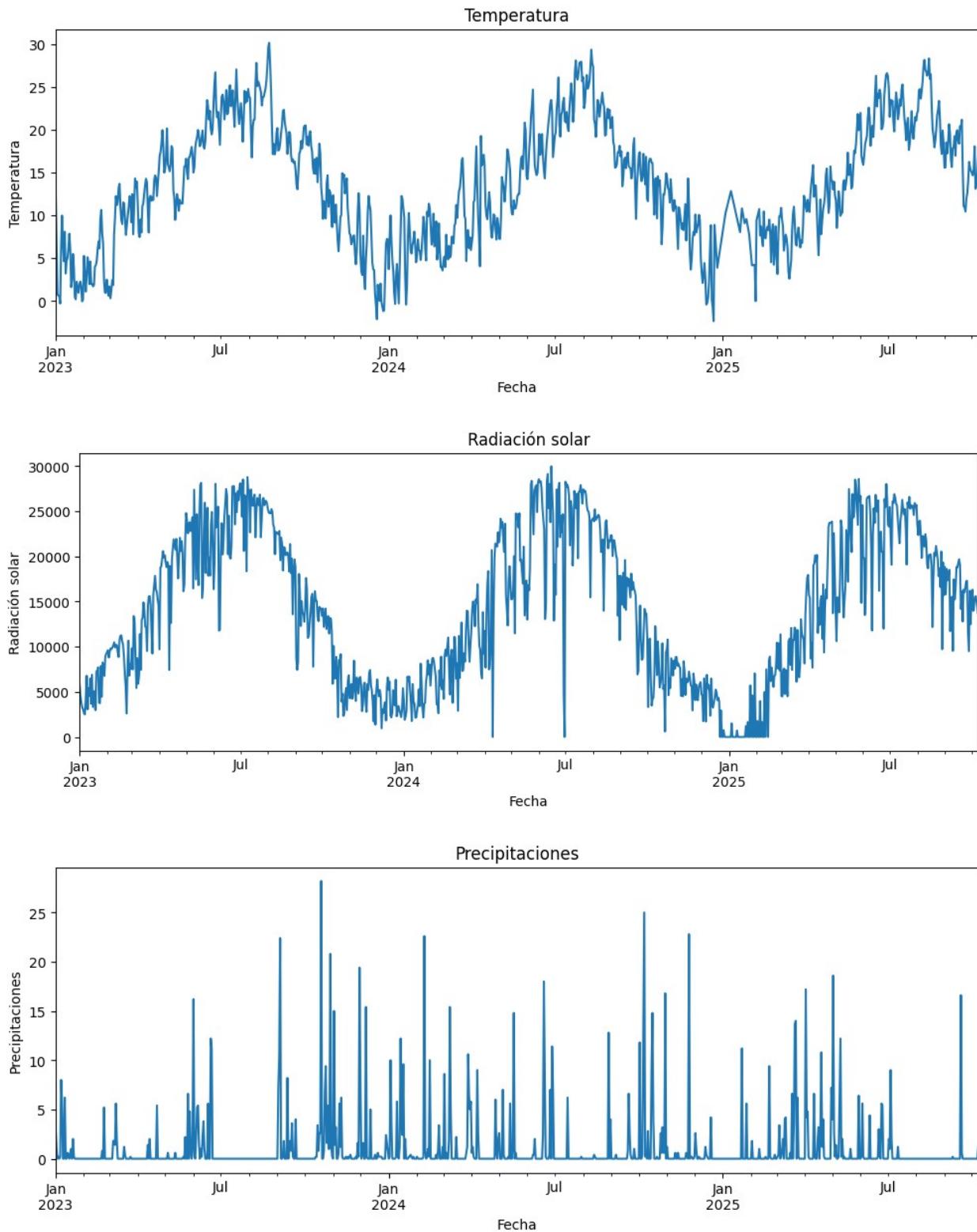
relativa mínima',
        'Temperatura', 'Radiación solar', 'Precipitaciones',
'Velocidad del viento']

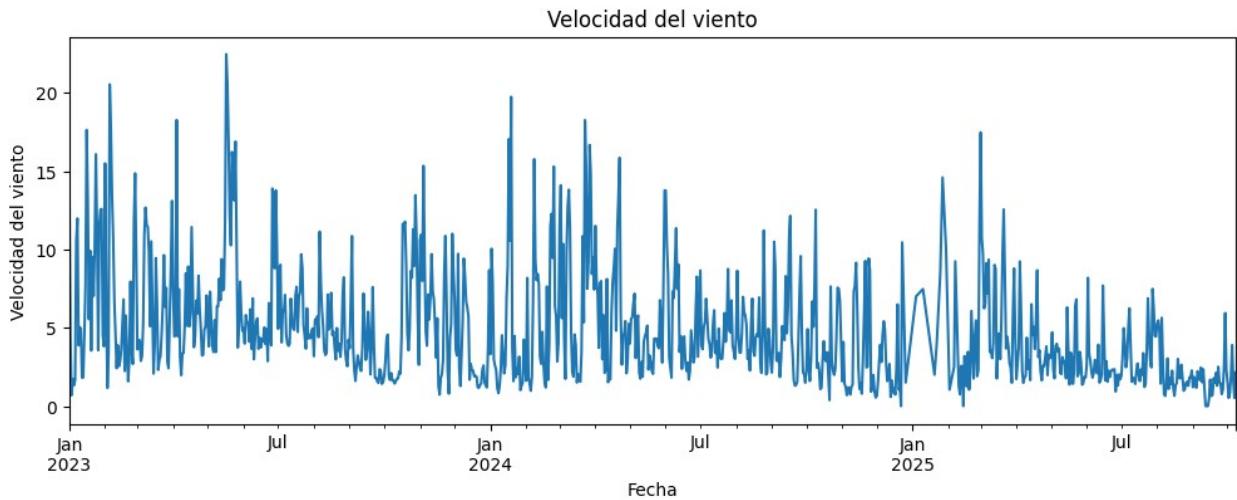
for var in vars_clave:
    plt.figure(figsize=(12,4))
    daily_df[var].plot(title=var)
    plt.xlabel("Fecha")
    plt.ylabel(var)
    plt.show()

```









## Selección de features

Mantendremos únicamente en el dataframe, aquellas variables que necesitemos para el calculo de la formula FAO-56.

- Temperatura
- Humedad relativa
- Humedad relativa mínima
- Velocidad del viento
- Presión atmosférica
- Radiación solar
- Precipitaciones

```
columnas_a_eliminar = [
    'Agua útil',
    'Contenido volumétrico (10 cm)',
    'Contenido volumétrico (20 cm)',
    'Contenido volumétrico (40 cm)',
    'Contenido volumétrico (60 cm)',
    'Humectación en hoja (porcentaje)',
    'Presión de vapor',
    'Punto de rocío',
    'Racha de viento máxima',
    'Temperatura del bulbo húmedo',
    'Temperatura en suelo (10 cm)',
    'Temperatura en suelo (20 cm)',
```

```

        'Temperatura en suelo (40 cm)',
        'Temperatura en suelo (60 cm)',
        'Índice de calor',
        'Dirección del viento'
    ]
daily_df.drop(columnas_a_eliminar, axis=1, inplace=True)

display(daily_df)

```

Fecha	Humedad relativa	Temperatura	Velocidad del viento	\
2023-01-01	58.583333	12.891667	13.854167	
2023-01-02	87.062500	6.855208	4.093750	
2023-01-03	96.882979	0.775532	0.691489	
2023-01-04	99.197917	0.633333	1.729167	
2023-01-05	99.916667	0.539583	1.333333	
...	...	...	...	...
2025-10-04	63.895833	18.047917	1.927083	
2025-10-05	70.437500	13.093750	3.895833	
2025-10-06	67.520833	13.386458	1.625000	
2025-10-07	64.229167	14.877083	0.520833	
2025-10-08	66.125000	16.794792	2.145833	
Fecha	Presión atmosférica	Radiación solar	Precipitaciones	\
2023-01-01	937.696875	5826.0	3.0	
2023-01-02	942.617708	4620.0	1.0	
2023-01-03	949.270213	3837.0	0.0	
2023-01-04	952.208333	3282.0	0.2	
2023-01-05	948.176042	3054.0	0.0	
...	...	...	...	...
2025-10-04	940.113542	14664.0	0.0	
2025-10-05	942.435417	15558.0	0.0	
2025-10-06	942.463542	15573.0	0.0	
2025-10-07	941.246875	14898.0	0.0	
2025-10-08	939.396875	13752.0	1.2	
Fecha	Humedad relativa mínima			
2023-01-01	42.0			
2023-01-02	68.0			
2023-01-03	83.0			
2023-01-04	97.0			
2023-01-05	99.0			
...	...			
2025-10-04	44.0			
2025-10-05	47.0			
2025-10-06	28.0			
2025-10-07	37.0			

2025-10-08

39.0

[1012 rows x 7 columns]

---

## Generación datos sintéticos

[[Volver al índice](#)]

---

---

### SIMULACIÓN BASADA EN FÍSICA AGRÍCOLA (MODIFICADA PARA PASADO)

[[Volver al índice](#)]

---

```
from datetime import datetime, timedelta

def calcular_parametros_estacionales(df_real):
    """Calcula parámetros estacionales a partir de datos reales"""
    df = df_real.copy()
    df['dia_año'] = df.index.dayofyear

    parametros = {}

    for columna in df.columns:
        if columna == 'dia_año':
            continue

            # Calcular media y amplitud estacional por día del año
            patron_diario = df.groupby('dia_año')[columna].agg(['mean',
            'std']).ffill()

            # Suavizar con media móvil
            patron_diario['mean_smooth'] =
```

```

patron_diario['mean'].rolling(7, center=True, min_periods=1).mean()
    patron_diario['std_smooth'] = patron_diario['std'].rolling(7,
center=True, min_periods=1).mean()

    parametros[columna] = {
        'media': patron_diario['mean_smooth'].mean(),
        'amplitud': (patron_diario['mean_smooth'].max() -
patron_diario['mean_smooth'].min()) / 2,
        'std_promedio': patron_diario['std_smooth'].mean(),
        'patron_diario': patron_diario['mean_smooth']
    }

    return parametros

def simular_datos_fisicos_completo(fecha, parametros_estacionales):
    """
    Simula datos basados en relaciones físicas conocidas para todas
    las variables
    """
    if isinstance(fecha, (pd.Timestamp, datetime)):
        dia_del_ano = fecha.timetuple().tm_yday
    else:
        dia_del_ano = fecha.dayofyear

    # Patrón estacional base
    estacionalidad_base = np.sin(2 * np.pi * (dia_del_ano - 80) / 365)
    estacionalidad_secundaria = np.sin(4 * np.pi * (dia_del_ano - 80) /
/ 365)

    # TEMPERATURA (patrón principal)
    temp_media = parametros_estacionales['Temperatura'][
    'patron_diario'].get(dia_del_ano,
    parametros_estacionales['Temperatura']['media'])
    temp_sintetica = temp_media + np.random.normal(0,
    parametros_estacionales['Temperatura']['std_promedio'] * 0.5)

    # HUMEDAD RELATIVA (inversamente relacionada con temperatura)
    hr_media = parametros_estacionales['Humedad relativa'][
    'patron_diario'].get(dia_del_ano, parametros_estacionales['Humedad
    relativa']['media'])
    hr_sintetica = hr_media - 0.5 * (temp_sintetica - temp_media) +
    np.random.normal(0, parametros_estacionales['Humedad relativa'][
    'std_promedio'] * 0.5)

    # RADIACIÓN SOLAR (correlacionada con temperatura, opuesta a
    humedad)
    rad_media = parametros_estacionales['Radiación solar'][
    'patron_diario'].get(dia_del_ano, parametros_estacionales['Radiación
    solar']['media'])

```

```

    rad_sintetica = rad_media + 20 * (temp_sintetica - temp_media) +
    np.random.normal(0, parametros_estacionales['Radiación solar']
    ['std_promedio'] * 0.5)

    # VELOCIDAD DEL VIENTO (patrón independiente)
    viento_media = parametros_estacionales['Velocidad del viento']
    ['patron_diario'].get(dia_del_ano, parametros_estacionales['Velocidad
    del viento']['media'])
    viento_sintetico = viento_media + np.random.normal(0,
    parametros_estacionales['Velocidad del viento']['std_promedio'] * 0.5)

    # PRESIÓN ATMOSFÉRICA (inversamente relacionada con temperatura)
    presion_media = parametros_estacionales['Presión atmosférica']
    ['patron_diario'].get(dia_del_ano, parametros_estacionales['Presión
    atmosférica']['media'])
    presion_sintetica = presion_media - 0.1 * (temp_sintetica -
    temp_media) + np.random.normal(0, parametros_estacionales['Presión
    atmosférica']['std_promedio'] * 0.3)

    # PRECIPITACIONES (eventos raros con patrón estacional)
    precip_media = parametros_estacionales['Precipitaciones']
    ['patron_diario'].get(dia_del_ano,
    parametros_estacionales['Precipitaciones']['media'])
    prob_lluvia = min(0.3, max(0.01, hr_sintetica / 500))
    if np.random.random() < prob_lluvia:
        precipitacion_sintetica = np.random.exponential(precip_media +
    0.1) if precip_media > 0 else np.random.exponential(0.5)
    else:
        precipitacion_sintetica = 0.0

    # HUMEDAD RELATIVA MÍNIMA (relacionada con humedad relativa)
    hr_min_media = parametros_estacionales['Humedad relativa mínima']
    ['patron_diario'].get(dia_del_ano, parametros_estacionales['Humedad
    relativa mínima']['media'])
    hr_min_sintetica = hr_min_media + 0.7 * (hr_sintetica - hr_media)
    + np.random.normal(0, parametros_estacionales['Humedad relativa
    mínima']['std_promedio'] * 0.5)

    # Aplicar restricciones físicas
    return {
        'Humedad relativa': np.clip(hr_sintetica, 10, 100),
        'Temperatura': np.clip(temp_sintetica, -15, 45),
        'Velocidad del viento': max(viento_sintetico, 0),
        'Presión atmosférica': np.clip(presion_sintetica, 900, 1050),
        'Radiación solar': max(rad_sintetica, 0),
        'Precipitaciones': max(precipitacion_sintetica, 0),
        'Humedad relativa mínima': np.clip(hr_min_sintetica, 5, 100)
    }

```

```

def generar_datos_fisicos_pasado(df_real, años_generar=5):
    """Genera datos sintéticos en el PASADO usando simulación
    física"""

    # Calcular fecha de inicio (años_generar antes del primer dato
    # real)
    fecha_inicio_real = df_real.index.min()
    fecha_inicio_sintetico = fecha_inicio_real - timedelta(days=365 * 
años_generar)

    parametros = calcular_parametros_estacionales(df_real)

    datos_sinteticos = []
    fechas = []

    fecha_actual = fecha_inicio_sintetico
    for _ in range(años_generar * 365):
        datos_dia = simular_datos_fisicos_completo(fecha_actual,
parametros)
        datos_sinteticos.append(datos_dia)
        fechas.append(fecha_actual)
        fecha_actual += timedelta(days=1)

    df_sintetico = pd.DataFrame(datos_sinteticos, index=fechas)
    df_sintetico.index.name = 'Fecha'

    return df_sintetico

```

---

## BOOTSTRAPPING ESTACIONAL CON PERTURBACIÓN (MODIFICADO PARA PASADO)

[ [Volver al índice](#) ]

---

```

def aplicar_restricciones_fisicas(datos_mes_sintetico):
    """Aplica restricciones físicas a los datos sintéticos"""
    # Humedad relativa entre 10% y 100%
    datos_mes_sintetico[:, 0] = np.clip(datos_mes_sintetico[:, 0], 10,
100)
    # Temperatura entre -15°C y 45°C
    datos_mes_sintetico[:, 1] = np.clip(datos_mes_sintetico[:, 1], -15,
45)

```

```

# Velocidad del viento no negativa
datos_mes_sintetico[:, 2] = np.maximum(datos_mes_sintetico[:, 2],
0)
# Presión atmosférica entre 900 y 1050 hPa
datos_mes_sintetico[:, 3] = np.clip(datos_mes_sintetico[:, 3],
900, 1050)
# Radiación solar no negativa
datos_mes_sintetico[:, 4] = np.maximum(datos_mes_sintetico[:, 4],
0)
# Precipitaciones no negativas
datos_mes_sintetico[:, 5] = np.maximum(datos_mes_sintetico[:, 5],
0)
# Humedad relativa mínima entre 5% y 100%
datos_mes_sintetico[:, 6] = np.clip(datos_mes_sintetico[:, 6], 5,
100)

return datos_mes_sintetico

def bootstrap_estacional_perturbado_pasado(df_existente,
años_extra=5):
    """Crea datos sintéticos en el PASADO perturbando años
existentes"""

    datos_sinteticos = []
    columnas = df_existente.columns

    # Determinar el primer año disponible y calcular años anteriores
    primer_año_real = df_existente.index.min().year
    primer_año_sintetico = primer_año_real - años_extra

    for año_sintetico in range(primer_año_sintetico, primer_año_real):
        for mes in range(1, 13):
            # Tomar datos del mes correspondiente de años existentes
            datos_mes_original = df_existente[df_existente.index.month
== mes]

            if len(datos_mes_original) > 0:
                # Seleccionar aleatoriamente días del mes de años
                existentes
                n_dias = len(datos_mes_original)
                # Usar el número correcto de días para cada mes
                if mes in [1, 3, 5, 7, 8, 10, 12]:
                    dias_en_mes = 31
                elif mes in [4, 6, 9, 11]:
                    dias_en_mes = 30
                else: # Febrero
                    # Simplificación: usar 28 días para febrero
                    dias_en_mes = 28

                indices_aleatorios = np.random.choice(n_dias,

```

```

size=dias_en_mes, replace=True)
            datos_mes_seleccionados =
datos_mes_original.iloc[indices_aleatorios].values

            # Perturbación controlada manteniendo correlaciones
            perturbacion = np.random.normal(1, 0.15,
datos_mes_seleccionados.shape)
            datos_mes_sintetico = datos_mes_seleccionados *
perturbacion

            # Aplicar restricciones de dominio
            datos_mes_sintetico =
aplicar_restricciones_fisicas(datos_mes_sintetico)

            # Crear fechas sintéticas para el mes en el pasado
            for dia in range(len(datos_mes_sintetico)):
                try:
                    fecha_sintetica =
pd.Timestamp(year=año_sintetico, month=mes, day=dia+1)
                    datos_sinteticos.append([fecha_sintetica] +
list(datos_mes_sintetico[dia]))
                except ValueError:
                    # Para días inválidos (como 30 de febrero)
                    continue

            df_sintetico = pd.DataFrame(datos_sinteticos, columns=['Fecha'] +
list(columnas))
            df_sintetico.set_index('Fecha', inplace=True)
            df_sintetico = df_sintetico.sort_index()
            return df_sintetico

```

---

## Creación gráficos comparadores datos sintéticos vs datos reales

[ [Volver al índice](#) ]

---

```

def comparar_datos_reales_vs_sinteticos(df_real, df_sintetico,
atributos=None,                                     años_comparacion=4,
fecha_inicio_comparacion=None):
    """
    Compara visualmente datos reales vs sintéticos para cada atributo
    """

```

```

if atributos is None:
    atributos = [col for col in df_real.columns if col not in
['fecha', 'Fecha', 'date', 'Date', 'origen']]

# Asegurar que ambos dataframes tienen índice datetime
if not isinstance(df_real.index, pd.DatetimeIndex):
    df_real = df_real.copy()
    df_real.index = pd.to_datetime(df_real.index)

if not isinstance(df_sintetico.index, pd.DatetimeIndex):
    df_sintetico = df_sintetico.copy()
    df_sintetico.index = pd.to_datetime(df_sintetico.index)

# Determinar fecha de inicio para comparación
if fecha_inicio_comparacion is None:
    # Usar la fecha más temprana entre reales y sintéticos
    fecha_inicio = min(df_real.index.min(),
df_sintetico.index.min())
else:
    fecha_inicio = pd.to_datetime(fecha_inicio_comparacion)

# Calcular fecha fin basada en años_comparacion
fecha_fin = fecha_inicio + timedelta(days=365 * años_comparacion +
len(df_real))

# Asegurar que la fecha fin no excede los datos disponibles
fecha_fin = min(fecha_fin, max(df_real.index.max(),
df_sintetico.index.max()))

# Filtrar datos para el período de comparación
mask_real = (df_real.index >= fecha_inicio) & (df_real.index <=
fecha_fin)
mask_sintetico = (df_sintetico.index >= fecha_inicio) &
(df_sintetico.index <= fecha_fin)

df_real_comp = df_real[mask_real].copy()
df_sintetico_comp = df_sintetico[mask_sintetico].copy()

# Añadir columnas auxiliares
for df in [df_real_comp, df_sintetico_comp]:
    df['tipo'] = 'Real' if df is df_real_comp else 'Sintético'
    df['año'] = df.index.year
    df['dia_año'] = df.index.dayofyear

# Crear gráficos para cada atributo
figuras = []
for i, atributo in enumerate(atributos):
    fig = crear_grafico_comparacion(atributo, df_real_comp,

```

```

df_sintetico_comp,
                           fecha_inicio, fecha_fin, i + 1)
    figuras.append(fig)

return figuras

def crear_grafico_comparacion(atributo, df_real, df_sintetico,
fecha_inicio, fecha_fin, num_grafico):
    """Crea un gráfico de comparación para un atributo específico"""

    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(15, 10))

    # Gráfico 1: Serie temporal completa
    ax1.plot(df_real.index, df_real[atributo],
              color='blue', alpha=0.7, linewidth=1, label='Datos
    Reales')
    ax1.plot(df_sintetico.index, df_sintetico[atributo],
              color='red', alpha=0.7, linewidth=1, label='Datos
    Sintéticos')

    ax1.set_title(f'Comparación Temporal: {atributo}\nPeríodo:
{fecha_inicio.date()} - {fecha_fin.date()}',

    fontsize=14, fontweight='bold')
    ax1.set_ylabel(atributo, fontsize=12)
    ax1.legend(fontsize=10)
    ax1.grid(True, alpha=0.3)

    # Gráfico 2: Comparación por año (superpuesto)
    años_unicos = sorted(set(df_real['año'].unique()) |
set(df_sintetico['año'].unique()))

    # Paleta de colores para años
    colores_reales = plt.cm.Blues(np.linspace(0.4, 0.8,
len(años_unicos)))
    colores_sinteticos = plt.cm.Reds(np.linspace(0.4, 0.8,
len(años_unicos)))

    for i, año in enumerate(años_unicos):
        # Datos reales
        datos_año_real = df_real[df_real['año'] == año]
        if not datos_año_real.empty:
            ax2.plot(datos_año_real['dia_año'],
        datos_año_real[atributo],
                    color=colores_reales[i], alpha=0.7, linewidth=1.5,
                    label=f'Real {año}' if i < 3 else "")

        # Datos sintéticos
        datos_año_sint = df_sintetico[df_sintetico['año'] == año]
        if not datos_año_sint.empty:

```

```

        ax2.plot(datos_año_sint['dia_año'],
datos_año_sint[atributo],
            color=colores_sinteticos[i], alpha=0.7,
linewidth=1.5, linestyle='--',
            label=f'Sintético {año}' if i < 3 else "")

ax2.set_title(f'Comparación Estacional por Año: {atributo}', fontsize=14, fontweight='bold')
ax2.set_xlabel('Día del Año', fontsize=12)
ax2.set_ylabel(atributo, fontsize=12)
ax2.legend(fontsize=9)
ax2.grid(True, alpha=0.3)

plt.tight_layout()

# Añadir estadísticas comparativas
stats_text = f"""
Estadísticas Comparativas ({fecha_inicio.year}-{fecha_fin.year}):
Real - Media: {df_real[atributo].mean():.2f}, Std:
{df_real[atributo].std():.2f}
Sintético - Media: {df_sintetico[atributo].mean():.2f}, Std:
{df_sintetico[atributo].std():.2f}
Diferencia Media: {abs(df_real[atributo].mean() - df_sintetico[atributo].mean()):.2f}
"""

fig.text(0.02, 0.02, stats_text, fontsize=9,
         bbox=dict(boxstyle="round", pad=0.3",
facecolor="lightgray", alpha=0.7))

return fig

```

---

## Ejecución técnicas de generación de datos sintéticos

[ [Volver al índice](#) ]

---

```

def ejecutar_comparacion_pasado_completa(df_real, años_generar=4,
fecha_inicio_comparacion=None):
    """
    Ejecuta la generación sintética en el PASADO y la comparación
    visual
    """

```

```

print("Iniciando generación de datos sintéticos EN EL PASADO...")

# Método 1: Simulación Física en el pasado
print("Generando datos en el pasado con simulación física...")
df_sintetico_fisico_pasado = generar_datos_fisicos_pasado(df_real,
años_generar=años_generar)

# Método 2: Bootstrapping Estacional en el pasado
print("Generando datos en el pasado con bootstrapping
estacional...")
df_sintetico_bootstrap_pasado =
bootstrap_estacional_perturbado_pasado(df_real,
años_extra=años_generar)

# Combinar datos sintéticos del pasado con datos reales para
entrenamiento
print("Combinando datos sintéticos del pasado con datos
reales...")
df_combinado_fisico = pd.concat([df_sintetico_fisico_pasado,
df_real])
df_combinado_bootstrap = pd.concat([df_sintetico_bootstrap_pasado,
df_real])

# Comparar ambos métodos con datos reales
print("Generando comparaciones visuales...")

# Comparación método físico
figuras_fisico = comparar_datos_reales_vs_sinteticos(
    df_real=df_real,
    df_sintetico=df_sintetico_fisico_pasado,
    años_comparacion=años_generar,
    fecha_inicio_comparacion=fecha_inicio_comparacion
)

# Comparación método bootstrapping
figuras_bootstrap = comparar_datos_reales_vs_sinteticos(
    df_real=df_real,
    df_sintetico=df_sintetico_bootstrap_pasado,
    años_comparacion=años_generar,
    fecha_inicio_comparacion=fecha_inicio_comparacion
)

return {
    'sintetico_fisico_pasado': df_sintetico_fisico_pasado,
    'sintetico_bootstrap_pasado': df_sintetico_bootstrap_pasado,
    'combinado_fisico': df_combinado_fisico,
    'combinado_bootstrap': df_combinado_bootstrap,
}

```

```

        'figuras_fisico': figuras_fisico,
        'figuras_bootstrap': figuras_bootstrap
    }

try:
    # Asumiendo que daily_df es tu DataFrame real
    resultados = ejecutar_comparacion_pasado_completa(
        df_real=daily_df, # Tu DataFrame real
        años_generar=4,   # Generar 4 años de datos sintéticos en el
    pasado
        fecha_inicio_comparacion=None # Se calculará automáticamente
    )

    print("¡Proceso completado exitosamente!")

    # Mostrar rangos de fechas
    print("\nRangos de fechas:")
    print(f"Real: {daily_df.index.min()} a {daily_df.index.max()}")
    print(f"Sintético físico:
{resultados['sintetico_fisico_pasado'].index.min()} a
{resultados['sintetico_fisico_pasado'].index.max()}")
    print(f"Sintético bootstrap:
{resultados['sintetico_bootstrap_pasado'].index.min()} a
{resultados['sintetico_bootstrap_pasado'].index.max()}")
    print(f"Combinado físico:
{resultados['combinado_fisico'].index.min()} a
{resultados['combinado_fisico'].index.max()")

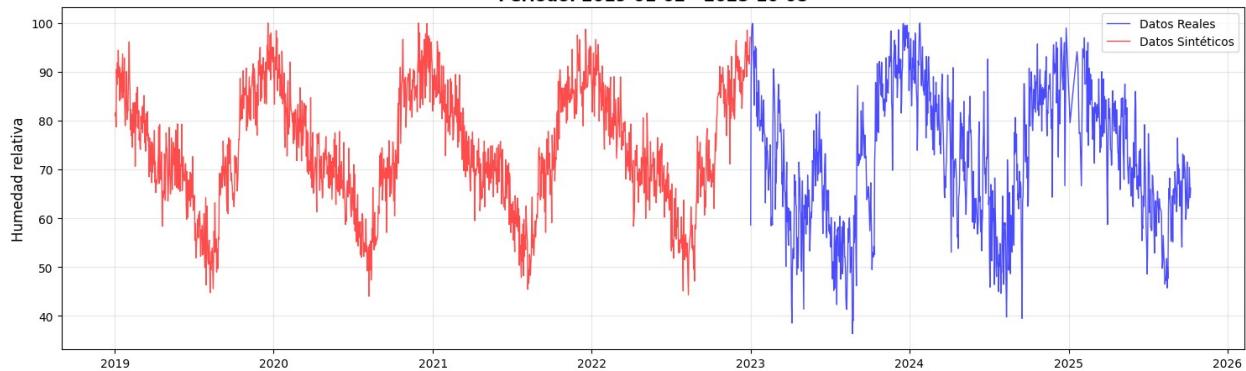
except Exception as e:
    print(f"Error durante la ejecución: {e}")

```

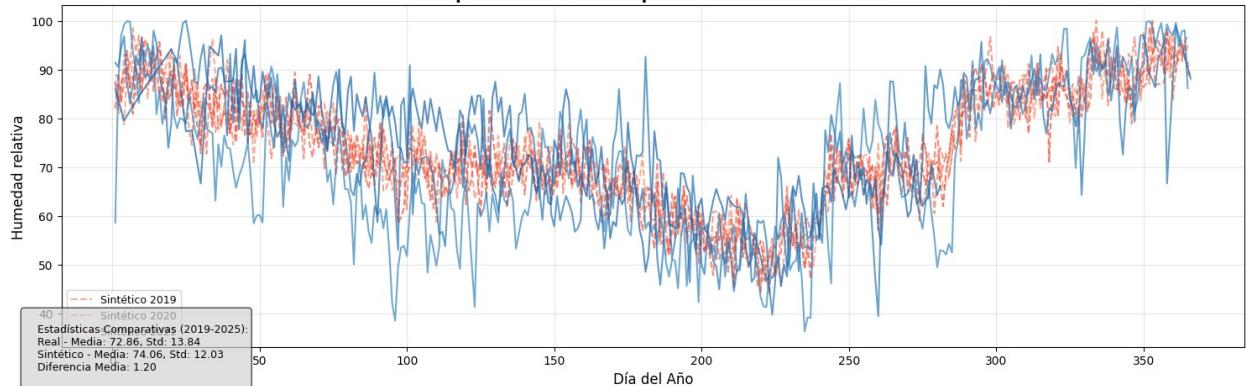
Iniciando generación de datos sintéticos EN EL PASADO...  
 Generando datos en el pasado con simulación física...  
 Generando datos en el pasado con bootstrapping estacional...  
 Combinando datos sintéticos del pasado con datos reales...  
 Generando comparaciones visuales...  
 ¡Proceso completado exitosamente!

Rangos de fechas:  
 Real: 2023-01-01 00:00:00 a 2025-10-08 00:00:00  
 Sintético físico: 2019-01-02 00:00:00 a 2022-12-31 00:00:00  
 Sintético bootstrap: 2019-01-01 00:00:00 a 2022-12-31 00:00:00  
 Combinado físico: 2019-01-02 00:00:00 a 2025-10-08 00:00:00

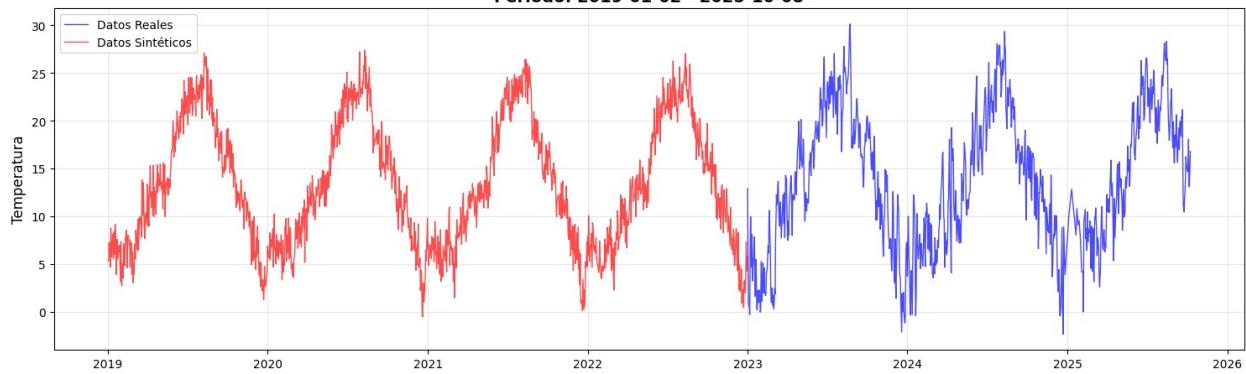
Comparación Temporal: Humedad relativa  
Período: 2019-01-02 - 2025-10-08



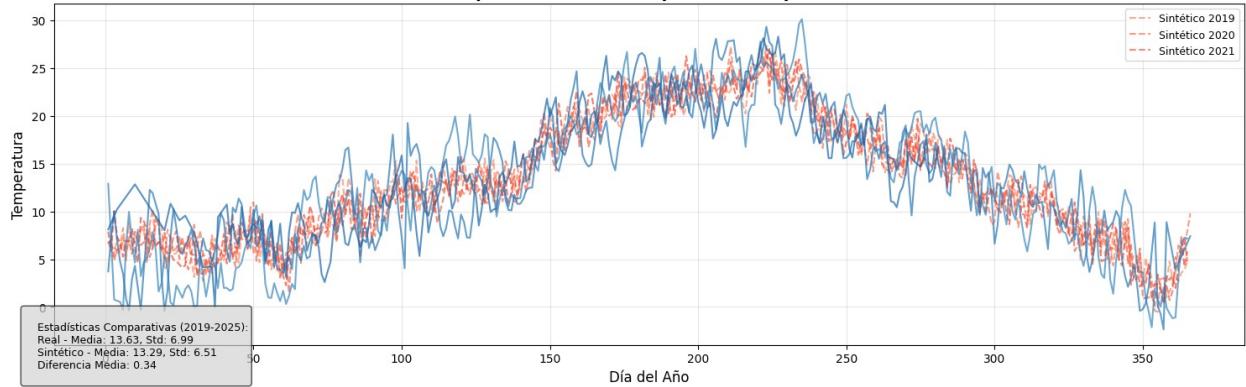
Comparación Estacional por Año: Humedad relativa



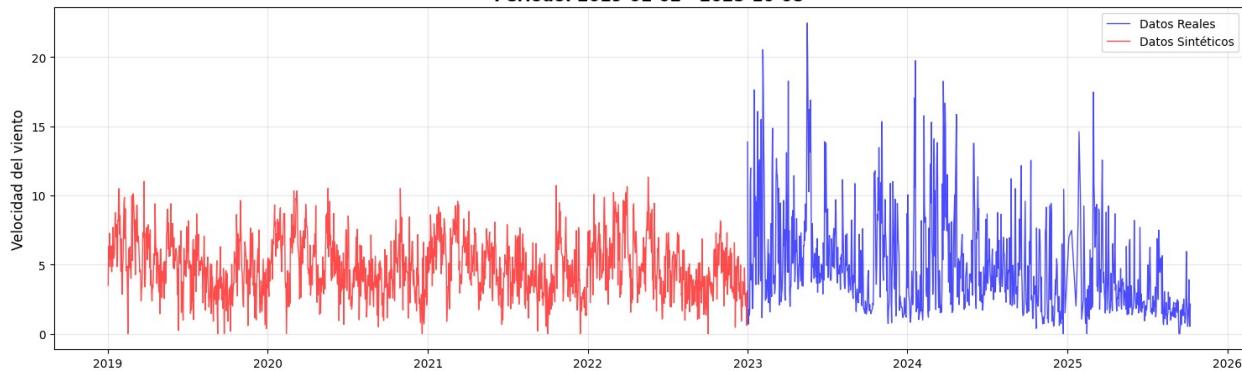
Comparación Temporal: Temperatura  
Período: 2019-01-02 - 2025-10-08



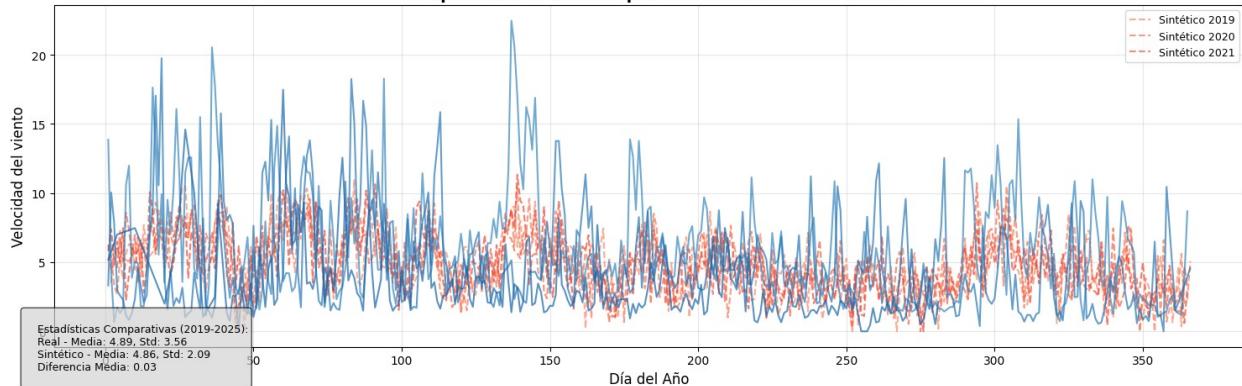
Comparación Estacional por Año: Temperatura



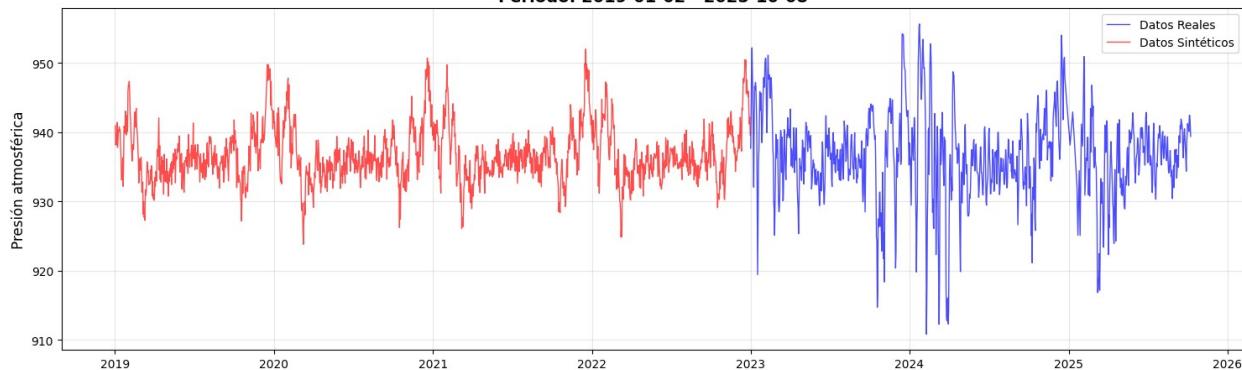
Comparación Temporal: Velocidad del viento  
Período: 2019-01-02 - 2025-10-08



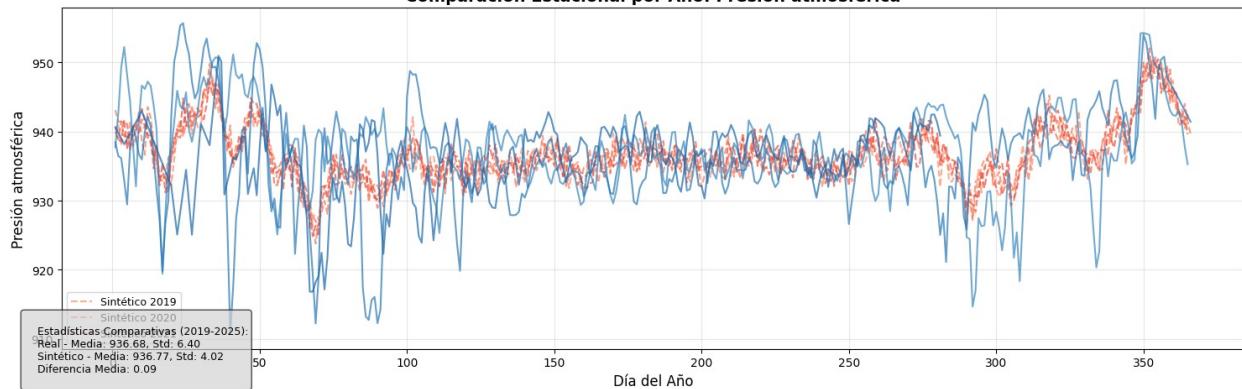
Comparación Estacional por Año: Velocidad del viento



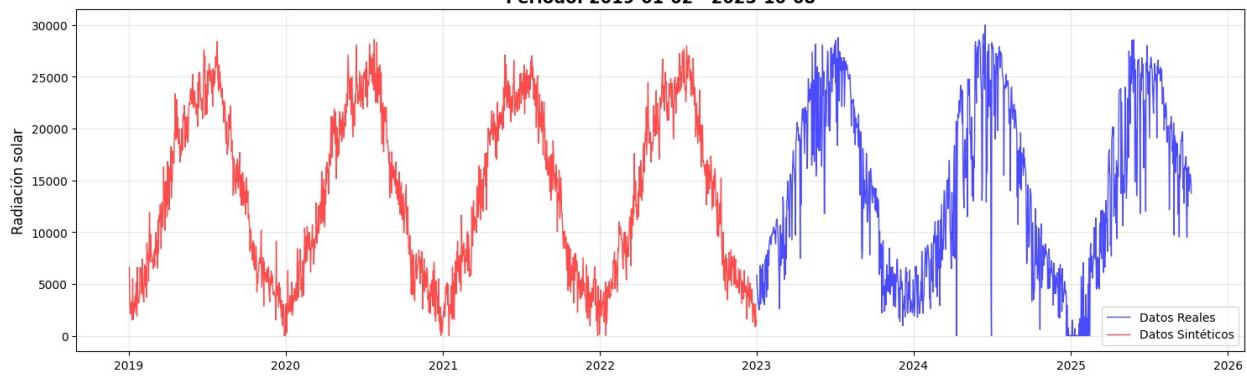
Comparación Temporal: Presión atmosférica  
Período: 2019-01-02 - 2025-10-08



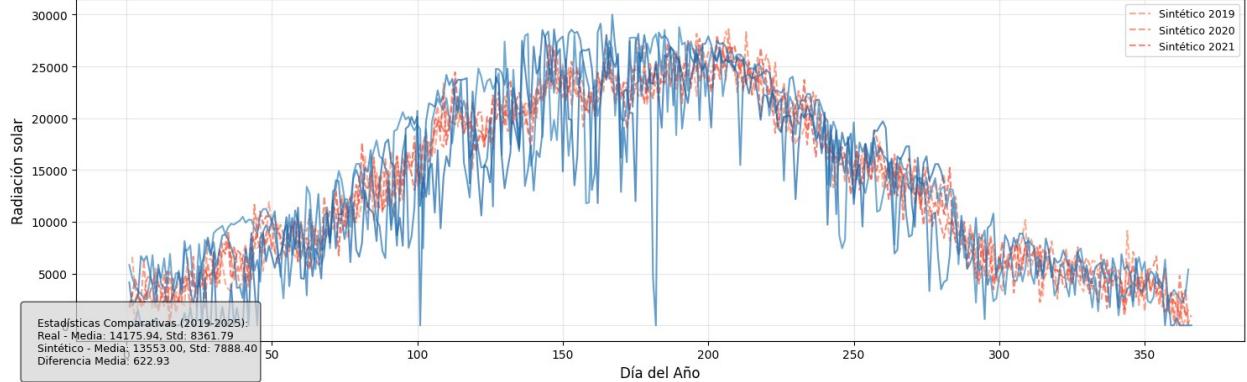
Comparación Estacional por Año: Presión atmosférica



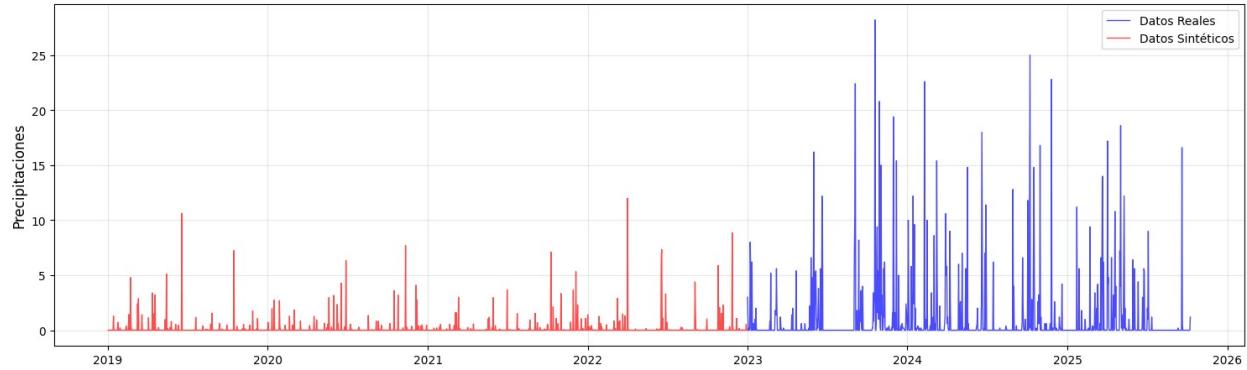
Comparación Temporal: Radiación solar  
Período: 2019-01-02 - 2025-10-08



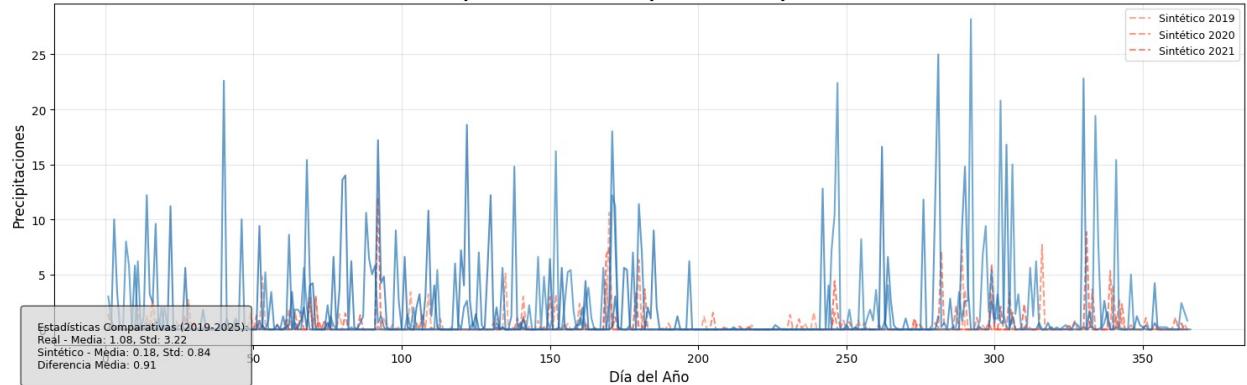
Comparación Estacional por Año: Radiación solar



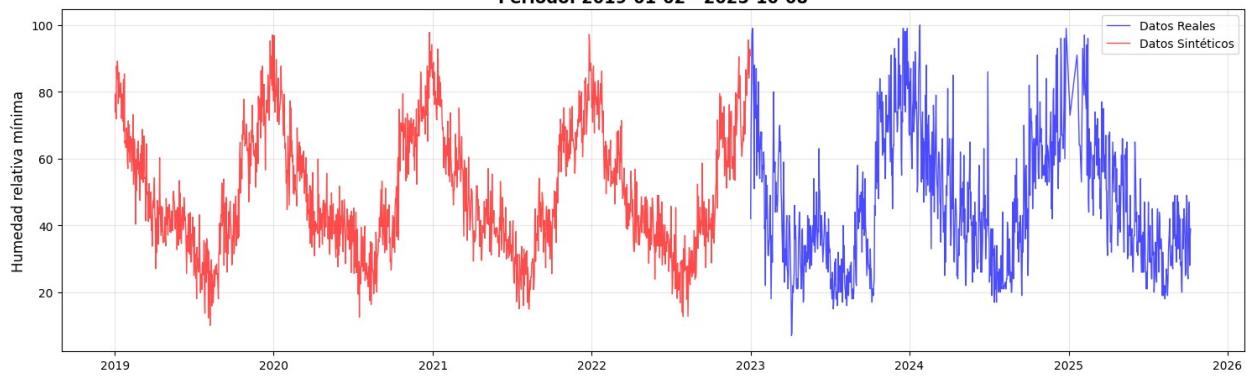
Comparación Temporal: Precipitaciones  
Período: 2019-01-02 - 2025-10-08



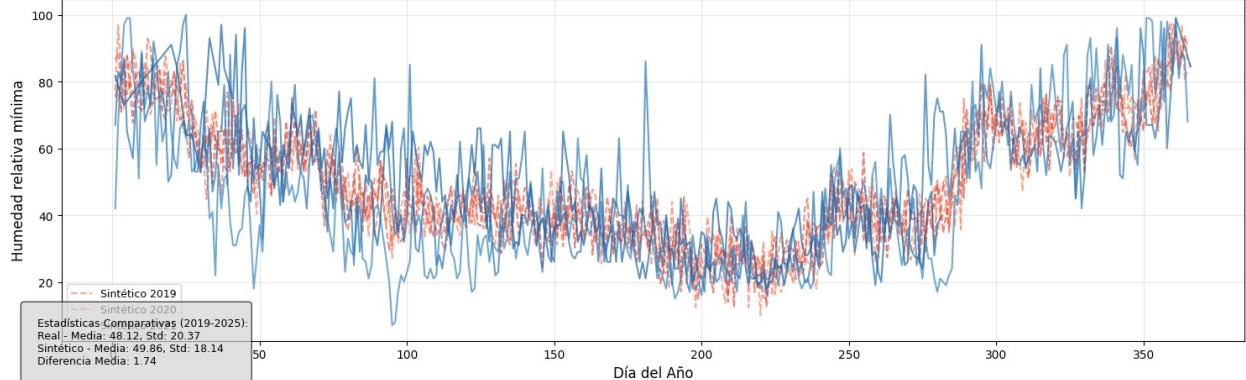
Comparación Estacional por Año: Precipitaciones



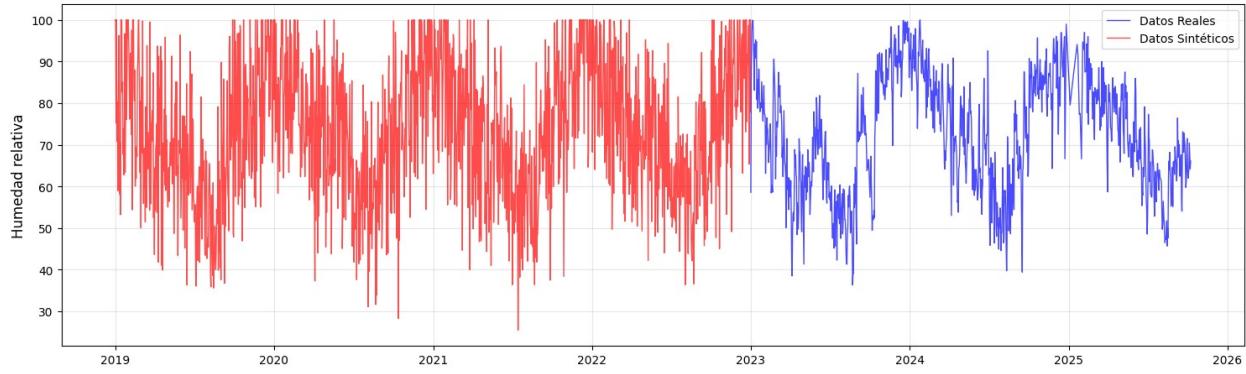
Comparación Temporal: Humedad relativa mínima  
Período: 2019-01-02 - 2025-10-08



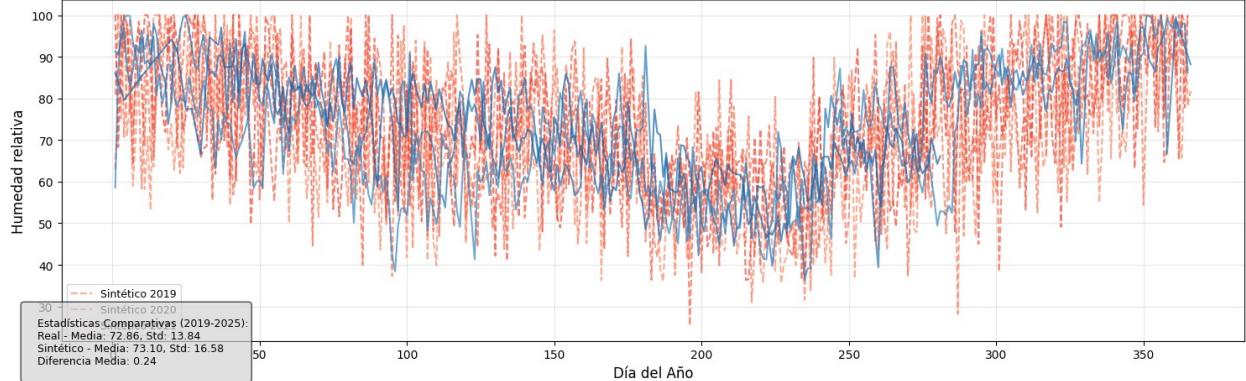
Comparación Estacional por Año: Humedad relativa mínima



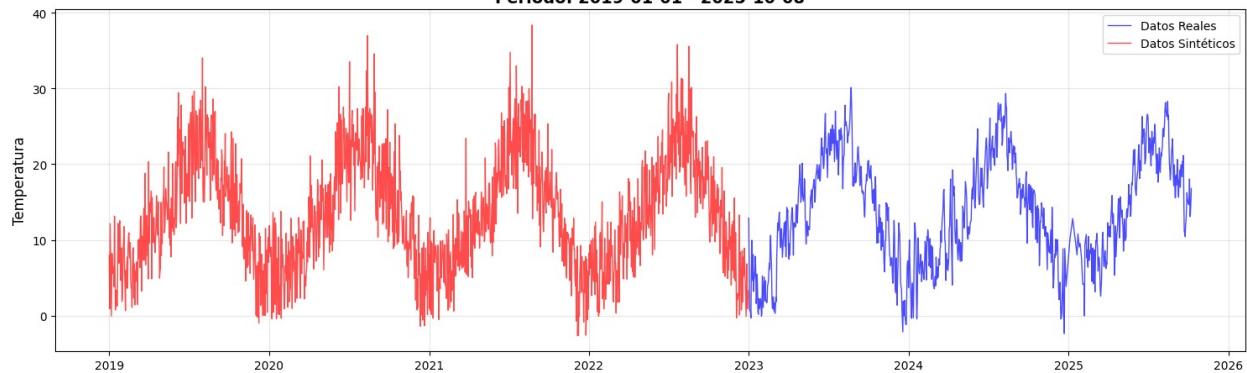
Comparación Temporal: Humedad relativa  
Período: 2019-01-01 - 2025-10-08



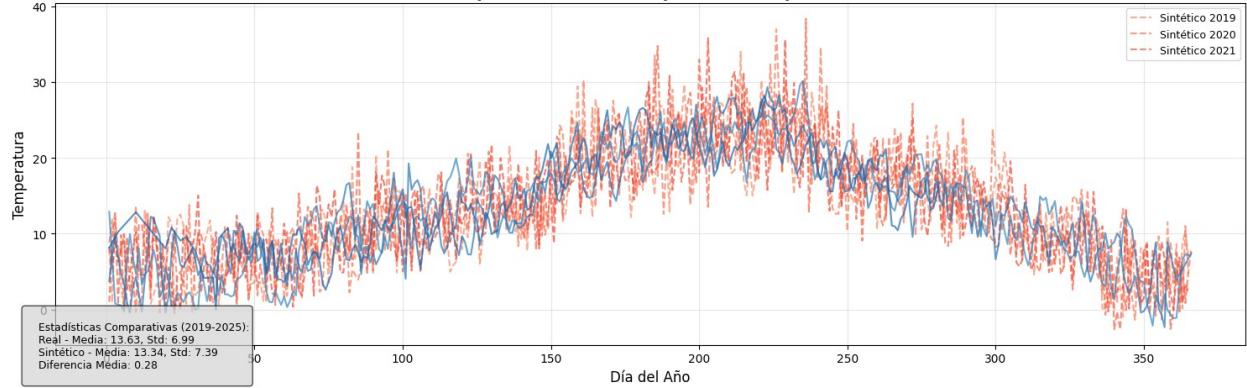
Comparación Estacional por Año: Humedad relativa



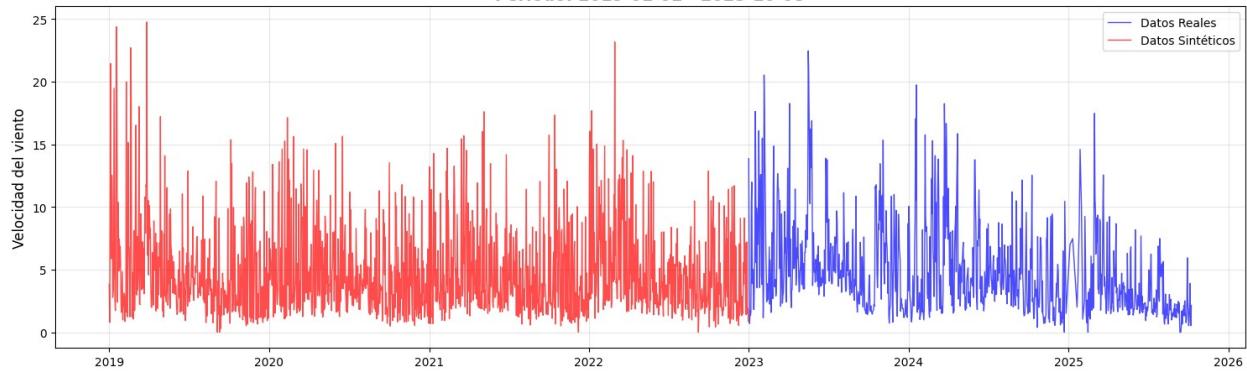
Comparación Temporal: Temperatura  
Período: 2019-01-01 - 2025-10-08



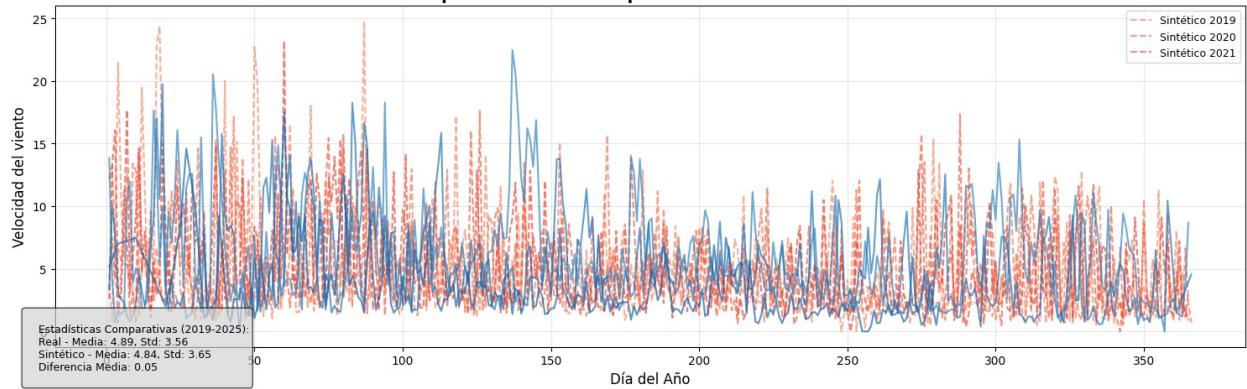
Comparación Estacional por Año: Temperatura



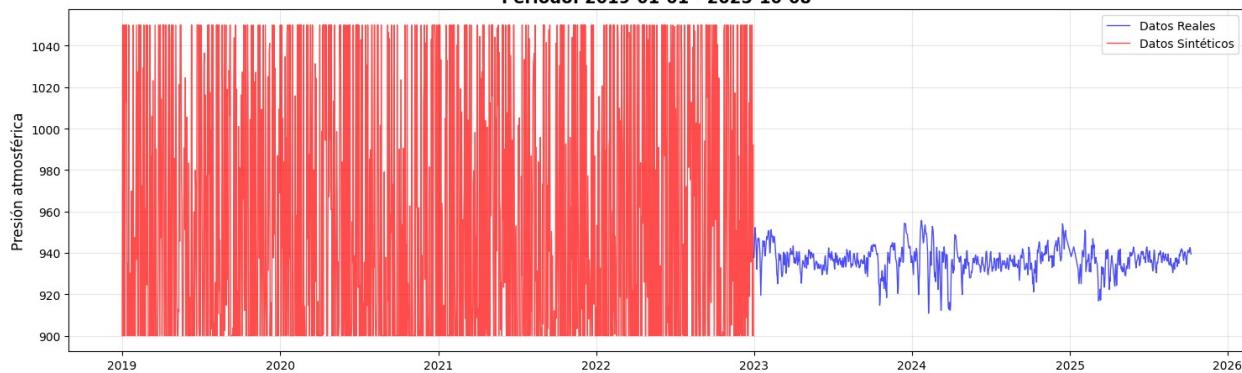
Comparación Temporal: Velocidad del viento  
Período: 2019-01-01 - 2025-10-08



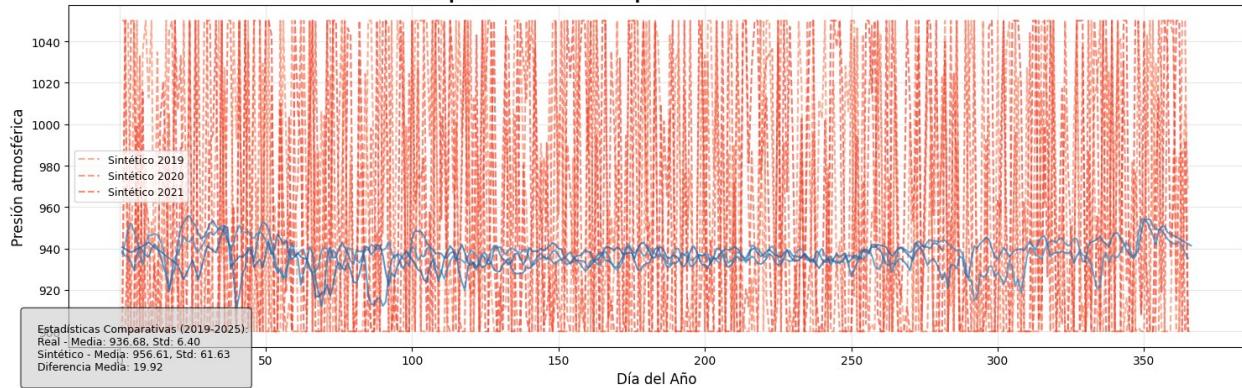
Comparación Estacional por Año: Velocidad del viento



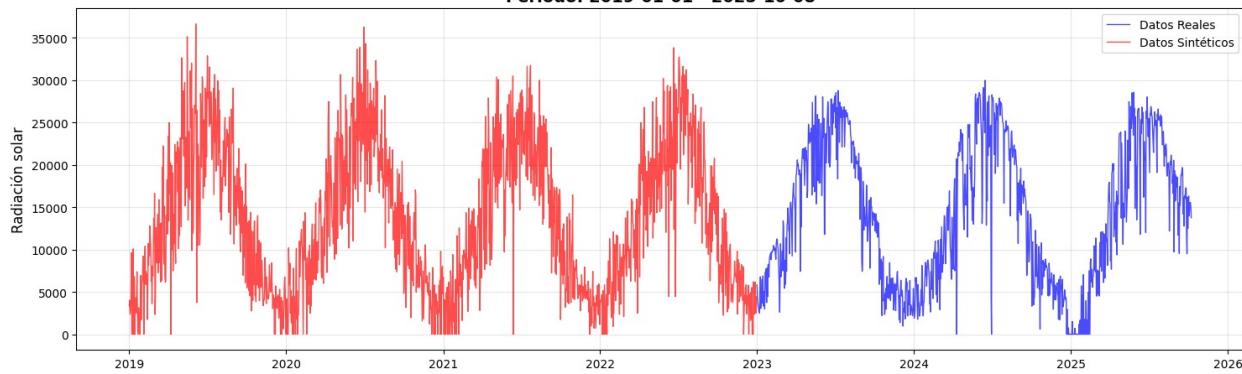
Comparación Temporal: Presión atmosférica  
Período: 2019-01-01 - 2025-10-08



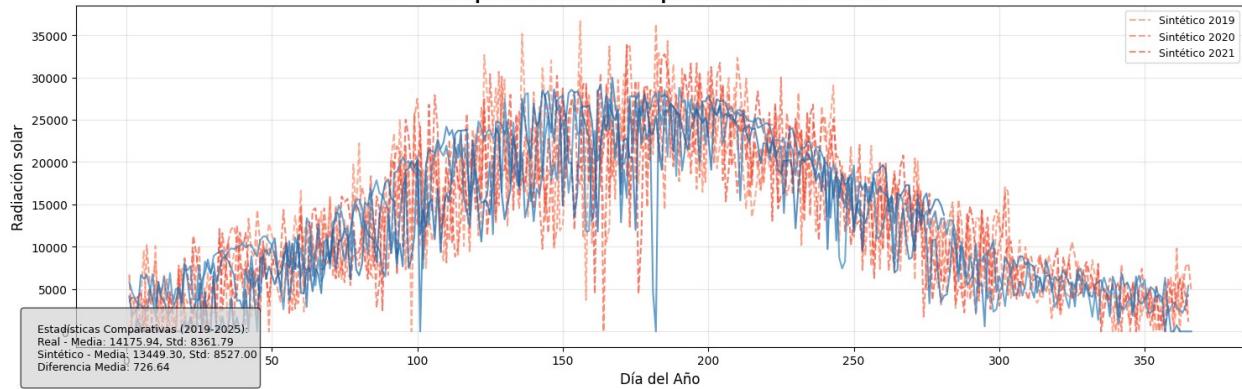
Comparación Estacional por Año: Presión atmosférica



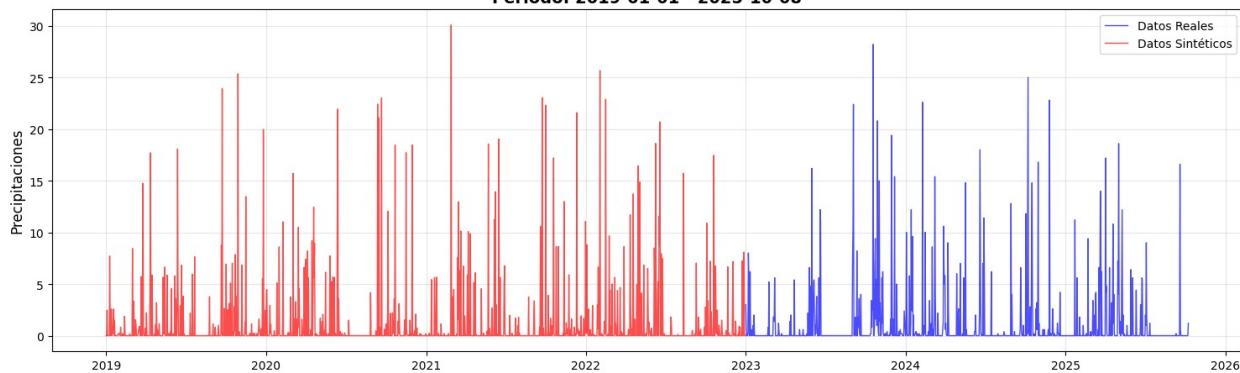
Comparación Temporal: Radiación solar  
Período: 2019-01-01 - 2025-10-08



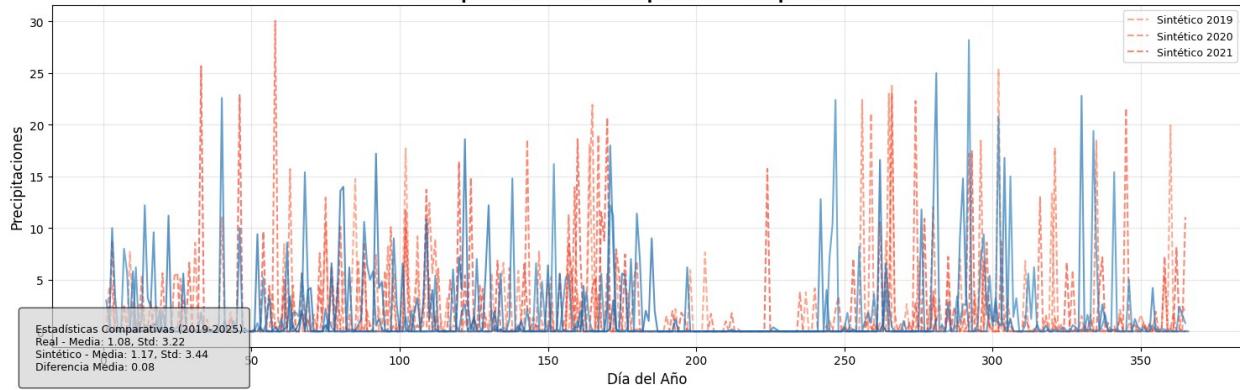
Comparación Estacional por Año: Radiación solar



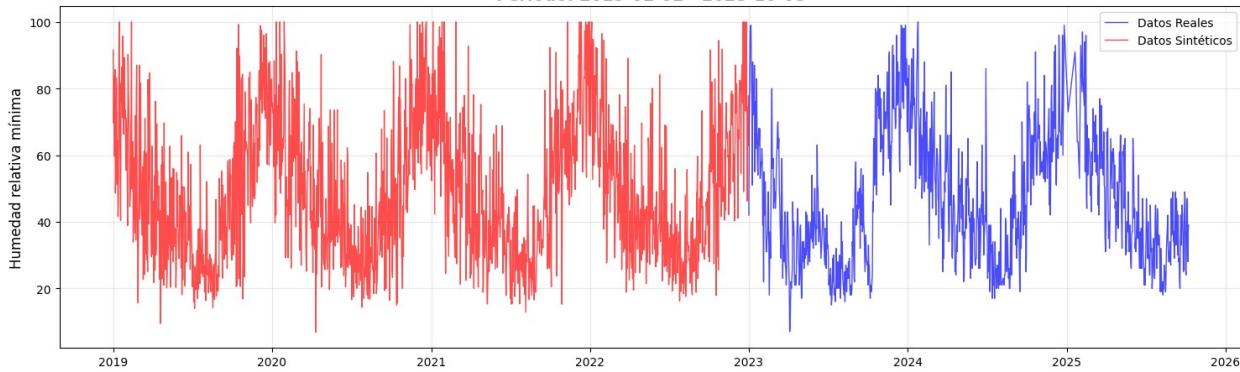
Comparación Temporal: Precipitaciones  
Período: 2019-01-01 - 2025-10-08



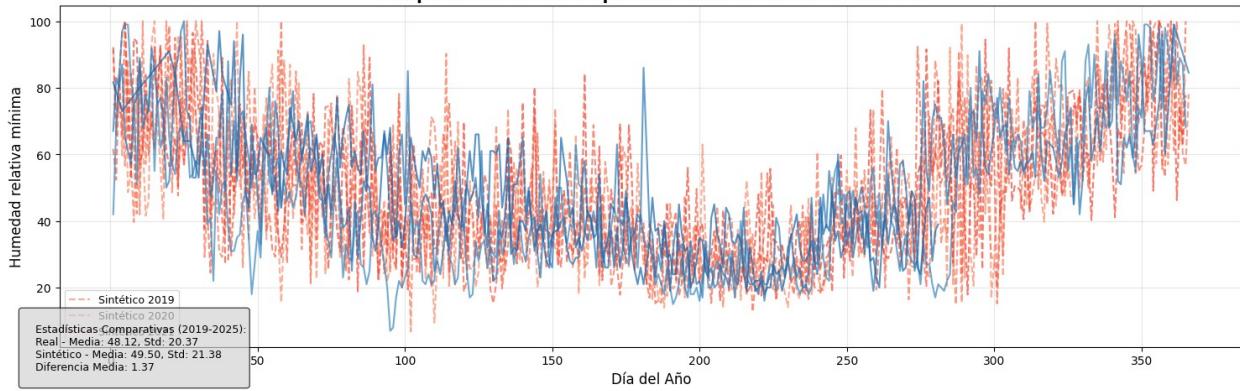
Comparación Estacional por Año: Precipitaciones



Comparación Temporal: Humedad relativa mínima  
Período: 2019-01-01 - 2025-10-08



Comparación Estacional por Año: Humedad relativa mínima



---

## Guardar datos de entrenamiento para un futuro uso en los modelos

[ [Volver al índice](#) ]

---

```
import os

def crear_directorio_si_no_existe(ruta):
    """Crea un directorio si no existe"""
    if not os.path.exists(ruta):
        os.makedirs(ruta)
    print(f"Directorio creado: {ruta}")

# Función adicional para guardar los datos combinados para
entrenamiento
def guardar_datos_entrenamiento(resultados,
ruta_guardado=".data/datos_entrenamiento"):
    """Guarda los datos combinados para entrenamiento de modelos"""
    crear_directorio_si_no_existe(ruta_guardado)

    resultados['combinado_fisico'].to_csv(f"{ruta_guardado}/datos_entrenamiento_fisico.csv")
    resultados['combinado_bootstrap'].to_csv(f"{ruta_guardado}/datos_entrenamiento_bootstrap.csv")

    print(f"Datos de entrenamiento guardados en: {ruta_guardado}")

guardar_datos_entrenamiento(resultados)
Datos de entrenamiento guardados en: ./data/datos_entrenamiento
```

### #### Importar datos de entrenamiento combinados (sintéticos y reales) a DataFrame

```
synthetic_data_path = base_path / "datos_entrenamiento"  
physic_data_path = synthetic_data_path /  
"datos_entrenamiento_fisico.csv"  
  
daily_combined_df = pd.read_csv(physic_data_path, index_col='Fecha',  
parse_dates=True)  
  
display(daily_combined_df)
```

Fecha	Humedad relativa	Temperatura	Velocidad del viento	\
2019-01-02	80.968206	5.286936	3.492735	
2019-01-03	81.666437	5.476028	6.302758	
2019-01-04	78.686045	7.230553	5.411157	
2019-01-05	90.455253	6.327118	7.261875	
2019-01-06	87.356777	7.239543	4.845675	
...	...	...	...	...
2025-10-04	63.895833	18.047917	1.927083	
2025-10-05	70.437500	13.093750	3.895833	
2025-10-06	67.520833	13.386458	1.625000	
2025-10-07	64.229167	14.877083	0.520833	
2025-10-08	66.125000	16.794792	2.145833	
Fecha	Presión atmosférica	Radiación solar	Precipitaciones	\
2019-01-02	940.956288	6611.525552	0.0	
2019-01-03	938.157792	3528.141094	0.0	
2019-01-04	939.646331	2612.183850	0.0	
2019-01-05	938.273529	2092.541036	0.0	
2019-01-06	941.433851	2317.185347	0.0	
...	...	...	...	...
2025-10-04	940.113542	14664.000000	0.0	
2025-10-05	942.435417	15558.000000	0.0	
2025-10-06	942.463542	15573.000000	0.0	
2025-10-07	941.246875	14898.000000	0.0	
2025-10-08	939.396875	13752.000000	1.2	
Fecha	Humedad relativa mínima			
2019-01-02	73.991470			
2019-01-03	79.395424			
2019-01-04	71.876329			
2019-01-05	87.638952			
2019-01-06	83.844725			
...	...			
2025-10-04	44.000000			

```
2025-10-05          47.000000
2025-10-06          28.000000
2025-10-07          37.000000
2025-10-08          39.000000
```

```
[2472 rows x 7 columns]
```

## Normalización de los datos

[\[ Volver al índice \]](#)

Usamos `ColumnsTransformer`, que aunque no se haya visto en el contenido de la asignatura, nos permite aplicar diferentes scalers a distintas columnas, adaptándose mejor según el tipo de datos que contiene la columna, por ejemplo:

Usamos `StandarScaler` cuando:

- Cuando los datos siguen una distribución normal (o cerca de ello).

Usamos `RobustScaler` cuando:

- Si tus datos tienen outliers fuertes. (Esto se puede observar en el diagrama de caja que hemos obtenido anteriormente, aplicamos `RobustScaler` a aquellas variables cuyos puntos circulares negros(Valores fuera del minimo o maximo) esten a un  $3 \times \text{IQR}$ )

```
from scipy.stats import normaltest
import math

sns.set(style="whitegrid")

columnas_numericas =
daily_df.select_dtypes(include=['number']).columns
n_cols = 3 # Número de gráficas por fila (ajústalo a tu gusto)
n_filas = math.ceil(len(columnas_numericas) / n_cols)

# Crear figura con subplots
fig, axes = plt.subplots(n_filas, n_cols, figsize=(5*n_cols,
4*n_filas))
axes = axes.flatten() # Aplanar el arreglo de axes para iterar
#fácilmente

# Dibujar histogramas
for i, columna in enumerate(columnas_numericas):
    sns.histplot(daily_df[columna], kde=True, bins=30,
```

```
color='skyblue', ax=axes[i])
    axes[i].set_title(f"Distribución de {columna}")
    axes[i].set_xlabel(columna)
    axes[i].set_ylabel("Frecuencia")

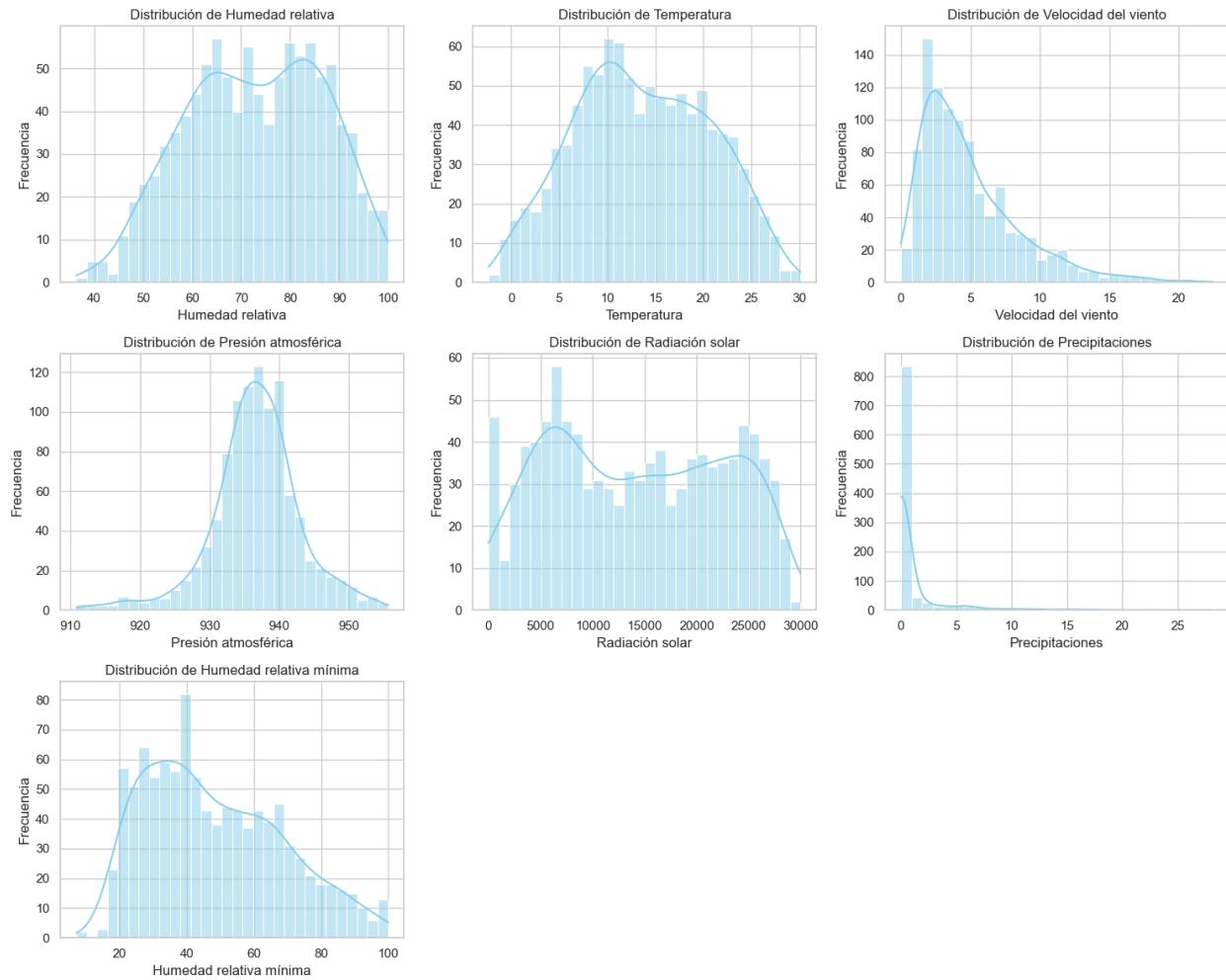
# Si sobran subplots vacíos, los eliminamos
for j in range(i+1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout() # Ajusta automáticamente para que no se solapen
las gráficas
plt.show()

# --- 2. Calcular los valores atípicos (outliers) usando 3xIQR ---
outliers_info = {}

for columna in columnas_numericas:
    Q1 = daily_df[columna].quantile(0.25)
    Q3 = daily_df[columna].quantile(0.75)
    IQR = Q3 - Q1
    limite_inferior = Q1 - 3 * IQR
    limite_superior = Q3 + 3 * IQR

    outliers = daily_df[(daily_df[columna] < limite_inferior) |
(daily_df[columna] > limite_superior)][columna]
    outliers_info[columna] = len(outliers)
```



## Test de normalidad y recomendación de escalado

[ [Volver al índice](#) ]

```
recomendaciones = {}

for columna in columnas_numericas:
    estadistico, p_valor = normaltest(daily_df[columna].dropna())
    es_normal = p_valor > 0.05 # Si p > 0.05, no se rechaza la
    #normalidad
    tiene_outliers = outliers_info[columna] > 0

    if tiene_outliers:
        escalador = "RobustScaler"
    elif es_normal:
```

```

        escalador = "StandardScaler"
    else:
        escalador = "StandardScaler (aproximado)"

recomendaciones[columna] = {
    "p_valor_normaltest": p_valor,
    "outliers_(3xIQR)": outliers_info[columna],
    "escalador_recomendado": escalador
}

# --- 4. Mostrar resumen de recomendaciones ---
recomendaciones_df = pd.DataFrame(recomendaciones).T
print("\n[] Recomendaciones de escalado:\n")
print(recomendaciones_df)

```

[] Recomendaciones de escalado:

	p_valor_normaltest	outliers_(3xIQR)	escalador_recomendado
Humedad relativa	0.0	0	StandardScaler (aproximado)
Temperatura	0.0	0	StandardScaler (aproximado)
Velocidad del viento	0.0	4	RobustScaler
Presión atmosférica	0.0	5	RobustScaler
Radiación solar	0.0	0	StandardScaler (aproximado)
Precipitaciones	0.0	178	RobustScaler
Humedad relativa mínima	0.0	0	StandardScaler (aproximado)

## Desarrollo de Modelos

[ [\[\] Volver al índice](#) ]

Fase	Objetivo	Modelos sugeridos
<b>Fase 1</b>	Exploración y análisis de estacionalidad	SARIMA / SARIMAX
<b>Fase 2</b>	Modelado multivariable lineal	VAR
<b>Fase 3</b>	Primeros experimentos con LSTM / GRU	LSTM, GRU
<b>Fase 4</b>	Predicción multi-step	LSTM / TCN con ventanas móviles
<b>Fase 5</b>	Predicción end-to-end a largo plazo	Informer, Autoformer, PatchTST
<b>Fase 6</b>	Implementación avanzada en entorno agrícola	<b>SageFormer</b>
<ul style="list-style-type: none"> <li>• Evaluación de modelos: <ul style="list-style-type: none"> <li>– Utilizar métricas como accuracy, precisión, recall, F1-score y AUC-ROC, teniendo en cuenta el desequilibrio de clases.</li> <li>– Aplicar validación cruzada para evaluar la capacidad de generalización de los modelos.</li> </ul> </li> </ul>		

## Importación de librerías necesarias

[ [Volver al índice](#) ]

```
import joblib
#warnings.filterwarnings('ignore')

# Scikit-learn para pipelines
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.compose import ColumnTransformer
from sklearn.base import BaseEstimator, TransformerMixin

# Modelos de series temporales
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.vector_ar.var_model import VAR
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

# LSTM
from tensorflow.keras.models import Sequential
```

```

from tensorflow.keras.layers import LSTM, Dense, Dropout,
BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau

# Utilidades
from typing import Dict, List, Tuple

print("Librerías importadas correctamente")

2026-02-13 08:27:33.881388: I
tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow
binary is optimized to use available CPU instructions in performance-
critical operations.
To enable the following instructions: AVX2 FMA, in other operations,
rebuild TensorFlow with the appropriate compiler flags.

Librerías importadas correctamente

```

---

## Parámetros de configuración del proceso de entrenamiento

### Parámetros de los modelos

- `SARIMA_ORDER (p, d, q)`: define la estructura autorregresiva, de diferenciación y de media móvil del modelo SARIMA.
- `SARIMA_SEASONAL_ORDER (P, D, Q, s)`: especifica la componente estacional del modelo SARIMA, donde `s = 30` representa una estacionalidad mensual (30 días).
- `VAR_MAXLAGS`: número máximo de retardos utilizados en el modelo VAR para capturar dependencias temporales entre variables.

### Parámetros de predicción

- `PREDICTION_HORIZON`: horizonte temporal de predicción en días (240 días  $\approx$  8 meses).
- `TEST_SIZE`: número de días reservados para el conjunto de test (180 días  $\approx$  6 meses).

### Rutas de almacenamiento

- `MODEL_PATH`: directorio donde se guardan los modelos entrenados.
- `RESULTS_PATH`: directorio donde se almacenan los resultados y métricas de los experimentos.

## Gestión de directorios

- `setup_directories()`: crea automáticamente las carpetas necesarias para modelos y resultados si no existen.

## Datos de entrenamiento

- `SYNTHETIC_DATA_PATH`: ruta al directorio que contiene los datos de entrenamiento.
- `PHYSIC_DATA_PATH`: archivo CSV con los datos de entrenamiento basados en modelos físicos.

```
class Config:  
    """Clase de configuración para el proyecto"""\n\n    # Parámetros de modelos  
    SARIMA_ORDER = (1, 1, 1)  
    SARIMA_SEASONAL_ORDER = (1, 1, 1, 30)  
    VAR_MAXLAGS = 15\n\n    # Parámetros de predicción  
    PREDICTION_HORIZON = 240 # 8 meses ≈ 240 días  
    TEST_SIZE = 180 # 6 meses para test\n\n    # Rutas para guardar modelos  
    MODEL_PATH = base_path / "models"  
    RESULTS_PATH = base_path / "results"\n\n    @classmethod  
    def setup_directories(cls):  
        os.makedirs(cls.MODEL_PATH, exist_ok=True)  
        os.makedirs(cls.RESULTS_PATH, exist_ok=True)\n\n    SYNTHETIC_DATA_PATH = base_path / "datos_entrenamiento"  
    PHYSIC_DATA_PATH = SYNTHETIC_DATA_PATH /  
        "datos_entrenamiento_fisico.csv"\n\n    # Configurar directorios  
    Config.setup_directories()
```

---

## Wrappers de modelos para integración en Pipelines

[ [Volver al índice](#) ]

Estas clases implementan *wrappers* que adaptan modelos de series temporales al interfaz de `scikit-learn`, permitiendo su uso dentro de `Pipeline` y `GridSearch`.

En particular, el wrapper `SarimaModel` encapsula un modelo **SARIMA**:

- Hereda de `BaseEstimator` y `TransformerMixin` para cumplir con la API de scikit-learn.
- Permite seleccionar la **variable objetivo** mediante el parámetro `column`.
- Expone los hiperparámetros del modelo (`order`, `seasonal_order`, `trend`) para su ajuste y validación.
- Implementa los métodos `fit` y `predict` adaptados a series temporales.
- Define `get_params` y `set_params` para facilitar la optimización automática de hiperparámetros.

Este enfoque unifica modelos estadísticos externos con el flujo de trabajo estándar de scikit-learn, facilitando la experimentación, reutilización y comparación de modelos.

---

## Wrapper modelo SARIMA

[ [Volver al índice](#) ]

---

```
class SarimaModel(BaseEstimator, TransformerMixin):  
    """Wrapper de SARIMA para scikit-learn"""  
  
    def __init__(self, column: str, order: tuple = (1,1,1),  
                 seasonal_order: tuple = (1,1,1,30), trend: str =  
                 'c'):  
        self.column = column  
        self.order = order  
        self.seasonal_order = seasonal_order  
        self.trend = trend  
        self.model = None  
        self.fitted_model = None  
  
    def fit(self, X, y=None):  
        # Asegurarse de que X es una Serie temporal  
        if isinstance(X, pd.DataFrame):  
            series = X[self.column]  
        else:  
            series = X  
  
        self.model = SARIMAX(series,  
                            order=self.order,  
                            seasonal_order=self.seasonal_order,  
                            trend=self.trend,
```

```

        enforce_stationarity=False,
        enforce_invertibility=False)

    self.fitted_model = self.model.fit(disp=False)
    return self

def predict(self, X, n_periods: int = 240):
    # Predicción para n períodos futuros
    forecast = self.fitted_model.get_forecast(steps=n_periods)
    return forecast.predicted_mean

def get_params(self, deep=True):
    return {'column': self.column,
            'order': self.order,
            'seasonal_order': self.seasonal_order,
            'trend': self.trend}

def set_params(self, **parameters):
    for parameter, value in parameters.items():
        setattr(self, parameter, value)
    return self

```

---

## Wrapper modelo SARIMAX

[ [Volver al índice](#) ]

---

```

class SarimaxModel(BaseEstimator, TransformerMixin):
    """Wrapper de SARIMAX para scikit-learn"""

    def __init__(self, target_col: str, exog_cols: list,
                 order: tuple = (1,1,1), seasonal_order: tuple =
(1,1,1,30)):
        self.target_col = target_col
        self.exog_cols = exog_cols
        self.order = order
        self.seasonal_order = seasonal_order
        self.model = None
        self.fitted_model = None

    def fit(self, X, y=None):
        endog = X[self.target_col]
        exog = X[self.exog_cols]

```

```

        self.model = SARIMAX(endog, exog=exog,
                              order=self.order,
                              seasonal_order=self.seasonal_order,
                              enforce_stationarity=False,
                              enforce_invertibility=False)

        self.fitted_model = self.model.fit(disp=False)
        return self

    def predict(self, X, n_periods: int = 240):
        # Para predicción necesitamos las variables exógenas futuras
        if len(X) < n_periods:
            raise ValueError("X debe contener al menos n_periods filas
para las variables exógenas")

        exog_future = X[self.exog_cols].iloc[:n_periods]
        forecast = self.fitted_model.get_forecast(steps=n_periods,
exog=exog_future)
        return forecast.predicted_mean

```

---

## Wrapper modelo VAR

[ [Volver al índice](#) ]

---

```

class VarModel(BaseEstimator, TransformerMixin):
    """Wrapper de VAR para scikit-learn"""

    def __init__(self, maxlags: int = 15, ic: str = 'aic'):
        self.maxlags = maxlags
        self.ic = ic
        self.model = None
        self.fitted_model = None

    def fit(self, X, y=None):
        self.model = VAR(X)
        self.fitted_model = self.model.fit(maxlags=self.maxlags,
ic=self.ic)
        return self

    def predict(self, X, n_periods: int = 240):
        # VAR usa los últimos valores del conjunto de entrenamiento

```

```

para predecir
    lag_order = self.fitted_model.k_ar
    last_observations = X.values[-lag_order:]

    forecast = self.fitted_model.forecast(y=last_observations,
steps=n_periods)
    return pd.DataFrame(forecast, columns=X.columns,
                           index=pd.date_range(start=X.index[-1] +
pd.Timedelta(days=1),
                           periods=n_periods,
                           freq='D'))

```

---

## Wrapper modelo LSTM

[ [Volver al índice](#) ]

---

```

class LSTMModel(BaseEstimator, TransformerMixin):
    """Wrapper de LSTM multivariante para scikit-learn"""

    def __init__(self, sequence_length: int = 365, lstm_units: list =
[128, 64, 32],
                 dropout_rate: float = 0.3, learning_rate: float =
0.001,
                 epochs: int = 100, batch_size: int = 32,
                 validation_split: float = 0.1, patience: int = 20):
        ...

    Args:
        sequence_length: Longitud de la secuencia temporal (días)
        lstm_units: Lista con número de unidades por capa LSTM
        dropout_rate: Tasa de dropout para regularización
        learning_rate: Tasa de aprendizaje del optimizador
        epochs: Número máximo de épocas de entrenamiento
        batch_size: Tamaño del batch
        validation_split: Porcentaje para validación
        patience: Paciencia para early stopping
    ...

    self.sequence_length = sequence_length
    self.lstm_units = lstm_units
    self.dropout_rate = dropout_rate
    self.learning_rate = learning_rate
    self.epochs = epochs
    self.batch_size = batch_size

```

```

    self.validation_split = validation_split
    self.patience = patience

    self.model = None

    from sklearn.preprocessing import MinMaxScaler

    self.scaler = MinMaxScaler(feature_range=(0, 1))
    self.last_sequence = None
    self.feature_names = None

def fit(self, X, y=None):
    """
    Entrenar modelo LSTM multivariante
    X: DataFrame con múltiples variables temporales
    """
    # Guardar nombres de características
    self.feature_names = X.columns.tolist()

    scaled_data = self.scaler.fit_transform(X)

    # Crear secuencias
    X_seq, y_seq = self._create_sequences(scaled_data)

    # Guardar última secuencia para predicciones futuras
    self.last_sequence = X_seq[-1:].copy()

    # Construir modelo
    self.model = self._build_model(X_seq.shape[1:],
y_seq.shape[1])

    # Callbacks para evitar sobreajuste
    callbacks = [
        EarlyStopping(
            monitor='val_loss',
            patience=self.patience,
            restore_best_weights=True,
            verbose=0
        ),
        ReduceLROnPlateau(
            monitor='val_loss',
            factor=0.5,
            patience=self.patience // 2,
            verbose=0
        )
    ]

    # Entrenar modelo
    self.model.fit(
        X_seq, y_seq,

```

```

        validation_split=self.validation_split,
        epochs=self.epochs,
        batch_size=self.batch_size,
        callbacks=callbacks,
        verbose=0,
        shuffle=False # Importante para series temporales
    )

    return self

def _create_sequences(self, data):
    """Crear secuencias para LSTM multivariante"""
    X, y = [], []

    for i in range(self.sequence_length, len(data)):
        # Input: secuencia de 'sequence_length' pasos
        X.append(data[i-self.sequence_length:i])
        # Output: siguiente paso (todas las variables)
        y.append(data[i])

    return np.array(X), np.array(y)

def _build_model(self, input_shape, output_dim):
    """Construir arquitectura LSTM simplificada"""
    model = Sequential()

    # Primera capa LSTM
    model.add(LSTM(
        units=self.lstm_units[0],
        return_sequences=True,
        input_shape=input_shape
    ))
    model.add(BatchNormalization())
    model.add(Dropout(self.dropout_rate))

    # Segunda capa LSTM
    if len(self.lstm_units) > 1:
        model.add(LSTM(
            units=self.lstm_units[1],
            return_sequences=True
        ))
        model.add(BatchNormalization())
        model.add(Dropout(self.dropout_rate))

    # Tercera capa LSTM
    if len(self.lstm_units) > 2:
        model.add(LSTM(
            units=self.lstm_units[2],
            return_sequences=False
        ))

```

```

        ))
model.add(Dropout(self.dropout_rate))

# Capas densas para decodificar
model.add(Dense(100, activation='relu'))
model.add(Dropout(self.dropout_rate))
model.add(Dense(50, activation='relu'))

# Capa de salida: todas las variables
model.add(Dense(output_dim))

# Compilar modelo
model.compile(
    optimizer=Adam(learning_rate=self.learning_rate),
    loss='mse',
    metrics=['mae']
)

return model

def predict(self, X, n_periods: int = 240):
    """Realizar predicciones para n periodos futuros"""
    # Escalar los datos de entrada
    scaled_data = self.scaler.fit_transform(X)

    # Obtener última secuencia conocida
    if self.last_sequence is None:
        # Si no hay última secuencia guardada, crear una
        last_sequence = scaled_data[-self.sequence_length:]
        last_sequence = last_sequence.reshape(1,
self.sequence_length, -1)
    else:
        last_sequence = self.last_sequence

    # Realizar predicción recursiva
    predictions_scaled = self._recursive_forecast(last_sequence,
n_periods)

    # Invertir escalado
    predictions =
self.scaler.inverse_transform(predictions_scaled)

    # Crear DataFrame con predicciones
    predictions_df = pd.DataFrame(
        predictions,
        columns=self.feature_names,
        index=pd.date_range(
            start=X.index[-1] + pd.Timedelta(days=1),
            periods=n_periods,

```

```

        freq='D'
    )
)

return predictions_df

def _recursive_forecast(self, last_sequence, n_periods):
    """Predicción recursiva paso a paso"""
    predictions = []
    current_sequence = last_sequence.copy()

    for _ in range(n_periods):
        # Predecir siguiente paso
        next_step = self.model.predict(current_sequence,
verbose=0)
        predictions.append(next_step[0])

        # Actualizar secuencia para siguiente predicción
        current_sequence = np.roll(current_sequence, -1, axis=1)
        current_sequence[0, -1, :] = next_step[0]

    return np.array(predictions)

def get_params(self, deep=True):
    return {
        'sequence_length': self.sequence_length,
        'lstm_units': self.lstm_units,
        'dropout_rate': self.dropout_rate,
        'learning_rate': self.learning_rate,
        'epochs': self.epochs,
        'batch_size': self.batch_size,
        'validation_split': self.validation_split,
        'patience': self.patience
    }

def set_params(self, **parameters):
    for parameter, value in parameters.items():
        setattr(self, parameter, value)
    return self

```

---

## Carga datos sintéticos + reales desde archivo

[ [Volver al índice](#) ]

---

```
def load_and_prepare_data(file_path: str) -> pd.DataFrame:
    """Cargar y preparar los datos temporales"""

    # Cargar datos (ajusta según tu fuente real)
    df = pd.read_csv(file_path, index_col='Fecha', parse_dates=True)

    # Verificar que el índice es temporal y está ordenado
    if not isinstance(df.index, pd.DatetimeIndex):
        df.index = pd.to_datetime(df.index)

    df = df.sort_index()

    # Completar fechas faltantes si es necesario
    full_date_range = pd.date_range(start=df.index.min(),
                                    end=df.index.max(), freq='D')
    df = df.reindex(full_date_range)

    # Interpolación de valores faltantes
    df = df.interpolate(method='time')

    print(f"Datos cargados: {df.shape[0]} observaciones, {df.shape[1]} variables")
    print(f"Rango temporal: {df.index.min()} a {df.index.max()}")

    return df
```

---

## Preprocesador personalizado para aplicar una normalización personalizada por columnas

[\[ Volver al índice \]](#)

---

```
def create_custom_scaler(df: pd.DataFrame) -> ColumnTransformer:
    """
    Crear ColumnTransformer personalizado que devuelve DataFrame
    usando set_output
    """

    # Definir columnas para cada tipo de escalado
```

```

    std_scaler_cols = df.columns.drop(["Velocidad del viento",
    "Presión atmosférica", "Precipitaciones"]).tolist()
    robust_cols = ["Velocidad del viento", "Presión atmosférica",
    "Precipitaciones"]

    # Crear el ColumnTransformer
    preprocessor = ColumnTransformer(
        transformers=[
            ('std_scaler', StandardScaler(), std_scaler_cols),
            ('robust_scaler', RobustScaler(), robust_cols)
        ],
        remainder='passthrough', # Asegurar que todas las columnas se
        mantienen
        verbose_feature_names_out=False # Para mantener nombres
        originales de columnas
    )

    # Configurar para que devuelva DataFrame en lugar de array numpy
    return preprocessor.set_output(transform="pandas")

```

---

## División del conjunto de datos en entrenamiento y test

[ [Volver al índice](#) ]

---

```

def temporal_train_test_split(df: pd.DataFrame, test_size: int) ->
    Tuple[pd.DataFrame, pd.DataFrame]:
    """División temporal de datos"""

    split_point = len(df) - test_size
    train_data = df.iloc[:split_point]
    test_data = df.iloc[split_point:]

    print(f"Train: {train_data.shape[0]} observaciones
    ({train_data.index.min()} a {train_data.index.max()})")
    print(f"Test: {test_data.shape[0]} observaciones
    ({test_data.index.min()} a {test_data.index.max()})")

    return train_data, test_data

```

---

# Creación de pipelines para los diferentes modelos

[ [Volver al índice](#) ]

La implementación previa de *wrappers* para los modelos (SARIMA, SARIMAX, VAR y LSTM) permite adaptarlos a la **API estándar de scikit-learn**, haciendo posible su integración directa dentro de `Pipeline`.

Gracias a estos wrappers, todos los modelos:

- Comparten una **interfaz común (fit / predict)**, independientemente de su naturaleza estadística o neuronal.
- Pueden combinarse fácilmente con etapas de **preprocesado** (escalado, selección de variables, transformación de datos).
- Permiten un **entrenamiento reproducible y modular**, evitando fugas de información entre train y test.
- Facilitan la **comparación homogénea entre modelos** y su extensión a validación cruzada o ajuste de hiperparámetros.

Este diseño hace posible definir pipelines específicos por variable (SARIMA, SARIMAX), multivariantes (VAR) o basados en deep learning (LSTM), manteniendo un flujo de trabajo unificado, limpio y escalable.

```
def create_sarima_PIPELINES(df: pd.DataFrame) -> Dict[str, Pipeline]:  
    """Crear pipelines SARIMA para cada variable"""  
  
    pipelines = {}  
    preprocessor = create_custom_scaler(df)  
  
    for column in df.columns:  
        pipeline = Pipeline([  
            ('scaler', preprocessor),  
            ('sarima', SarimaModel(column=column,  
                                  order=Config.SARIMA_ORDER,  
                                  seasonal_order=Config.SARIMA_SEASONAL_ORDER))  
        ])  
        pipelines[f'sarima_{column}'] = pipeline  
  
    return pipelines  
  
def create_sarimax_PIPELINES(df: pd.DataFrame) -> Dict[str, Pipeline]:  
    """Crear pipelines SARIMAX para cada variable"""  
  
    pipelines = {}  
    all_columns = df.columns.tolist()
```

```

preprocessor = create_custom_scaler(df)

for target_col in df.columns:
    exog_cols = [col for col in all_columns if col != target_col]

    pipeline = Pipeline([
        ('scaler', preprocessor),
        ('sarimax', SarimaxModel(target_col=target_col,
                                  exog_cols=exog_cols,
                                  order=Config.SARIMA_ORDER,
                                  seasonal_order=Config.SARIMA_SEASONAL_ORDER))
    ])
    pipelines[f'sarimax_{target_col}'] = pipeline

return pipelines

def create_var_pipeline(df: pd.DataFrame) -> Pipeline:
    """Crear pipeline VAR para todas las variables"""

    preprocessor = create_custom_scaler(df)

    pipeline = Pipeline([
        ('scaler', preprocessor),
        ('var', VarModel(maxlags=Config.VAR_MAXLAGS))
    ])

    return pipeline

def create_lstm_pipeline(df: pd.DataFrame, sequence_length: int = 365) -> Pipeline:
    """
    Crear pipeline para LSTM multivariante

    Args:
        df: DataFrame con los datos de entrenamiento
        sequence_length: Longitud de la secuencia temporal

    Returns:
        Pipeline configurado para LSTM
    """

    # Crear pipeline
    pipeline = Pipeline([
        ('lstm', LSTMModel(sequence_length=sequence_length))
    ])

    return pipeline

```

---

# Entrenamiento, guardado y evaluación de los modelos

[ [Volver al índice](#) ]

---

```
class ModelTrainer:
    """Clase para entrenar y evaluar modelos"""

    def __init__(self, pipelines: Dict, train_data: pd.DataFrame,
    test_data: pd.DataFrame):
        self.pipelines = pipelines
        self.train_data = train_data
        self.test_data = test_data
        self.trained_models = {}
        self.predictions = {}

    def train_all_models(self):
        """Entrenar todos los modelos"""

        for name, pipeline in self.pipelines.items():
            print(f"Entrenando {name}...")

            try:
                pipeline.fit(self.train_data)
                self.trained_models[name] = pipeline
                print(f"✓ {name} entrenado correctamente")
            except Exception as e:
                print(f"✗ Error entrenando {name}: {str(e)}")

    def generate_predictions(self, horizon: int = None):
        """Generar predicciones para todos los modelos"""

        if horizon is None:
            horizon = len(self.test_data)

        for name, model in self.trained_models.items():
            try:
```

```

        # Generar predicciones
        predictions = model.predict(self.train_data,
n_periods=horizon)
        self.predictions[name] = predictions
        print(f"\u2708 Predicciones generadas para {name}\u2709")

    except Exception as e:
        print(f"\u2708x Error generando predicciones para {name}:
{str(e)}\u2709")

    def save_models(self):
        """Guardar modelos entrenados"""

        for name, model in self.trained_models.items():
            filename = Config.MODEL_PATH / f"{name}_model.pkl"
            joblib.dump(model, filename)
            print(f"\u2708 Modelos {name} guardado en {filename}\u2709")

    def save_predictions(self):
        """Guardar predicciones"""

        for name, pred in self.predictions.items():
            if isinstance(pred, (pd.DataFrame, pd.Series)):
                filename = Config.RESULTS_PATH /
f"{name}_predictions.csv"
                pred.to_csv(filename)
                print(f"\u2708 Predicciones {name} guardadas en
{filename}\u2709")

# Ejecución del proceso de entrenamiento

def main_execution():
    """Función principal de ejecución incluyendo LSTM simplificado"""

    print("== INICIO DEL PROCESO CON LSTM SIMPLIFICADO ==")

    # 1. Cargar datos
    print("\n1. Cargando datos...")
    df = load_and_prepare_data(PHYSIC_DATA_PATH)

    # 2. Dividir datos
    print("\n2. Dividiendo datos...")
    train_df, test_df = temporal_train_test_split(df,
Config.TEST_SIZE)

    # 3. Crear pipelines
    print("\n3. Creando pipelines...")

    # Pipelines tradicionales
    sarima_PIPELINES = create_sarima_PIPELINES(train_df)

```

```

sarimax_PIPELINES = create_sarimax_PIPELINES(train_df)
var_PIPELINE = {'var_multivariate': create_var_PIPELINE(train_df)}

# Pipeline LSTM simplificado
print("\n3.1 Creando pipeline LSTM...")
lstm_PIPELINE = create_lstm_PIPELINE(train_df,
sequence_length=365)

# Combinar todos los pipelines
all_PIPELINES = {
    **sarima_PIPELINES,
    **sarimax_PIPELINES,
    **var_PIPELINE,
    'lstm_multivariate': lstm_PIPELINE
}

print(f"Total de modelos a entrenar: {len(all_PIPELINES)}")

# 4. Entrenar modelos
print("\n4. Entrenando modelos...")
trainer = ModelTrainer(all_PIPELINES, train_df, test_df)
trainer.train_all_models()

# 5. Generar predicciones
print("\n5. Generando predicciones...")
trainer.generate_predictions(Config.PREDICTION_HORIZON)

# 6. Guardar resultados
print("\n6. Guardando modelos y resultados...")
trainer.save_models()
trainer.save_predictions()

print("\n==== PROCESO COMPLETADO ===")

return trainer

trainer = main_execution()

==== INICIO DEL PROCESO CON LSTM SIMPLIFICADO ===

1. Cargando datos...
Datos cargados: 2472 observaciones, 7 variables
Rango temporal: 2019-01-02 00:00:00 a 2025-10-08 00:00:00

2. Dividiendo datos...
Train: 2292 observaciones (2019-01-02 00:00:00 a 2025-04-11 00:00:00)
Test: 180 observaciones (2025-04-12 00:00:00 a 2025-10-08 00:00:00)

3. Creando pipelines...

```

```
3.1 Creando pipeline LSTM...
Total de modelos a entrenar: 16
```

```
4. Entrenando modelos...
```

```
Entrenando sarima_Humedad relativa...
```

```
/Users/rodrigocastroviejoausucua/.pyenv/versions/3.10.12/lib/
python3.10/site-packages/statsmodels/base/model.py:607:
ConvergenceWarning: Maximum Likelihood optimization failed to
converge. Check mle_retvals
    warnings.warn("Maximum Likelihood optimization failed to "
```

```
✓ sarima_Humedad relativa entrenado correctamente
```

```
Entrenando sarima_Temperatura...
```

```
✓ sarima_Temperatura entrenado correctamente
```

```
Entrenando sarima_Velocidad del viento...
```

```
✓ sarima_Velocidad del viento entrenado correctamente
```

```
Entrenando sarima_Presión atmosférica...
```

```
✓ sarima_Presión atmosférica entrenado correctamente
```

```
Entrenando sarima_Radiación solar...
```

```
/Users/rodrigocastroviejoausucua/.pyenv/versions/3.10.12/lib/
python3.10/site-packages/statsmodels/base/model.py:607:
ConvergenceWarning: Maximum Likelihood optimization failed to
converge. Check mle_retvals
    warnings.warn("Maximum Likelihood optimization failed to "
```

```
✓ sarima_Radiación solar entrenado correctamente
```

```
Entrenando sarima_Precipitaciones...
```

```
✓ sarima_Precipitaciones entrenado correctamente
```

```
Entrenando sarima_Humedad relativa mínima...
```

```
✓ sarima_Humedad relativa mínima entrenado correctamente
```

```
Entrenando sarimax_Humedad relativa...
```

```
✓ sarimax_Humedad relativa entrenado correctamente
```

```
Entrenando sarimax_Temperatura...
```

```
✓ sarimax_Temperatura entrenado correctamente
```

```
Entrenando sarimax_Velocidad del viento...
```

```
/Users/rodrigocastroviejoausucua/.pyenv/versions/3.10.12/lib/
python3.10/site-packages/statsmodels/base/model.py:607:
ConvergenceWarning: Maximum Likelihood optimization failed to
converge. Check mle_retvals
    warnings.warn("Maximum Likelihood optimization failed to "
```

```
✓ sarimax_Velocidad del viento entrenado correctamente
```

```
Entrenando sarimax_Presión atmosférica...
```

```
/Users/rodrigocastroviejoausucua/.pyenv/versions/3.10.12/lib/
python3.10/site-packages/statsmodels/base/model.py:607:
ConvergenceWarning: Maximum Likelihood optimization failed to
```

```
converge. Check mle_retrvals
    warnings.warn("Maximum Likelihood optimization failed to "

✓ sarimax_Presión atmosférica entrenado correctamente
Entrenando sarimax_Radiación solar...
✓ sarimax_Radiación solar entrenado correctamente
Entrenando sarimax_Precipitaciones...

/Users/rodrigocastroviejoausucua/.pyenv/versions/3.10.12/lib/
python3.10/site-packages/statsmodels/base/model.py:607:
ConvergenceWarning: Maximum Likelihood optimization failed to
converge. Check mle_retrvals
    warnings.warn("Maximum Likelihood optimization failed to "

✓ sarimax_Precipitaciones entrenado correctamente
Entrenando sarimax_Humedad relativa mínima...
✓ sarimax_Humedad relativa mínima entrenado correctamente
Entrenando var_multivariate...
✓ var_multivariate entrenado correctamente
Entrenando lstm_multivariate...

/Users/rodrigocastroviejoausucua/.pyenv/versions/3.10.12/lib/
python3.10/site-packages/keras/src/layers/rnn/rnn.py:199: UserWarning:
Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
    super().__init__(**kwargs)

✓ lstm_multivariate entrenado correctamente

5. Generando predicciones...
✓ Predicciones generadas para sarima_Humedad relativa
✓ Predicciones generadas para sarima_Temperatura

/Users/rodrigocastroviejoausucua/.pyenv/versions/3.10.12/lib/
python3.10/site-packages/sklearn/pipeline.py:61: FutureWarning: This
Pipeline instance is not fitted yet. Call 'fit' with appropriate
arguments before using other methods such as transform, predict, etc.
This will raise an error in 1.8 instead of the current warning.
    warnings.warn(
/Users/rodrigocastroviejoausucua/.pyenv/versions/3.10.12/lib/python3.1
0/site-packages/sklearn/pipeline.py:61: FutureWarning: This Pipeline
instance is not fitted yet. Call 'fit' with appropriate arguments
before using other methods such as transform, predict, etc. This will
raise an error in 1.8 instead of the current warning.
    warnings.warn(
/Users/rodrigocastroviejoausucua/.pyenv/versions/3.10.12/lib/python3.1
0/site-packages/sklearn/pipeline.py:61: FutureWarning: This Pipeline
instance is not fitted yet. Call 'fit' with appropriate arguments
before using other methods such as transform, predict, etc. This will
raise an error in 1.8 instead of the current warning.
```

```
warnings.warn()
/Users/rodrigocastroviejoausucua/.pyenv/versions/3.10.12/lib/python3.1
0/site-packages/sklearn/pipeline.py:61: FutureWarning: This Pipeline
instance is not fitted yet. Call 'fit' with appropriate arguments
before using other methods such as transform, predict, etc. This will
raise an error in 1.8 instead of the current warning.
    warnings.warn()
/Users/rodrigocastroviejoausucua/.pyenv/versions/3.10.12/lib/python3.1
0/site-packages/sklearn/pipeline.py:61: FutureWarning: This Pipeline
instance is not fitted yet. Call 'fit' with appropriate arguments
before using other methods such as transform, predict, etc. This will
raise an error in 1.8 instead of the current warning.
    warnings.warn()
✓ Predicciones generadas para sarima_Velocidad del viento
✓ Predicciones generadas para sarima_Presión atmosférica
✓ Predicciones generadas para sarima_Radiación solar
✓ Predicciones generadas para sarima_Precipitaciones
✓ Predicciones generadas para sarima_Humedad relativa mínima

/Users/rodrigocastroviejoausucua/.pyenv/versions/3.10.12/lib/
python3.10/site-packages/sklearn/pipeline.py:61: FutureWarning: This
Pipeline instance is not fitted yet. Call 'fit' with appropriate
arguments before using other methods such as transform, predict, etc.
This will raise an error in 1.8 instead of the current warning.
    warnings.warn()
/Users/rodrigocastroviejoausucua/.pyenv/versions/3.10.12/lib/python3.1
0/site-packages/sklearn/pipeline.py:61: FutureWarning: This Pipeline
instance is not fitted yet. Call 'fit' with appropriate arguments
before using other methods such as transform, predict, etc. This will
raise an error in 1.8 instead of the current warning.
    warnings.warn()
/Users/rodrigocastroviejoausucua/.pyenv/versions/3.10.12/lib/python3.1
0/site-packages/sklearn/pipeline.py:61: FutureWarning: This Pipeline
instance is not fitted yet. Call 'fit' with appropriate arguments
before using other methods such as transform, predict, etc. This will
raise an error in 1.8 instead of the current warning.
    warnings.warn()
/Users/rodrigocastroviejoausucua/.pyenv/versions/3.10.12/lib/python3.1
0/site-packages/sklearn/pipeline.py:61: FutureWarning: This Pipeline
instance is not fitted yet. Call 'fit' with appropriate arguments
before using other methods such as transform, predict, etc. This will
raise an error in 1.8 instead of the current warning.
    warnings.warn()
/Users/rodrigocastroviejoausucua/.pyenv/versions/3.10.12/lib/python3.1
0/site-packages/sklearn/pipeline.py:61: FutureWarning: This Pipeline
instance is not fitted yet. Call 'fit' with appropriate arguments
before using other methods such as transform, predict, etc. This will
raise an error in 1.8 instead of the current warning.
    warnings.warn()
```

```
✓ Predicciones generadas para sarimax_Humedad relativa
✓ Predicciones generadas para sarimax_Temperatura
✓ Predicciones generadas para sarimax_Velocidad del viento
✓ Predicciones generadas para sarimax_Presión atmosférica
✓ Predicciones generadas para sarimax_Radiación solar

/Users/rodrigocastroviejoausucua/.pyenv/versions/3.10.12/lib/
python3.10/site-packages/sklearn/pipeline.py:61: FutureWarning: This
Pipeline instance is not fitted yet. Call 'fit' with appropriate
arguments before using other methods such as transform, predict, etc.
This will raise an error in 1.8 instead of the current warning.
    warnings.warn(
/Users/rodrigocastroviejoausucua/.pyenv/versions/3.10.12/lib/python3.1
0/site-packages/sklearn/pipeline.py:61: FutureWarning: This Pipeline
instance is not fitted yet. Call 'fit' with appropriate arguments
before using other methods such as transform, predict, etc. This will
raise an error in 1.8 instead of the current warning.
    warnings.warn(
/Users/rodrigocastroviejoausucua/.pyenv/versions/3.10.12/lib/python3.1
0/site-packages/sklearn/pipeline.py:61: FutureWarning: This Pipeline
instance is not fitted yet. Call 'fit' with appropriate arguments
before using other methods such as transform, predict, etc. This will
raise an error in 1.8 instead of the current warning.
    warnings.warn(
/Users/rodrigocastroviejoausucua/.pyenv/versions/3.10.12/lib/python3.1
0/site-packages/sklearn/pipeline.py:61: FutureWarning: This Pipeline
instance is not fitted yet. Call 'fit' with appropriate arguments
before using other methods such as transform, predict, etc. This will
raise an error in 1.8 instead of the current warning.
    warnings.warn(
/Users/rodrigocastroviejoausucua/.pyenv/versions/3.10.12/lib/python3.1
0/site-packages/sklearn/pipeline.py:61: FutureWarning: This Pipeline
instance is not fitted yet. Call 'fit' with appropriate arguments
before using other methods such as transform, predict, etc. This will
raise an error in 1.8 instead of the current warning.
    warnings.warn(
✓ Predicciones generadas para sarimax_Precipitaciones
✓ Predicciones generadas para sarimax_Humedad relativa mínima
✓ Predicciones generadas para var_multivariate

/Users/rodrigocastroviejoausucua/.pyenv/versions/3.10.12/lib/
python3.10/site-packages/sklearn/pipeline.py:61: FutureWarning: This
Pipeline instance is not fitted yet. Call 'fit' with appropriate
arguments before using other methods such as transform, predict, etc.
This will raise an error in 1.8 instead of the current warning.
    warnings.warn(
✓ Predicciones generadas para lstm_multivariate
```

## 6. Guardando modelos y resultados...

```
✓ Modelo sarima_Humedad relativa guardado en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/models/sarima_Humedad
relativa_model.pkl
✓ Modelo sarima_Temperatura guardado en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/models/
sarima_Temperatura_model.pkl
✓ Modelo sarima_Velocidad del viento guardado en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/models/sarima_Velocidad
del viento_model.pkl
✓ Modelo sarima_Presión atmosférica guardado en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/models/sarima_Presión
atmosférica_model.pkl
✓ Modelo sarima_Radiación solar guardado en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/models/sarima_Radiación
solar_model.pkl
✓ Modelo sarima_Precipitaciones guardado en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/models/
sarima_Precipitaciones_model.pkl
✓ Modelo sarima_Humedad relativa mínima guardado en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/models/sarima_Humedad
relativa mínima_model.pkl
✓ Modelo sarimax_Humedad relativa guardado en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/models/sarimax_Humedad
relativa_model.pkl
✓ Modelo sarimax_Temperatura guardado en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/models/
sarimax_Temperatura_model.pkl
✓ Modelo sarimax_Velocidad del viento guardado en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/models/
sarimax_Velocidad del viento_model.pkl
✓ Modelo sarimax_Presión atmosférica guardado en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/models/sarimax_Presión
atmosférica_model.pkl
✓ Modelo sarimax_Radiación solar guardado en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/models/
sarimax_Radiación solar_model.pkl
```

```
✓ Modelo sarimax_Precipitaciones guardado en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/models/
sarimax_Precipitaciones_model.pkl
✓ Modelo sarimax_Humedad relativa mínima guardado en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/models/sarimax_Humedad
relativa mínima_model.pkl
✓ Modelo var_multivariate guardado en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/models/
var_multivariate_model.pkl
✓ Modelo lstm_multivariate guardado en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/models/
lstm_multivariate_model.pkl
✓ Predicciones sarima_Humedad relativa guardadas en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/results/sarima_Humedad
relativa_predictions.csv
✓ Predicciones sarima_Temperatura guardadas en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/results/
sarima_Temperatura_predictions.csv
✓ Predicciones sarima_Velocidad del viento guardadas en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/results/
sarima_Velocidad del viento_predictions.csv
✓ Predicciones sarima_Presión atmosférica guardadas en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/results/sarima_Presión
atmosférica_predictions.csv
✓ Predicciones sarima_Radiación solar guardadas en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/results/
sarima_Radiación solar_predictions.csv
✓ Predicciones sarima_Precipitaciones guardadas en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/results/
sarima_Precipitaciones_predictions.csv
✓ Predicciones sarima_Humedad relativa mínima guardadas en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/results/sarima_Humedad
relativa_mínima_predictions.csv
✓ Predicciones sarimax_Humedad relativa guardadas en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/results/sarimax_Humedad
relativa_predictions.csv
✓ Predicciones sarimax_Temperatura guardadas en
```

```

/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/results/
sarimax_Temperatura_predictions.csv
✓ Predicciones sarimax_Velocidad del viento guardadas en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/results/
sarimax_Velocidad del viento_predictions.csv
✓ Predicciones sarimax_Presión atmosférica guardadas en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/results/sarimax_Presión
atmosférica_predictions.csv
✓ Predicciones sarimax_Radiación solar guardadas en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/results/
sarimax_Radiación solar_predictions.csv
✓ Predicciones sarimax_Precipitaciones guardadas en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/results/
sarimax_Precipitaciones_predictions.csv
✓ Predicciones sarimax_Humedad relativa mínima guardadas en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/results/sarimax_Humedad
relativa mínima_predictions.csv
✓ Predicciones var_multivariate guardadas en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/results/
var_multivariate_predictions.csv
✓ Predicciones lstm_multivariate guardadas en
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/results/
lstm_multivariate_predictions.csv

==== PROCESO COMPLETADO ===

def load_saved_models():
    """Cargar modelos guardados para hacer predicciones sin re-
    entrenar"""

    models = {}
    model_files = Config.MODEL_PATH.glob("*.pkl")

    for file_path in model_files:
        # Obtener nombre del archivo correctamente
        model_name = file_path.stem.replace('_model', '')

        try:
            models[model_name] = joblib.load(file_path)
            print(f"/ Modelo {model_name} cargado correctamente")
        except Exception as e:
            print(f"\x Error cargando {model_name}: {str(e)}")

```

```

    return models

# Carga y uso de resultados guardados
def load_predictions(model_name):
    """
        Carga las predicciones de un modelo específico y las combina en un
        DataFrame.

    Args:
        model_name (str): Nombre del modelo ('sarima', 'sarimax',
        'var' o 'lstm')

    Returns:
        pd.DataFrame: DataFrame con todas las predicciones del modelo
    """
    if model_name.lower() in ('var', 'lstm'):
        # Para VAR y LSTM, solo hay un archivo
        filename = Config.RESULTS_PATH /
f"{model_name.lower()}_multivariate_predictions.csv"
        if not filename.exists():
            raise FileNotFoundError(f"No se encontró el archivo:
{filename}")

        df = pd.read_csv(filename, index_col=0)
        # Convertir de nuevo el index (Fecha) a datetime (era str)
        df.index = pd.to_datetime(df.index)
        print(f"\u2708 Cargadas predicciones VAR de {filename}")
        return df

    else:
        # Para SARIMA y SARIMAX, cargar múltiples archivos
        model_name_lower = model_name.lower()
        pattern = f"{model_name_lower}_*predictions.csv"

        # Encontrar todos los archivos que coincidan con el patrón
        files = list(Config.RESULTS_PATH.glob(pattern))

        if not files:
            raise FileNotFoundError(f"No se encontraron archivos con
el patrón: {pattern}")

        # Diccionario para almacenar DataFrames temporales
        dfs = {}

        for file in files:
            # Extraer el nombre del atributo del nombre del archivo
            # Ejemplo: "resultssarima_Temperatura_predictions.csv" ->

```

```

"Temperatura"
    attr_name = file.stem.replace(f"{{model_name_lower}}_",
"").replace("_predictions", "")
    # Cargar el archivo CSV
    temp_df = pd.read_csv(file, index_col=0)

    # Si es una Serie (una sola columna), renombrarla con el
    # nombre del atributo
    if temp_df.shape[1] == 1:
        temp_df.columns = [attr_name]

    dfs[attr_name] = temp_df

    # Combinar todos los DataFrames por índice (fechas)
    combined_df = pd.concat(dfs.values(), axis=1)

    # Convertir de nuevo el index (Fecha) a datetime (era str)
    combined_df.index = pd.to_datetime(combined_df.index)

    print(f"✓ Cargadas {len(files)} predicciones de
{model_name.upper()}")
    print(f"  Atributos: {list(dfs.keys())}")

    return combined_df

def make_future_predictions(loader_models: Dict, last_available_data:
pd.DataFrame,
                           horizon: int = 240):
    """Hacer predicciones futuras con modelos cargados"""

    predictions = {}

    for name, model in loader_models.items():
        try:
            pred = model.predict(last_available_data,
n_periods=horizon)
            predictions[name] = pred
            print(f"✓ Predicción futura generada para {name}")
        except Exception as e:
            print(f"✗ Error en predicción para {name}: {str(e)}")

    return predictions

# Ejemplo de uso:
#loaded_models = load_saved_models()

#future_predictions = make_future_predictions(loader_models,
#load_and_prepare_data(PHYSIC_DATA_PATH), horizon=240)

```

## Visualización de resultados

[\[ Volver al índice \]](#)

```
def plot_predictions(original_data: pd.DataFrame, predictions: Dict,
                     variables: List[str] = None, n_last_points: int =
365):
    """Visualizar predicciones vs datos reales"""

    if variables is None:
        variables = original_data.columns

    fig, axes = plt.subplots(len(variables), 1, figsize=(15,
5*len(variables)))
    if len(variables) == 1:
        axes = [axes]

    for i, var in enumerate(variables):
        # Datos originales (últimos n puntos)
        recent_data = original_data[var].iloc[-n_last_points:]

        axes[i].plot(recent_data.index, recent_data.values,
                     label='Real', linewidth=2, color='blue')

        # Predicciones de cada modelo
        colors = ['red', 'green', 'orange', 'purple']
        for j, (model_name, pred_dict) in
enumerate(predictions.items()):
            if var in pred_dict.columns if isinstance(pred_dict,
pd.DataFrame) else model_name.endswith(var):
                pred_series = pred_dict[var] if isinstance(pred_dict,
pd.DataFrame) else pred_dict

                # Asegurar que las fechas coinciden
                if isinstance(pred_series, pd.Series):
                    start_date = recent_data.index[-1] +
pd.Timedelta(days=1)
                    pred_dates = pd.date_range(start=start_date,
periods=len(pred_series), freq='D')

                    axes[i].plot(pred_dates, pred_series.values,
                     label=model_name, linestyle='--')
```

```

color=colors[j % len(colors)],
alpha=0.8)

    axes[i].set_title(f'Predicciones para {var}')
    axes[i].set_ylabel(var)
    axes[i].legend()
    axes[i].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

lstm_predictions = load_predictions("lstm")
sarima_predictions = load_predictions("sarima")
sarimax_predictions = load_predictions("sarimax")
var_predictions = load_predictions("var")

# Ejemplo de uso:
print("PREDICCIONES MODELO LSTM: ")
plot_predictions(load_and_prepare_data(PHYSIC_DATA_PATH),
lstm_predictions, n_last_points=180)

print("PREDICCIONES MODELO SARIMA: ")
plot_predictions(load_and_prepare_data(PHYSIC_DATA_PATH),
sarima_predictions, n_last_points=180)

print("PREDICCIONES MODELO SARIMAX: ")
plot_predictions(load_and_prepare_data(PHYSIC_DATA_PATH),
sarimax_predictions, n_last_points=180)

print("PREDICCIONES MODELO VAR: ")
plot_predictions(load_and_prepare_data(PHYSIC_DATA_PATH),
var_predictions, n_last_points=180)

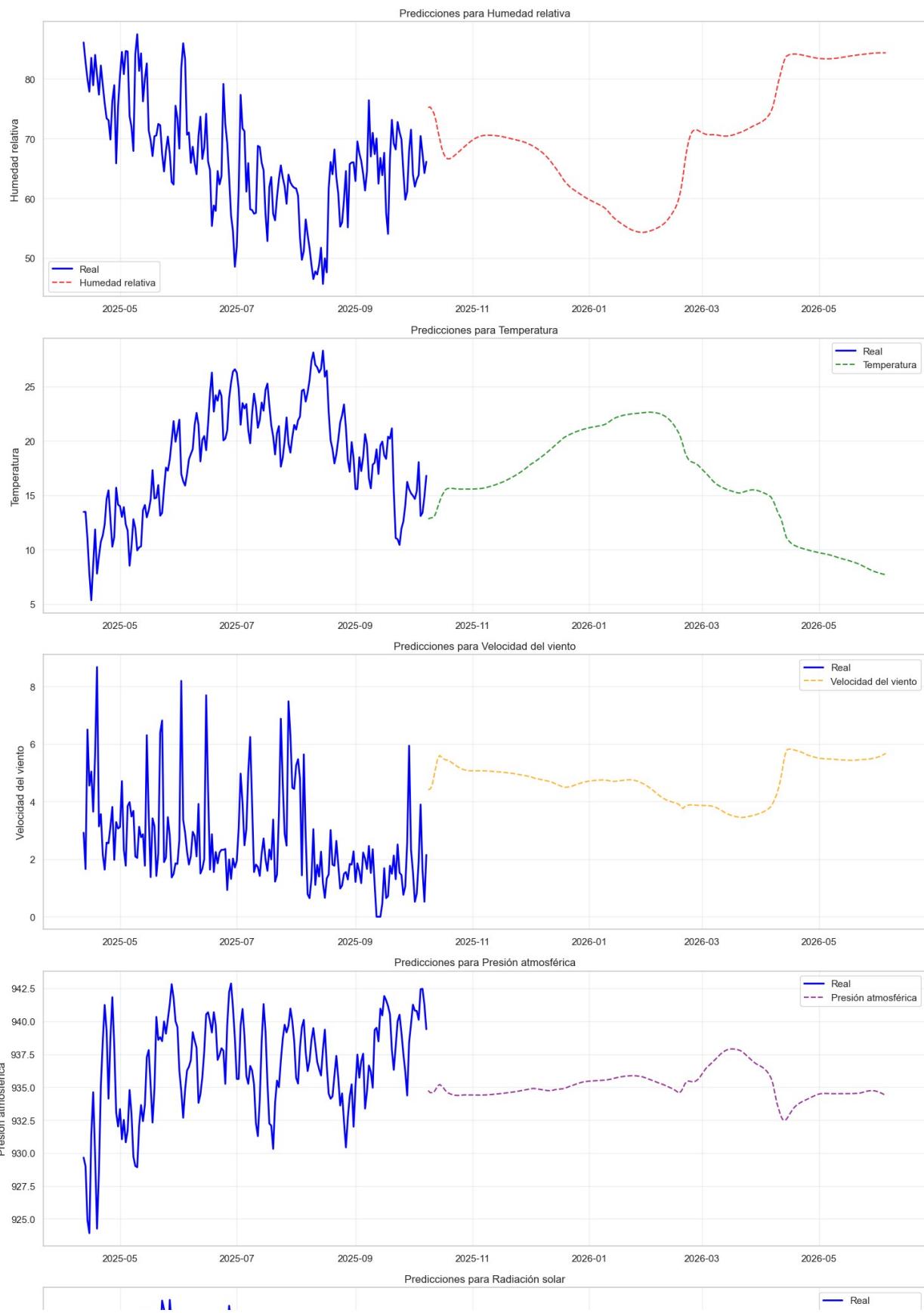
✓ Cargadas predicciones VAR de
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/results/
lstm_multivariate_predictions.csv
✓ Cargadas 7 predicciones de SARIMA
Atributos: ['Humedad relativa', 'Velocidad del viento', 'Humedad
relativa mínima', 'Precipitaciones', 'Temperatura', 'Radiación solar',
'Presión atmosférica']
✓ Cargadas 7 predicciones de SARIMAX
Atributos: ['Radiación solar', 'Humedad relativa mínima',
'Precipitaciones', 'Temperatura', 'Humedad relativa', 'Presión
atmosférica', 'Velocidad del viento']
✓ Cargadas predicciones VAR de
/Users/rodrigocastroviejoausucua/Desktop/Material Ubu/Edge-Computing-
para-Riego-Deficitario-en-Almendros/model/data/results/
var_multivariate_predictions.csv

```

PREDICCIONES MODELO LSTM:

Datos cargados: 2472 observaciones, 7 variables

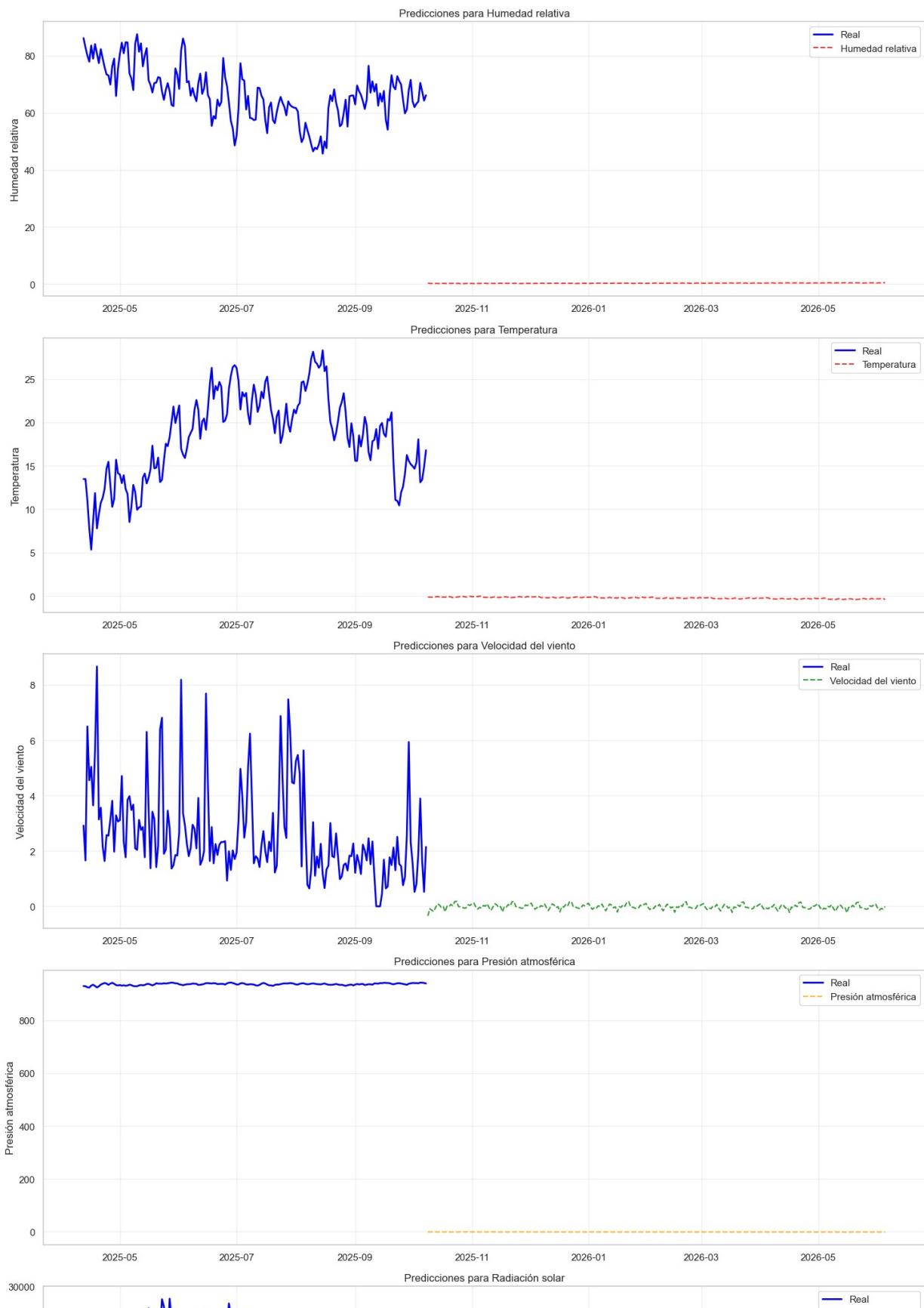
Rango temporal: 2019-01-02 00:00:00 a 2025-10-08 00:00:00



PREDICCIONES MODELO SARIMA:

Datos cargados: 2472 observaciones, 7 variables

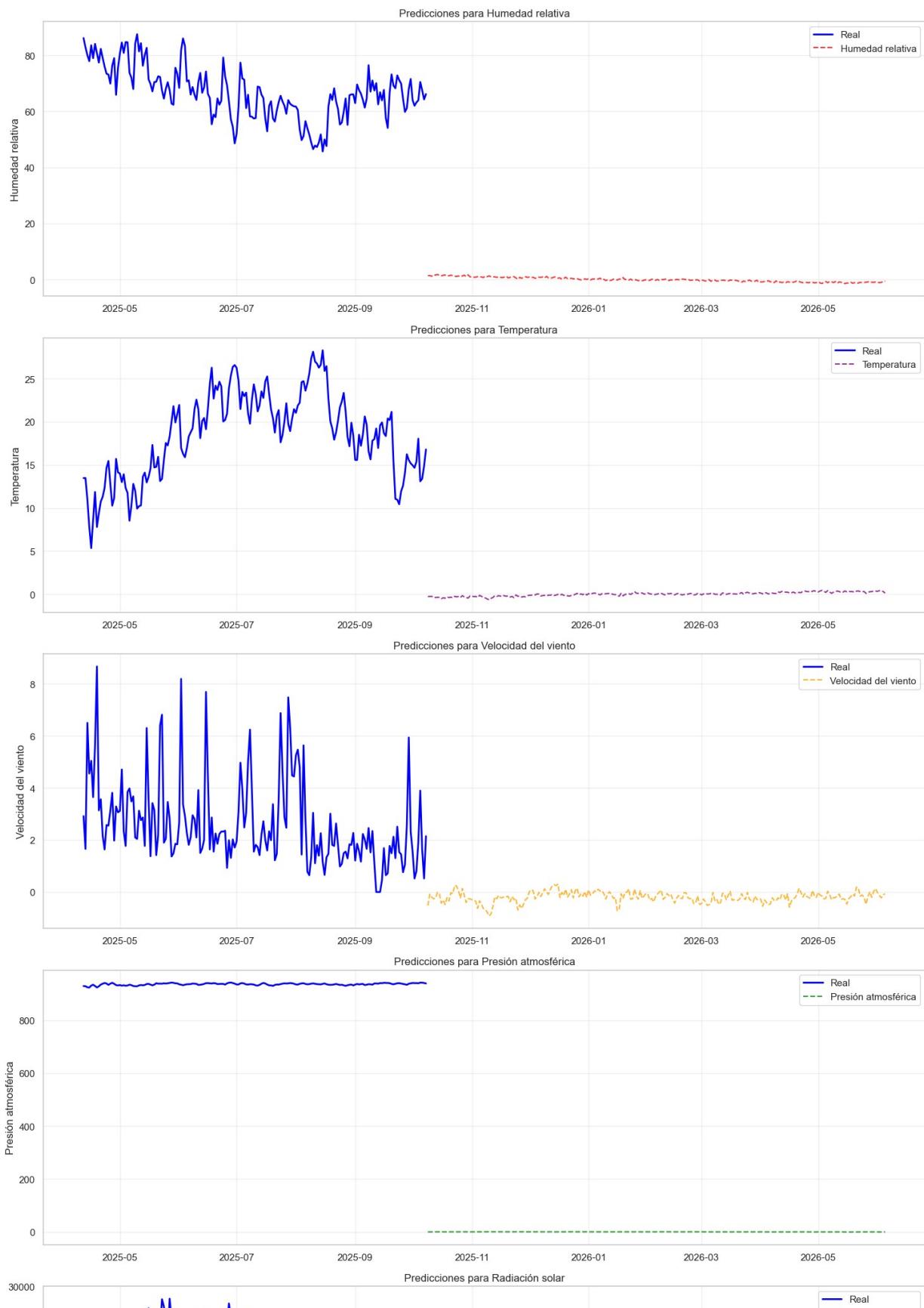
Rango temporal: 2019-01-02 00:00:00 a 2025-10-08 00:00:00



PREDICCIONES MODELO SARIMAX:

Datos cargados: 2472 observaciones, 7 variables

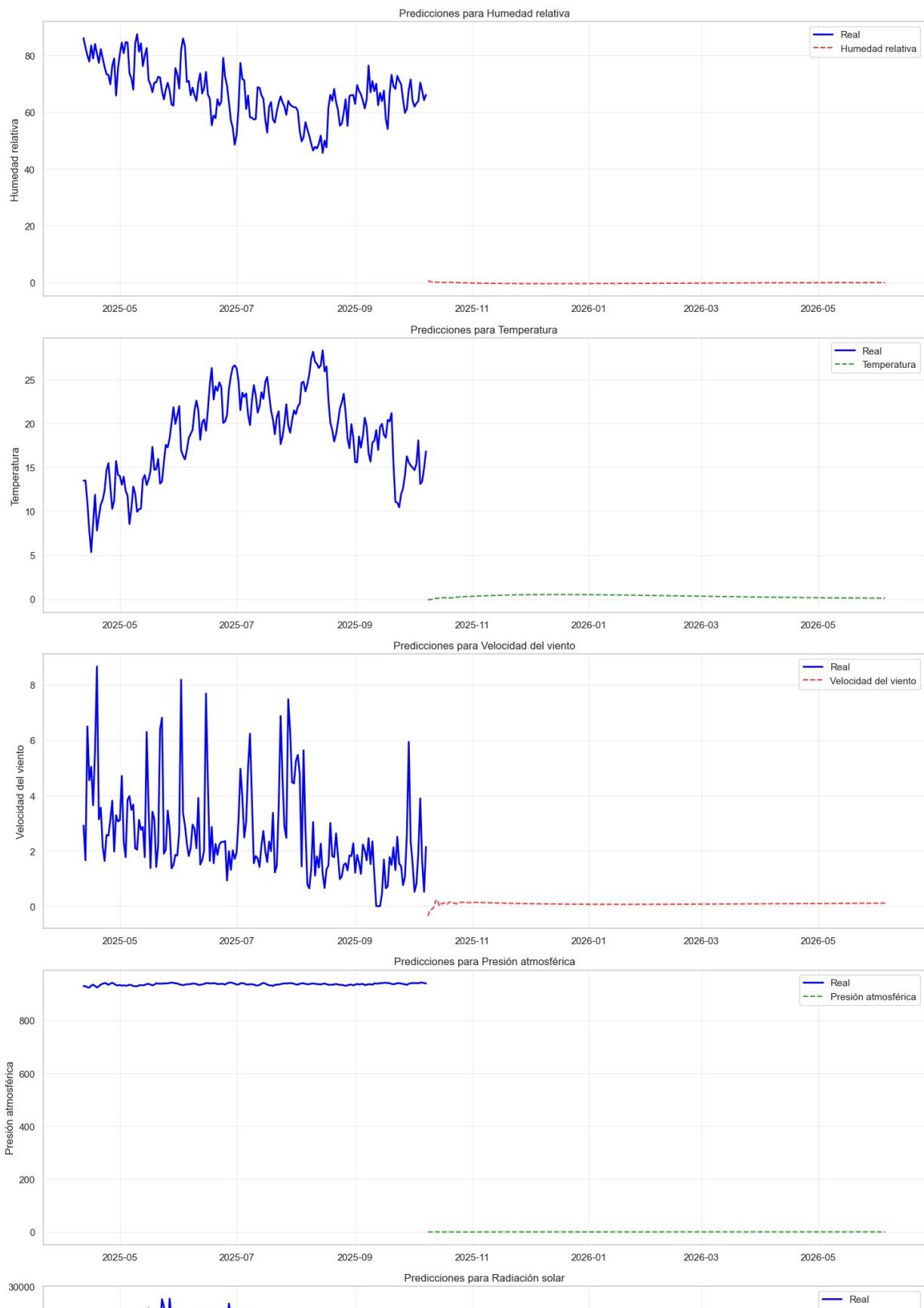
Rango temporal: 2019-01-02 00:00:00 a 2025-10-08 00:00:00



PREDICCIONES MODELO VAR:

Datos cargados: 2472 observaciones, 7 variables

Rango temporal: 2019-01-02 00:00:00 a 2025-10-08 00:00:00



```
MODELO_ELEGIDO = lstm_predictions
```

---

## Cálculo Necesidad de riego y aplicación RDC

[ [Volver al índice](#) ]

---

### Cálculo de la ET0 (Evotranspiración de referencia)

[ [Volver al índice](#) ]

---

```
def calculate_et0_fao_penman_monteith(df):
    """
    Calcula la Evapotranspiración de Referencia (ET0) usando la
    ecuación FAO Penman-Monteith
    """

    # Obtener datos del dataframe
    T = df['Temperatura'] # Temperatura media del aire [°C]
    RH = df['Humedad relativa'] # Humedad relativa [%]
    u2 = df['Velocidad del viento'] # Velocidad del viento a 2m [m/s]
    P = df['Presión atmosférica'] * 0.1 # Convertir hPa a kPa
    Rs = df['Radiación solar'] / 1e6 # Convertir J/m²/día a MJ/m²/día

    # 1. PRESIÓN DE VAPOR DE SATURACIÓN (es)
    # Fórmula: es = 0.6108 * exp(17.27 * T / (T + 237.3))
    es = 0.6108 * np.exp(17.27 * T / (T + 237.3))

    # 2. PRESIÓN DE VAPOR ACTUAL (ea)
    # Fórmula: ea = (RH/100) * es
    ea = es * RH / 100
```

```

# 3. PENDIENTE DE LA CURVA DE PRESIÓN DE VAPOR ( $\Delta$ )
# Fórmula:  $\Delta = 4098 * es / (T + 237.3)^2$ 
delta = 4098 * es / (T + 237.3)**2

# 4. CONSTANTE PSICROMÉTRICA ( $\gamma$ )
# Fórmula:  $\gamma = 0.665 \times 10^{-3} * P$ 
gamma = 0.665 * 1e-3 * P

# 5. RADIACIÓN NETA ( $Rn$ ) - Simplificada
# Fórmula:  $Rn = 0.77 * Rs$  (asumiendo albedo 0.23)
Rn = (1 - 0.23) * Rs

# 6. FLUJO DE CALOR DEL SUELO ( $G$ ) - Cero para periodos diarios
G = 0

# 7. ECUACIÓN FAO PENMAN-MONTEITH COMPLETA
# Fórmula:
#  $ET0 = [0.408 * \Delta * (Rn - G) + \gamma * (900/(T + 273)) * u2 * (es - ea)] / [\Delta + \gamma * (1 + 0.34 * u2)]$ 

numerador = (0.408 * delta * (Rn - G) +
             gamma * (900 / (T + 273)) * u2 * (es - ea))

denominador = delta + gamma * (1 + 0.34 * u2)

et0 = numerador / denominador

return et0

# Aplicar al dataframe
MODELO_ELEGIDO['ET0'] =
calculate_et0_fao_penman_monteith(MODELO_ELEGIDO)

# Mostrar resultados
print("Resultados ET0:")
display(MODELO_ELEGIDO)

```

Resultados ET0:

	Humedad relativa	Temperatura	Velocidad del viento	\
2025-04-12	75.225680	12.824626	4.414980	
2025-04-13	75.295880	12.897439	4.428036	
2025-04-14	74.808580	12.939791	4.555692	
2025-04-15	74.216220	12.989694	4.880391	
2025-04-16	73.022860	13.382570	5.205284	
...	...	...	...	...
2025-12-03	84.357510	7.870454	5.543632	
2025-12-04	84.361390	7.819257	5.568295	
2025-12-05	84.361410	7.770551	5.597845	

2025-12-06	84.361565	7.726124	5.632737
2025-12-07	84.354004	7.685183	5.669702
	Presión atmosférica	Radiación solar	Precipitaciones \
2025-04-12	934.74460	16281.0080	0.834385
2025-04-13	934.58340	17165.5450	0.908292
2025-04-14	934.58570	18299.7150	0.894316
2025-04-15	934.62720	19228.1780	0.817538
2025-04-16	934.85800	20411.6200	0.720192
...	...	...	...
2025-12-03	934.64514	5192.1370	0.791780
2025-12-04	934.58830	5175.8250	0.800864
2025-12-05	934.52155	5165.1826	0.809427
2025-12-06	934.44630	5160.2974	0.817715
2025-12-07	934.36490	5159.9463	0.825855
	Humedad relativa mínima	ET0	
2025-04-12	45.845673	1.257589	
2025-04-13	46.221250	1.260038	
2025-04-14	45.515476	1.310253	
2025-04-15	44.447453	1.401913	
2025-04-16	42.892647	1.549377	
...	...	...	
2025-12-03	66.213880	0.729799	
2025-12-04	66.233470	0.729562	
2025-12-05	66.240960	0.729911	
2025-12-06	66.239030	0.730748	
2025-12-07	66.221730	0.732183	

[240 rows x 8 columns]

## Cálculo de la Curva Kc para Almendro Penta (FAO Papel 56)

[ [Volver al índice](#) ]

### 1. Identificar las Etapas de Crecimiento y sus Duraciones

Divisiones del ciclo del cultivo:

- **Inicial (Lini):** Brotación → 10% cobertura del suelo
- **Desarrollo (Ldev):** Hasta cobertura máxima (70-80%)
- **Mediados (Lmid):** Máxima cobertura y actividad fisiológica (floración → llenado de frutos)
- **Final (Llate):** Inicio maduración → cosecha/caída de hojas

## Duraciones típicas (Tabla 11 - Deciduous Orchard):

Etapa	Duración (días)
Inicial	20-30
Desarrollo	70-90
Mediados	90-130
Final	30-60

**Nota:** Ajustar según:

- Localización geográfica y clima
- Observaciones locales de la variedad Penta
- Fechas de brotación y cosecha

## 2. Seleccionar Coeficientes de Cultivo de Referencia

### Valores base (Tabla 12 - Almonds, no ground cover):

- $Kc_{ini} = 0.40$
- $Kc_{mid} = 0.90$
- $Kc_{end} = 0.65$

*Condiciones estándar: clima subhúmedo, viento moderado*

## 3. Ajustar Valores de Kc a Condiciones Locales

### Ajuste de $Kc_{ini}$ :

- **Factores:** Frecuencia de mojado (lluvia/riego) y tipo de suelo
- **Herramientas:** Figuras 29 y 30
  - Intervalo entre riegos/lluvias
  - Textura del suelo (Fig. 30a: suelos gruesos, Fig. 30b: suelos finos/medios)
- **Riego parcial (ej. goteo) - Ecuación 60:**

$$kc_{ini} = fw \times kc_{ini}(\text{Fig})$$

donde  $fw$  = fracción de suelo mojado

### Ajuste de $Kc_{mid}$ y $Kc_{end}$ :

- **Ecuaciones 62 y 65:**

$$Kc_{aj} = Kc(\text{Tab}) + [0.04 \times (u_2 - 2) - 0.004 \times (RH_{min} - 45)] \times (h/3)^{0.3}$$

### Variables:

- $u_2$  = velocidad del viento [m/s]
- $RH_{min}$  = humedad relativa mínima [%]
- $h$  = altura del cultivo [m]

## 4. Construir la Curva de Kc

Con valores ajustados y duraciones:

1. **Etapa inicial:** Línea horizontal en  $Kc_{ini}$
2. **Etapa desarrollo:** Línea diagonal de  $Kc_{ini} \rightarrow Kc_{mid}$
3. **Etapa mediados:** Línea horizontal en  $Kc_{mid}$
4. **Etapa final:** Línea diagonal de  $Kc_{mid} \rightarrow Kc_{end}$

**Resultado:** Curva lineal por segmentos que representa la evolución temporal del Kc

## Duración Estimada de las Etapas de Crecimiento del Almendro Penta

### Resumen de Etapas

Etapa de Crecimiento	Duración Estimada	Explicación
<b>Inicial</b>	15 días	- Desde brotación (28 feb, día 59) hasta floración (15 mar, día 74)- Brotación ocurre 15 días antes de la floración
<b>Desarrollo</b>	50 días	- Desde floración hasta cuajado de frutos y crecimiento vegetativo completo- Penta es de vigor bajo pero de entrada en producción precoz
<b>Mediados</b>	98 días	- Desde fin del desarrollo (4 may) hasta inicio de maduración (10 ago)- Maduración temprana característica de la variedad
<b>Final</b>	31 días	- Desde inicio de maduración (10 ago, día 222) hasta cosecha (10 sep, día 253)

# Cálculo Detallado

## Fechas Clave

- **Floración:** 15 de marzo (día 74 del año)
- **Maduración:** 10 de septiembre (día 253) - 10 días antes que 'Ferragnès'
- **Brotación:** 28 de febrero (día 59) - 15 días antes de la floración
- **Cosecha:** 10 de septiembre (día 253)

## Período Total

**Desde brotación hasta cosecha:** 194 días (253 - 59 = 194)

## Distribución Temporal

1. **Inicial:** día 59 → 74 (15 días)
2. **Desarrollo:** día 74 → 124 (50 días, ~4 de mayo)
3. **Mediados:** día 124 → 222 (98 días, ~10 de agosto)
4. **Final:** día 222 → 253 (31 días)

## Factores Considerados

- **Vigor bajo y porte semi-aberto:** Puede alargar ligeramente la etapa de desarrollo
- **Precocidad productiva:** Sugiere crecimiento eficiente
- **Floración extra-tardía y maduración temprana:** Indican ciclo reproductivo corto
- **Autofecundación:** Asegura cuajada uniforme, apoyando transición rápida entre etapas

```
etapas_almendro = {
    "inicial": {
        "inicio": "3/1/2025",
        "fin": "3/15/2025",
        "kc_tabla": 0.4,
        "h": 5
    },
    "desarrollo": {
        "inicio": "3/16/2025",
        "fin": "5/4/2025",
        "kc_tabla": np.nan,
        "h": 5
    },
    "medio": {
        "inicio": "5/5/2025",
        "fin": "8/10/2025",
        "kc_tabla": 0.9,
        "h": 5
    },
    "final": {
        "inicio": "8/11/2025",
        "fin": "9/10/2025",
        "kc_tabla": 0.65,
        "h": 5
    }
}
```

```
        "h": 5
    }
}
```

---

## Cálculo de Kc Inicial para Almendro Penta - Etapa Inicial

[ [Volver al índice](#) ]

---

### Paso 1: Calcular el Intervalo Promedio entre Eventos de Mojado

Eventos a considerar:

- Riegos y lluvias significativas (>3 mm por evento) durante la etapa inicial

Fórmula de cálculo:

Intervalo\_promedio = Duración\_etapa\_inicial / Número\_eventos\_mojado

Ejemplo:

- Duración etapa inicial: 15 días
- Eventos de riego: 5
- **Intervalo promedio = 15 / 5 = 3 días**

```
def calculo_intervalo_promedio_eventos_mojado(df, fecha_inicio,
fecha_fin):

    # Filtrar por rango de fechas y precipitación > 3 mm
    dias_lluvia = df[
        (df.index >= fecha_inicio) &
        (df.index <= fecha_fin) &
        (df['Precipitaciones'] > 3)
    ]

    # Contar los días
    cantidad_dias = len(dias_lluvia)

    return cantidad_dias
```

## Paso 2: Determinar la Profundidad de Infiltración Ajustada ( $I_w$ )

Para riego por goteo - Ecuación 61:

$$I_w = I / f_w$$

Donde:

- $I$  = Lámina de riego aplicada sobre área total del campo (mm/evento)
- $f_w$  = Fracción de superficie mojada (goteo: 0.3 a 0.4)

Ejemplo:

- $I = 12 \text{ mm}$
- $f_w = 0.4$
- $I_w = 12 / 0.4 = 30 \text{ mm}$

```
def calculo_profundidad_infiltracion(df, fecha_inicio, fecha_fin,
f_w=0.4):

    # Filtrar por rango de fechas y precipitación > 3 mm
    dias_lluvia = df[
        (df.index >= fecha_inicio) &
        (df.index <= fecha_fin) &
        (df['Precipitaciones'] > 3)
    ]

    I_w = np.mean(dias_lluvia['Precipitaciones'] / f_w)

    return I_w
```

## Paso 3: Clasificar el Tipo de Evento de Mojado

Basado en  $I_w$ :

Tipo Evento	Rango $I_w$	Figura a usar
<b>Ligero</b>	$I_w \leq 10 \text{ mm}$	Figura 29
<b>Pesado</b>	$I_w \geq 40 \text{ mm}$	Figura 30 (a o b)
<b>Intermedio</b>	$10 < I_w < 40 \text{ mm}$	Interpolación entre Figuras 29 y 30

Selección Figura 30:

- **Figura 30a:** Suelos gruesos (arenas y arenas francos)
- **Figura 30b:** Suelos finos/medios (francos, francos arcillosos, arcillas)

```

def clasificar_tipo_evento_mojado(df, fecha_inicio, fecha_fin):

    I_w_media = calculo_profundidad_infiltracion(df, fecha_inicio,
fecha_fin)

    if I_w_media <= 10:
        return "ligero"
    elif I_w_media > 10 and I_w_media < 40:
        return "intermedio"
    elif I_w_media >= 40:
        return "pesado"

```

## Paso 4: Obtener Kc ini de las Figuras 29 o 30

### Parámetros requeridos:

- Intervalo de mojado (Paso 2)
- ETo promedio durante etapa inicial (mm/día)

### Procedimiento:

1. **Figura 29** (eventos ligeros): Leer Kc ini con intervalo y ETo
2. **Figura 30** (eventos pesados): Usar subfigura según tipo de suelo

### Interpolación (Ecuación 59):

```

def calculo_eto_etapa_inicial(df, fecha_inicio, fecha_fin):
    # Filtrar por rango de fechas y precipitación > 3 mm
    dias_etapa_inicial = df[
        (df.index >= fecha_inicio) &
        (df.index <= fecha_fin)
    ]

    return
np.mean(calculate_et0_fao_penman_monteith(dias_etapa_inicial))

dias_promedio_evento_mojado =
calculo_intervalo_promedio_eventos_mojado(MODELO_ELEGIDO,
etapas_almendro["inicial"]["inicio"], etapas_almendro["inicial"]
["fin"])
I = calculo_profundidad_infiltracion(MODELO_ELEGIDO,
etapas_almendro["inicial"]["inicio"], etapas_almendro["inicial"]
["fin"])
intesidad_evento_mojado =
clasificar_tipo_evento_mojado(MODELO_ELEGIDO,
etapas_almendro["inicial"]["inicio"], etapas_almendro["inicial"]
["fin"])

```

```

eto_etapa_inicial = calculo_eto_etapa_inicial(MODELO_ELEGIDO,
etapas_almendro["inicial"]["inicio"], etapas_almendro["inicial"]
["fin"])

intesidad_evento_mojado = "ligero"
eto_etapa_inicial = 1.5

print(dias_promedio_evento_mojado)
print(intesidad_evento_mojado)
print(eto_etapa_inicial)

0
ligero
1.5

```

Tenemos que interpolar el dato de la figura 29 y de la 30b al ser eventos mojados intermedios

```

kc_ini_fig29 = 1.2
kc_ini_fig30 = 1.2

kc_ini_figure = kc_ini_fig29 + ((I - 10) / (40 - 10)) * (kc_ini_fig30
- kc_ini_fig29)

print(kc_ini_figure)

nan

```

## Paso 5: Ajustar por Mojado Parcial (Riego por Goteo)

Ecuación 60:

$$Kc_{ini} = f_w \times Kc_{ini}(Fig)$$

```

f_w = 0.4

kc_ini = f_w * kc_ini_figure

print(kc_ini)

nan

```

## Paso 6: Consideraciones Específicas para Almendros

### Cobertura del suelo:

Condición	Rango Kc ini
Con cubierta vegetal	0.8 - 0.9
Suelo desnudo, mojado infrecuente	0.3 - 0.4

### Estado del árbol (almendros deciduos):

- **Suelo seco sin mojar:** Kc ini = 0.3 - 0.4
- **Riego frecuente/lluvia:** Ajustar valores al alza

**Nota:** Para variedad Penta, considerar condiciones específicas pre-brotación

---

## Cálculo Kc mid

[ [Volver al índice](#) ]

---

### Paso 1: Obtener el Valor Básico de Kc mid

Consulta la tabla de referencia para encontrar el valor de Kc mid para tu cultivo específico.

#### Ejemplo para almendro:

- Kc mid = 0.90 (para "Almonds, no ground cover")
- Altura del cultivo (h) = 5 m

### Paso 2: Recopilar Datos Climáticos Locales

Necesitas dos parámetros climáticos promediados para la etapa de mediados:

#### a) Velocidad del viento ( $u_2$ )

- Media diaria del viento a 2 m sobre pasto [m/s]
- Rango válido:  $1 \text{ m/s} \leq u_2 \leq 6 \text{ m/s}$

#### b) Humedad relativa mínima (RHmin)

- Media diaria de la humedad relativa mínima [%]
- Rango válido:  $20\% \leq \text{RHmin} \leq 80\%$

## Paso 3: Aplicar la Ecuación de Ajuste

Usa la ecuación para ajustar Kc mid:  $Kc\_mid\_ajustado = Kc\_mid\_tabla + [0.04 \times (u_2 - 2) - 0.004 \times (RHmin - 45)] \times (h/3)^{0.3}$

formula\_kc\_mid.png

## Paso 4: Ajuste por Frecuencia de Riego

Si riegas frecuentemente (más de cada 3 días) y:

- Kc mid de tabla < 1.0
- Fracción de suelo mojado (fw) > 0.3

Puedes aumentar Kc mid a 1.1-1.3 para considerar:

- Suelo continuamente húmedo
- Evaporación por interceptación (riego por aspersión)
- Rugosidad de la vegetación

## Paso 5: Verificar Límites de Aplicación

- Altura del cultivo:  $0.1 \text{ m} < h < 10 \text{ m}$
- Viento:  $1 \text{ m/s} \leq u_2 \leq 6 \text{ m/s}$
- Humedad:  $20\% \leq RHmin \leq 80\%$

*Si  $h < 0.1 \text{ m}$ , NO apliques la ecuación (se comporta como referencia de pasto)*

```
def calcular_kc_ajustado(df, fecha_inicio, fecha_fin, kc_tabla, h=5, verbose=True):
    """
    Calcula el Kc ajustado para un período específico

    Parámetros:
    - df: DataFrame con los datos diarios
    - fecha_inicio: fecha de inicio del período (string)
    - fecha_fin: fecha de fin del período (string)
    - kc_tabla: valor de Kc de tabla para el período
    - h: altura del cultivo (default=5)
    - verbose: si True, muestra los valores intermedios

    Retorna:
    - kc_ajustado: valor de Kc ajustado
    """
    # Convertir fechas a datetime
    fecha_inicio = pd.to_datetime(fecha_inicio)
    fecha_fin = pd.to_datetime(fecha_fin)

    # Filtrar por rango de fechas
    df_período = df[
```

```

        (df.index >= fecha_inicio) &
        (df.index <= fecha_fin)
    ]

# Verificar que hay datos en el período
if df_periodo.empty:
    raise ValueError(f"No hay datos disponibles para el período
{fecha_inicio} a {fecha_fin}")

# Calcular datos climáticos locales
velocidad_viento_media = np.mean(df_periodo["Velocidad del
viento"])
humedad_relativa_minima_media = np.mean(df_periodo["Humedad
relativa mínima"])

# Ecuación de ajuste
kc_ajustado = kc_tabla + (0.04 * (velocidad_viento_media - 2) -
0.0004 * (humedad_relativa_minima_media - 45)) * ((h / 3)**0.3)

if verbose:
    print(f"Período: {fecha_inicio.strftime('%d/%m/%Y')} - "
{fecha_fin.strftime('%d/%m/%Y')}")
    print(f"Velocidad del viento media:
{velocidad_viento_media:.2f}")
    print(f"Humedad relativa mínima media:
{humedad_relativa_minima_media:.2f}")
    print(f"Kc ajustado: {kc_ajustado:.3f}")
    print("-" * 50)

return kc_ajustado

# Calcular Kc mid
kc_mid = calcular_kc_ajustado(
    df=MODELO_ELEGIDO,
    fecha_inicio=etapas_almendro["medio"]["inicio"],
    fecha_fin= etapas_almendro["medio"]["fin"],
    kc_tabla=0.9,
    h=5
)

```

Período: 05/05/2025 - 10/08/2025  
 Velocidad del viento media: 4.76  
 Humedad relativa mínima media: 33.66  
 Kc ajustado: 1.034

-----

# Guía para Calcular Kc end

[ [Volver al índice](#) ]

## Paso 1: Obtener el Valor Básico de Kc end

Consulta la tabla de referencia para encontrar el valor de Kc end para tu cultivo específico.

**Ejemplo para almendro:**

- Kc end = 0.65
- Altura del cultivo (h) = 5 m

## Paso 2: Verificar si Aplica el Ajuste Climático

**Condición CRUCIAL:**

- **SI** Kc end (Tabla) > 0.45 → **APLICA** la ecuación de ajuste
- **SI** Kc end (Tabla) ≤ 0.45 → **NO apliques** ajuste (Kc end = Kc end Tabla)

## Paso 3: Recopilar Datos Climáticos para Etapa Final

Necesitas datos promediados para la etapa de final de temporada:

### a) Velocidad del viento ( $u_2$ )

- Media diaria del viento a 2 m sobre pasto [m/s]
- Rango válido:  $1 \text{ m/s} \leq u_2 \leq 6 \text{ m/s}$

### b) Humedad relativa mínima (RHmin)

- Media diaria de la humedad relativa mínima [%]
- Rango válido:  $20\% \leq \text{RHmin} \leq 80\%$

### c) Altura del cultivo (h)

- Altura media durante la etapa final [m]
- Rango válido:  $0.1 \text{ m} \leq h \leq 10 \text{ m}$

## Paso 4: Aplicar la Ecuación de Ajuste (si corresponde)

formula\_kc\_end.png

## Paso 5: Considerar el Manejo de Cosecha y Riego

Factor DECISIVO en Kc end:

Cosecha en verde (alto Kc end):

- Riego frecuente hasta cosecha
- Suelo y vegetación permanecen húmedos
- Kc end será relativamente alto

Cosecha en seco (bajo Kc end):

- Riego reducido o suspendido antes de cosecha
- Suelo y vegetación se secan
- Kc end será relativamente bajo

**Para cultivos de maduración temprana:**

- Probablemente cosecha en condiciones más secas
- Kc end podría ser menor que el valor de tabla

```
# Calcular Kc end
kc_end = calcular_kc_ajustado(
    df=MODELO_ELEGIDO,
    fecha_inicio=etapas_almendro["final"]["inicio"],
    fecha_fin=etapas_almendro["final"]["fin"],
    kc_tabla=0.65,
    h=5
)
```

```
Período: 11/08/2025 - 10/09/2025
Velocidad del viento media: 3.91
Humedad relativa mínima media: 35.74
Kc ajustado: 0.743
-----
```

---

## Construcción de la gráfica de Kc a lo largo de las diferentes etapas

[ [Volver al índice](#) ]

---

```

import numpy as np
import matplotlib.pyplot as plt

# --- VALORES HARDCODEADOS (Para asegurar que la gráfica funcione) ---
kc_ini = 0.40
kc_mid = 0.90
kc_end = 0.55

# Duración de las etapas (días)
L_ini, L_dev, L_mid, L_late = 15, 50, 98, 31

# Puntos de transición
dia_fin_ini = L_ini
dia_fin_dev = L_ini + L_dev
dia_fin_mid = L_ini + L_dev + L_mid
dia_fin_late = L_ini + L_dev + L_mid + L_late

# Crear arrays
dias = np.arange(0, dia_fin_late + 1)
Kc_values = np.zeros_like(dias, dtype=float)

def calcular_Kc(dia):
    if dia <= dia_fin_ini:
        return kc_ini
    elif dia <= dia_fin_dev:
        # Usamos max(1, L_dev) para evitar división por cero
        # accidental
        progreso = (dia - dia_fin_ini) / max(1, L_dev)
        return kc_ini + progreso * (kc_mid - kc_ini)
    elif dia <= dia_fin_mid:
        return kc_mid
    elif dia <= dia_fin_late:
        progreso = (dia - dia_fin_mid) / max(1, L_late)
        return kc_mid + progreso * (kc_end - kc_mid)
    else:
        return kc_end

# Llenar el array
for i, dia in enumerate(dias):
    Kc_values[i] = calcular_Kc(dia)

# --- CORRECCIÓN DEL ERROR DE LÍMITES ---
# Si max() falla por NaNs, usamos un valor por defecto
try:
    y_max = np.nanmax(Kc_values) * 1.15
    if np.isnan(y_max) or np.isinf(y_max):
        y_max = 1.2
except:
    y_max = 1.2

```

```

plt.figure(figsize=(12, 6))
plt.plot(dias, Kc_values, 'b-', linewidth=2, label='Curva de Kc')

# Rellenar el área bajo la curva para mejor visualización
plt.fill_between(dias, Kc_values, color='skyblue', alpha=0.3)

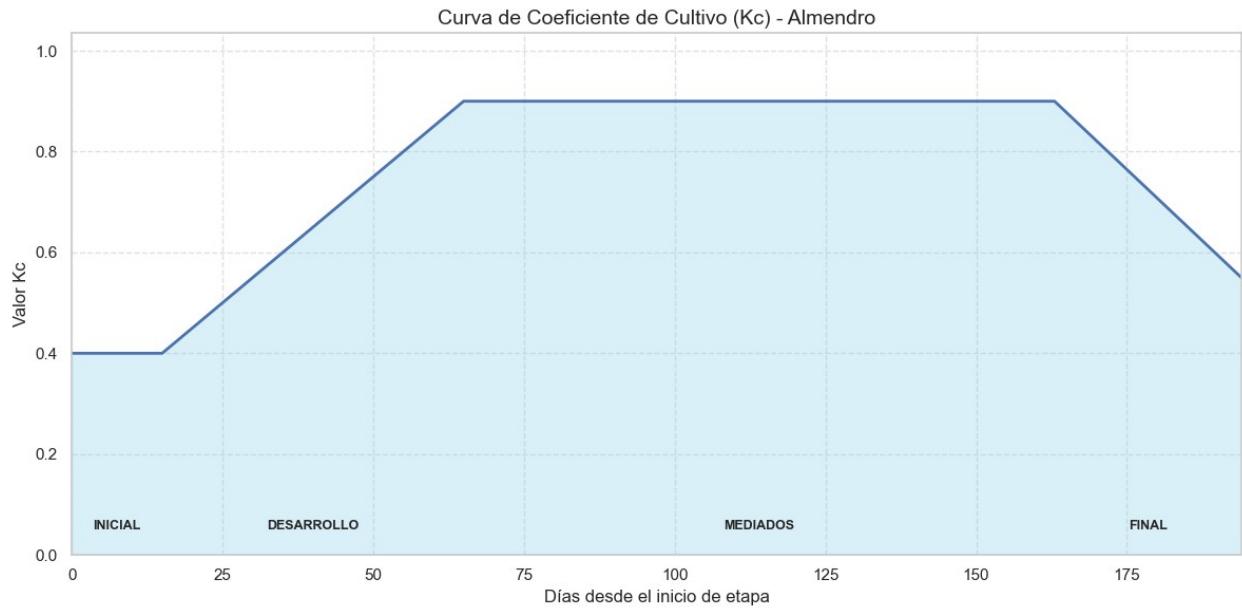
# Configuración de ejes y etiquetas
plt.ylim(0, y_max)
plt.xlim(0, dia_fin_late)
plt.grid(True, linestyle='--', alpha=0.6)
plt.title('Curva de Coeficiente de Cultivo (Kc) - Almendro',
          fontsize=14)
plt.xlabel('Días desde el inicio de etapa')
plt.ylabel('Valor Kc')

# Añadir nombres de las etapas en el gráfico
nombres = ['INICIAL', 'DESARROLLO', 'MEDIADOS', 'FINAL']
centros = [dia_fin_ini/2, (dia_fin_ini+dia_fin_dev)/2,
           (dia_fin_dev+dia_fin_mid)/2, (dia_fin_mid+dia_fin_late)/2]

for x, nombre in zip(centros, nombres):
    plt.text(x, 0.05, nombre, ha='center', fontsize=9,
              fontweight='bold')

plt.tight_layout()
plt.show()

```



# Cálculo ETc (Evotranspiración específica)

[ [Volver al índice](#) ]

---

```
# Calcular ETc para un día específico
def calcular_ETc(dia, ETo):
    """Calcula ETc para un día específico dado ETo"""
    kc = calcular_Kc(dia)
    return kc * ETo

MODELO_ELEGIDO["dia"] = MODELO_ELEGIDO.index.dayofyear

MODELO_ELEGIDO["ETc"] = MODELO_ELEGIDO.apply(lambda fila:
calcular_ETc(fila["dia"], fila["ETo"]), axis=1)

pd.set_option('display.max_rows', None)
display(MODELO_ELEGIDO)
```

	Humedad relativa	Temperatura	Velocidad del viento	\
2025-04-12	75.225680	12.824626	4.414980	
2025-04-13	75.295880	12.897439	4.428036	
2025-04-14	74.808580	12.939791	4.555692	
2025-04-15	74.216220	12.989694	4.880391	
2025-04-16	73.022860	13.382570	5.205284	
2025-04-17	71.347664	13.933340	5.504819	
2025-04-18	69.957726	14.447010	5.597008	
2025-04-19	68.477646	14.913964	5.534817	
2025-04-20	67.505750	15.225428	5.483214	
2025-04-21	66.894300	15.508081	5.445480	
2025-04-22	66.628555	15.614572	5.436857	
2025-04-23	66.605910	15.631982	5.408254	
2025-04-24	66.741860	15.633995	5.364250	
2025-04-25	66.968414	15.621820	5.316770	
2025-04-26	67.239456	15.598100	5.269108	
2025-04-27	67.523590	15.572513	5.224582	
2025-04-28	67.807610	15.565526	5.181275	
2025-04-29	68.083160	15.565923	5.143430	
2025-04-30	68.365870	15.564154	5.117047	
2025-05-01	68.665436	15.570940	5.097310	
2025-05-02	68.955260	15.566300	5.082831	
2025-05-03	69.234375	15.563714	5.071971	
2025-05-04	69.494050	15.563588	5.066979	
2025-05-05	69.727790	15.568577	5.065142	
2025-05-06	69.935265	15.578134	5.065277	
2025-05-07	70.115420	15.586374	5.065860	
2025-05-08	70.266640	15.596391	5.066580	
2025-05-09	70.388590	15.610187	5.067122	
2025-05-10	70.471940	15.631703	5.066842	

2025-05-11	70.528730	15.660814	5.065653
2025-05-12	70.555664	15.702044	5.062544
2025-05-13	70.567920	15.746706	5.058876
2025-05-14	70.567620	15.794768	5.054603
2025-05-15	70.556690	15.846026	5.049770
2025-05-16	70.536830	15.900153	5.044464
2025-05-17	70.507700	15.959335	5.039262
2025-05-18	70.472466	16.021900	5.034019
2025-05-19	70.432070	16.086252	5.028459
2025-05-20	70.379220	16.154260	5.022522
2025-05-21	70.312970	16.225925	5.015240
2025-05-22	70.238300	16.303242	5.007691
2025-05-23	70.162380	16.391523	5.001037
2025-05-24	70.086100	16.481110	4.994244
2025-05-25	70.011250	16.574425	4.986124
2025-05-26	69.937720	16.671305	4.976807
2025-05-27	69.864380	16.770410	4.965037
2025-05-28	69.788360	16.873999	4.952949
2025-05-29	69.708600	16.985178	4.942241
2025-05-30	69.623500	17.100582	4.931155
2025-05-31	69.531555	17.220491	4.919665
2025-06-01	69.429930	17.345827	4.907601
2025-06-02	69.308266	17.481699	4.893852
2025-06-03	69.172100	17.624070	4.879435
2025-06-04	69.018380	17.773602	4.864286
2025-06-05	68.862144	17.902946	4.845180
2025-06-06	68.711880	17.997179	4.820260
2025-06-07	68.503730	18.121414	4.803974
2025-06-08	68.283420	18.251057	4.789782
2025-06-09	68.057655	18.379704	4.775331
2025-06-10	67.815674	18.510267	4.760757
2025-06-11	67.548940	18.645014	4.746166
2025-06-12	67.251290	18.785383	4.731669
2025-06-13	66.918274	18.932167	4.717385
2025-06-14	66.556470	19.083357	4.700665
2025-06-15	66.160200	19.239809	4.682440
2025-06-16	65.736626	19.397408	4.657363
2025-06-17	65.312960	19.557947	4.625737
2025-06-18	64.873695	19.709806	4.596790
2025-06-19	64.412700	19.861887	4.568396
2025-06-20	63.939705	20.015432	4.544159
2025-06-21	63.417034	20.178246	4.513052
2025-06-22	62.955364	20.315584	4.495606
2025-06-23	62.586525	20.416918	4.497687
2025-06-24	62.239304	20.513190	4.501989
2025-06-25	61.919807	20.597590	4.511049
2025-06-26	61.658130	20.677107	4.531011
2025-06-27	61.432140	20.756245	4.555978
2025-06-28	61.213306	20.828962	4.580559

2025-06-29	61.001750	20.895313	4.604451
2025-06-30	60.797480	20.955528	4.627389
2025-07-01	60.569720	21.016367	4.648885
2025-07-02	60.355690	21.068464	4.667793
2025-07-03	60.152317	21.120428	4.684043
2025-07-04	59.963820	21.166298	4.696743
2025-07-05	59.787144	21.206844	4.707405
2025-07-06	59.620296	21.243162	4.716331
2025-07-07	59.458970	21.277102	4.724034
2025-07-08	59.298600	21.309520	4.730857
2025-07-09	59.141674	21.340607	4.736796
2025-07-10	58.986023	21.371147	4.741799
2025-07-11	58.826054	21.403326	4.745589
2025-07-12	58.649055	21.442080	4.747407
2025-07-13	58.431320	21.496555	4.745801
2025-07-14	58.153700	21.579927	4.738990
2025-07-15	57.806522	21.695568	4.726619
2025-07-16	57.413605	21.827744	4.711641
2025-07-17	57.028305	21.953838	4.699650
2025-07-18	56.717648	22.056700	4.699086
2025-07-19	56.444280	22.140184	4.703337
2025-07-20	56.192593	22.208906	4.710503
2025-07-21	55.956112	22.266460	4.719304
2025-07-22	55.730495	22.315590	4.728651
2025-07-23	55.513878	22.358015	4.737608
2025-07-24	55.306187	22.394753	4.745354
2025-07-25	55.108482	22.426380	4.751135
2025-07-26	54.922450	22.453276	4.754252
2025-07-27	54.753580	22.475640	4.752461
2025-07-28	54.608147	22.493507	4.743411
2025-07-29	54.502365	22.515635	4.729715
2025-07-30	54.416420	22.534433	4.711509
2025-07-31	54.351620	22.550240	4.688333
2025-08-01	54.308740	22.563446	4.659677
2025-08-02	54.290306	22.585382	4.625475
2025-08-03	54.321842	22.606997	4.591261
2025-08-04	54.383070	22.625498	4.552617
2025-08-05	54.469223	22.641070	4.509681
2025-08-06	54.552624	22.637857	4.460432
2025-08-07	54.655827	22.627462	4.408915
2025-08-08	54.787117	22.608751	4.352472
2025-08-09	54.934520	22.580510	4.297294
2025-08-10	55.089980	22.543724	4.242081
2025-08-11	55.265186	22.494530	4.189662
2025-08-12	55.491130	22.425594	4.142148
2025-08-13	55.731743	22.335900	4.102162
2025-08-14	56.014774	22.224330	4.064059
2025-08-15	56.387783	22.094572	4.036134
2025-08-16	56.813050	21.942238	4.012136

2025-08-17	57.273983	21.755173	3.991900
2025-08-18	57.780830	21.535282	3.972055
2025-08-19	58.347060	21.276367	3.952475
2025-08-20	59.052956	20.991175	3.931076
2025-08-21	60.081684	20.672783	3.895476
2025-08-22	61.542873	20.215479	3.810488
2025-08-23	63.496470	19.657460	3.757908
2025-08-24	66.003710	18.997690	3.821115
2025-08-25	67.982580	18.568659	3.851663
2025-08-26	69.605130	18.269112	3.879323
2025-08-27	70.624690	18.120584	3.882705
2025-08-28	71.225920	18.034378	3.882496
2025-08-29	71.456795	17.975320	3.877084
2025-08-30	71.484535	17.883574	3.867797
2025-08-31	71.361750	17.753773	3.864743
2025-09-01	71.165550	17.575123	3.865687
2025-09-02	70.987760	17.392113	3.863707
2025-09-03	70.843680	17.208270	3.862590
2025-09-04	70.721010	17.033873	3.860599
2025-09-05	70.653980	16.848701	3.857538
2025-09-06	70.641160	16.651400	3.851237
2025-09-07	70.655334	16.443218	3.842327
2025-09-08	70.659560	16.255243	3.826500
2025-09-09	70.640495	16.098185	3.801805
2025-09-10	70.596850	15.966593	3.770344
2025-09-11	70.537430	15.855779	3.734240
2025-09-12	70.473460	15.763170	3.696063
2025-09-13	70.431650	15.685937	3.655469
2025-09-14	70.410690	15.604089	3.615383
2025-09-15	70.423660	15.530075	3.579915
2025-09-16	70.459045	15.463474	3.549582
2025-09-17	70.515656	15.411521	3.524208
2025-09-18	70.594955	15.359430	3.502858
2025-09-19	70.679344	15.297734	3.486689
2025-09-20	70.799355	15.248343	3.471918
2025-09-21	70.938030	15.209516	3.459141
2025-09-22	71.033600	15.207192	3.444214
2025-09-23	71.149410	15.260514	3.445966
2025-09-24	71.297860	15.318026	3.452754
2025-09-25	71.467800	15.375912	3.462764
2025-09-26	71.646550	15.429811	3.475387
2025-09-27	71.820890	15.475207	3.489994
2025-09-28	71.988580	15.498934	3.505968
2025-09-29	72.148930	15.496710	3.522953
2025-09-30	72.290170	15.482785	3.540359
2025-10-01	72.459060	15.432754	3.562370
2025-10-02	72.614800	15.375790	3.584642
2025-10-03	72.777720	15.313294	3.608069
2025-10-04	72.986520	15.238349	3.639998

2025-10-05	73.239630	15.160199	3.676017
2025-10-06	73.558914	15.074017	3.716364
2025-10-07	73.975740	14.975987	3.763924
2025-10-08	74.504280	14.840746	3.825868
2025-10-09	75.369920	14.551550	3.939851
2025-10-10	76.632390	14.163325	4.093385
2025-10-11	78.180420	13.683331	4.245806
2025-10-12	79.621280	13.260023	4.474873
2025-10-13	80.873985	12.964972	4.756194
2025-10-14	82.114660	12.413604	5.124564
2025-10-15	83.155010	11.742459	5.518731
2025-10-16	83.772934	11.127421	5.774364
2025-10-17	83.972290	10.876471	5.815459
2025-10-18	84.089134	10.674532	5.818800
2025-10-19	84.137060	10.529670	5.808071
2025-10-20	84.160950	10.415526	5.791548
2025-10-21	84.150680	10.328170	5.775142
2025-10-22	84.116930	10.255722	5.757842
2025-10-23	84.070670	10.187320	5.737548
2025-10-24	84.013900	10.130377	5.709861
2025-10-25	83.945816	10.078815	5.678584
2025-10-26	83.880650	10.029536	5.647078
2025-10-27	83.811905	9.979412	5.614899
2025-10-28	83.735320	9.930774	5.593162
2025-10-29	83.655610	9.885158	5.573960
2025-10-30	83.582970	9.841228	5.554086
2025-10-31	83.519330	9.799174	5.535431
2025-11-01	83.466390	9.757691	5.519385
2025-11-02	83.422650	9.719449	5.506037
2025-11-03	83.390915	9.684566	5.495884
2025-11-04	83.371400	9.651986	5.488714
2025-11-05	83.359886	9.619158	5.483978
2025-11-06	83.355550	9.584151	5.480783
2025-11-07	83.359020	9.544024	5.478000
2025-11-08	83.370560	9.497465	5.474619
2025-11-09	83.389660	9.445276	5.470185
2025-11-10	83.417015	9.388298	5.464747
2025-11-11	83.452280	9.329926	5.458589
2025-11-12	83.495070	9.272427	5.452822
2025-11-13	83.544464	9.217359	5.448527
2025-11-14	83.598230	9.165439	5.445923
2025-11-15	83.649124	9.117517	5.440542
2025-11-16	83.701775	9.071577	5.436316
2025-11-17	83.755170	9.024772	5.433160
2025-11-18	83.807910	8.974718	5.430637
2025-11-19	83.855286	8.921333	5.428735
2025-11-20	83.897980	8.862513	5.427661
2025-11-21	83.950410	8.801373	5.433126
2025-11-22	83.998970	8.735981	5.441852

2025-11-23	84.037056	8.663819	5.450573
2025-11-24	84.069046	8.583900	5.457280
2025-11-25	84.099236	8.496535	5.459423
2025-11-26	84.143036	8.403842	5.460172
2025-11-27	84.186520	8.311686	5.463750
2025-11-28	84.227104	8.222577	5.472024
2025-11-29	84.260700	8.138454	5.481928
2025-11-30	84.290270	8.059818	5.493098
2025-12-01	84.320310	7.987067	5.505485
2025-12-02	84.344284	7.926364	5.522361
2025-12-03	84.357510	7.870454	5.543632
2025-12-04	84.361390	7.819257	5.568295
2025-12-05	84.361410	7.770551	5.597845
2025-12-06	84.361565	7.726124	5.632737
2025-12-07	84.354004	7.685183	5.669702
Presión atmosférica Radiación solar Precipitaciones \			
2025-04-12	934.74460	16281.0080	0.834385
2025-04-13	934.58340	17165.5450	0.908292
2025-04-14	934.58570	18299.7150	0.894316
2025-04-15	934.62720	19228.1780	0.817538
2025-04-16	934.85800	20411.6200	0.720192
2025-04-17	935.10230	21788.5500	0.526623
2025-04-18	935.20390	22693.9940	0.304877
2025-04-19	935.08380	23365.8930	0.171399
2025-04-20	934.93430	23702.3550	0.116096
2025-04-21	934.73535	23820.6680	0.139131
2025-04-22	934.59630	23761.2010	0.142089
2025-04-23	934.51654	23615.5080	0.148730
2025-04-24	934.46155	23416.5900	0.165135
2025-04-25	934.41620	23191.2190	0.183754
2025-04-26	934.38750	22958.1700	0.203379
2025-04-27	934.37274	22733.2460	0.223225
2025-04-28	934.37445	22518.6230	0.242613
2025-04-29	934.38635	22322.8930	0.261620
2025-04-30	934.40454	22163.0780	0.288791
2025-05-01	934.41254	22024.9570	0.322078
2025-05-02	934.41030	21897.2090	0.355617
2025-05-03	934.40240	21779.6040	0.398343
2025-05-04	934.40857	21678.3340	0.443410
2025-05-05	934.40936	21591.4920	0.489874
2025-05-06	934.40620	21517.7030	0.535791
2025-05-07	934.40295	21455.5080	0.575606
2025-05-08	934.40063	21404.8440	0.609425
2025-05-09	934.39923	21364.8610	0.637602
2025-05-10	934.40110	21337.1450	0.659033
2025-05-11	934.40510	21319.5160	0.676296
2025-05-12	934.40940	21312.1450	0.690615
2025-05-13	934.41690	21310.2930	0.701733

2025-05-14	934.42706	21313.0530	0.710170
2025-05-15	934.43970	21319.7270	0.716351
2025-05-16	934.45460	21329.7300	0.720627
2025-05-17	934.46814	21343.7480	0.724120
2025-05-18	934.48120	21360.3090	0.727005
2025-05-19	934.49580	21378.5370	0.728795
2025-05-20	934.51184	21400.1660	0.728483
2025-05-21	934.52985	21427.3180	0.726799
2025-05-22	934.54900	21458.4500	0.725812
2025-05-23	934.56730	21491.4220	0.730721
2025-05-24	934.58606	21524.4840	0.735763
2025-05-25	934.60490	21556.3670	0.742114
2025-05-26	934.62400	21587.0210	0.749790
2025-05-27	934.64610	21618.6150	0.758684
2025-05-28	934.66956	21652.2190	0.768392
2025-05-29	934.69410	21691.3280	0.779503
2025-05-30	934.71960	21732.5310	0.790786
2025-05-31	934.74630	21776.2250	0.802125
2025-06-01	934.77410	21823.1820	0.813336
2025-06-02	934.80160	21876.2640	0.824116
2025-06-03	934.83060	21934.1300	0.834270
2025-06-04	934.86150	21997.7270	0.843485
2025-06-05	934.88640	22065.7990	0.846613
2025-06-06	934.90100	22136.2500	0.841270
2025-06-07	934.88590	22199.9020	0.826434
2025-06-08	934.86150	22255.8220	0.810315
2025-06-09	934.83700	22308.4410	0.794185
2025-06-10	934.81290	22361.9800	0.776250
2025-06-11	934.78955	22419.3400	0.755119
2025-06-12	934.76685	22482.3750	0.729745
2025-06-13	934.74480	22552.2420	0.699335
2025-06-14	934.73364	22622.3480	0.666616
2025-06-15	934.73047	22695.1860	0.630041
2025-06-16	934.76080	22768.3440	0.604855
2025-06-17	934.80070	22834.5860	0.606885
2025-06-18	934.82800	22896.8610	0.617219
2025-06-19	934.84247	22963.8670	0.631635
2025-06-20	934.85840	23036.6880	0.645713
2025-06-21	934.87054	23109.0400	0.661900
2025-06-22	934.89996	23177.4670	0.678473
2025-06-23	934.95190	23241.3160	0.695044
2025-06-24	935.00946	23303.8090	0.707536
2025-06-25	935.06720	23370.8550	0.716488
2025-06-26	935.11530	23443.1250	0.714837
2025-06-27	935.17065	23509.1660	0.701927
2025-06-28	935.22300	23573.1290	0.689568
2025-06-29	935.27190	23633.9790	0.677823
2025-06-30	935.31710	23690.9820	0.666729
2025-07-01	935.36115	23751.0330	0.656424

2025-07-02	935.39606	23820.9100	0.650929
2025-07-03	935.41910	23862.6200	0.643515
2025-07-04	935.43744	23888.7420	0.635443
2025-07-05	935.45215	23912.5640	0.628476
2025-07-06	935.46454	23936.3400	0.622338
2025-07-07	935.47510	23956.2150	0.616914
2025-07-08	935.48600	23973.7010	0.612852
2025-07-09	935.49540	23988.0720	0.609146
2025-07-10	935.50366	23999.3930	0.605735
2025-07-11	935.51130	24007.5040	0.602441
2025-07-12	935.51965	24011.7910	0.598887
2025-07-13	935.53150	24010.9400	0.594373
2025-07-14	935.54840	24003.4800	0.590320
2025-07-15	935.57153	23988.4570	0.586325
2025-07-16	935.59845	23967.5820	0.580806
2025-07-17	935.62490	23946.0980	0.573210
2025-07-18	935.66595	23924.8600	0.555956
2025-07-19	935.70544	23903.6350	0.537361
2025-07-20	935.73930	23882.2050	0.519250
2025-07-21	935.76870	23858.1300	0.501577
2025-07-22	935.79425	23829.4410	0.484262
2025-07-23	935.81647	23794.5800	0.467270
2025-07-24	935.83520	23752.3200	0.450616
2025-07-25	935.85016	23701.6720	0.434357
2025-07-26	935.86096	23641.8090	0.418565
2025-07-27	935.86760	23570.5040	0.403506
2025-07-28	935.86975	23485.5100	0.389504
2025-07-29	935.86450	23417.4900	0.384916
2025-07-30	935.85333	23339.1970	0.381110
2025-07-31	935.83590	23249.7270	0.377995
2025-08-01	935.81180	23147.6950	0.375438
2025-08-02	935.77594	23029.6130	0.371872
2025-08-03	935.73120	22925.2400	0.373137
2025-08-04	935.67957	22813.4160	0.375644
2025-08-05	935.62250	22692.9670	0.378478
2025-08-06	935.56726	22542.7800	0.376858
2025-08-07	935.51060	22390.0160	0.375645
2025-08-08	935.45450	22235.5700	0.376057
2025-08-09	935.39844	22087.1930	0.378145
2025-08-10	935.33930	21940.1520	0.380518
2025-08-11	935.28204	21795.0600	0.385630
2025-08-12	935.23645	21623.4700	0.396753
2025-08-13	935.18195	21440.2660	0.412030
2025-08-14	935.11127	21231.0590	0.434414
2025-08-15	935.04900	20998.7400	0.434441
2025-08-16	934.98610	20751.0310	0.427049
2025-08-17	934.91956	20496.6800	0.419848
2025-08-18	934.84330	20216.9730	0.406517
2025-08-19	934.75710	19905.8800	0.388801

2025-08-20	934.65550	19534.2710	0.371351
2025-08-21	934.55490	19057.7010	0.397420
2025-08-22	934.66925	18362.5900	0.528931
2025-08-23	934.91364	17424.6760	0.610697
2025-08-24	935.24300	16519.1950	0.659538
2025-08-25	935.42040	15812.0200	0.727264
2025-08-26	935.46440	15237.5240	0.797059
2025-08-27	935.42950	14909.9180	0.858616
2025-08-28	935.40680	14731.2610	0.898867
2025-08-29	935.41223	14654.8040	0.906238
2025-08-30	935.48535	14605.5380	0.880438
2025-08-31	935.60910	14589.2705	0.833887
2025-09-01	935.78840	14581.4900	0.771775
2025-09-02	935.99695	14561.6980	0.716413
2025-09-03	936.20980	14530.0380	0.666558
2025-09-04	936.41440	14484.5070	0.620422
2025-09-05	936.57336	14410.8470	0.577918
2025-09-06	936.69183	14320.8510	0.538368
2025-09-07	936.83240	14180.5930	0.503986
2025-09-08	936.96985	14026.5980	0.472394
2025-09-09	937.11505	13875.8950	0.444280
2025-09-10	937.26370	13714.3560	0.417813
2025-09-11	937.40967	13542.6980	0.392599
2025-09-12	937.54760	13363.7100	0.369107
2025-09-13	937.66660	13168.5310	0.349728
2025-09-14	937.75696	12978.4240	0.331007
2025-09-15	937.83075	12784.1060	0.313833
2025-09-16	937.87744	12597.6400	0.294131
2025-09-17	937.89545	12429.2030	0.271665
2025-09-18	937.89800	12254.4920	0.249965
2025-09-19	937.88873	12061.8060	0.227364
2025-09-20	937.86505	11878.1875	0.208402
2025-09-21	937.82495	11708.2370	0.192878
2025-09-22	937.77484	11579.9230	0.188869
2025-09-23	937.68330	11491.5810	0.205224
2025-09-24	937.56730	11410.6960	0.224432
2025-09-25	937.43616	11339.1250	0.244359
2025-09-26	937.29880	11277.3870	0.263696
2025-09-27	937.16480	11225.8410	0.281108
2025-09-28	937.03400	11172.5820	0.294304
2025-09-29	936.90710	11113.5205	0.302549
2025-09-30	936.79663	11063.9730	0.307740
2025-10-01	936.70880	10995.4330	0.309469
2025-10-02	936.63360	10931.1900	0.309728
2025-10-03	936.55790	10861.5100	0.309279
2025-10-04	936.45570	10766.8770	0.308350
2025-10-05	936.33240	10660.0010	0.309572
2025-10-06	936.17760	10525.6650	0.312716
2025-10-07	935.97590	10351.4020	0.318796

2025-10-08	935.71110	10100.5260	0.325255
2025-10-09	935.25916	9590.5580	0.331196
2025-10-10	934.57650	8857.2130	0.360349
2025-10-11	933.84296	8201.1070	0.515810
2025-10-12	933.36840	7729.1997	0.643079
2025-10-13	932.87836	7380.5884	0.733444
2025-10-14	932.56070	6952.2170	0.874772
2025-10-15	932.48096	6465.3010	1.095761
2025-10-16	932.54706	6060.2850	1.189225
2025-10-17	932.75860	5967.7886	1.172641
2025-10-18	933.00640	5910.0420	1.124728
2025-10-19	933.23360	5880.8230	1.073634
2025-10-20	933.42830	5867.2314	1.023484
2025-10-21	933.57874	5867.0796	0.979006
2025-10-22	933.69560	5873.5310	0.939114
2025-10-23	933.79300	5881.5230	0.902006
2025-10-24	933.87790	5895.7666	0.869980
2025-10-25	933.95230	5910.9720	0.840270
2025-10-26	934.02234	5922.9590	0.813556
2025-10-27	934.08640	5933.9824	0.788226
2025-10-28	934.15344	5951.5337	0.771164
2025-10-29	934.22760	5983.3970	0.753711
2025-10-30	934.30035	6012.8650	0.737426
2025-10-31	934.36646	6038.3867	0.722741
2025-11-01	934.42620	6058.8050	0.709312
2025-11-02	934.47180	6073.9224	0.698692
2025-11-03	934.50244	6083.2540	0.691074
2025-11-04	934.51920	6087.0034	0.686274
2025-11-05	934.52374	6086.1350	0.683780
2025-11-06	934.51953	6081.1055	0.683097
2025-11-07	934.51210	6070.8620	0.683615
2025-11-08	934.50550	6055.1360	0.684903
2025-11-09	934.50195	6034.9453	0.686672
2025-11-10	934.50336	6010.3690	0.688535
2025-11-11	934.50696	5981.8600	0.691059
2025-11-12	934.50977	5950.0264	0.694375
2025-11-13	934.50934	5915.6950	0.698543
2025-11-14	934.50520	5879.8027	0.703569
2025-11-15	934.50604	5843.4326	0.709750
2025-11-16	934.50690	5807.0503	0.716011
2025-11-17	934.50680	5771.1426	0.722693
2025-11-18	934.50665	5736.0750	0.729663
2025-11-19	934.50840	5702.1400	0.735618
2025-11-20	934.51294	5667.9473	0.740572
2025-11-21	934.51904	5627.1396	0.747080
2025-11-22	934.52716	5585.5290	0.753163
2025-11-23	934.54070	5545.5480	0.757444
2025-11-24	934.56790	5507.3496	0.760215
2025-11-25	934.61000	5467.6978	0.761231

2025-11-26	934.65670	5423.4850	0.761699	
2025-11-27	934.69490	5380.8600	0.762450	
2025-11-28	934.71620	5340.7734	0.764314	
2025-11-29	934.72705	5304.1450	0.767076	
2025-11-30	934.72950	5270.8010	0.770612	
2025-12-01	934.72320	5239.6750	0.775115	
2025-12-02	934.69150	5213.5460	0.783092	
2025-12-03	934.64514	5192.1370	0.791780	
2025-12-04	934.58830	5175.8250	0.800864	
2025-12-05	934.52155	5165.1826	0.809427	
2025-12-06	934.44630	5160.2974	0.817715	
2025-12-07	934.36490	5159.9463	0.825855	
	Humedad relativa mínima	ET0	dia	ETc
2025-04-12	45.845673	1.257589	102	1.131830
2025-04-13	46.221250	1.260038	103	1.134035
2025-04-14	45.515476	1.310253	104	1.179228
2025-04-15	44.447453	1.401913	105	1.261722
2025-04-16	42.892647	1.549377	106	1.394440
2025-04-17	41.436886	1.739476	107	1.565528
2025-04-18	40.672836	1.880321	108	1.692289
2025-04-19	40.233200	1.996666	109	1.796999
2025-04-20	39.831080	2.071840	110	1.864656
2025-04-21	39.257690	2.125155	111	1.912639
2025-04-22	38.890200	2.148853	112	1.933967
2025-04-23	38.768406	2.144819	113	1.930337
2025-04-24	38.788060	2.125645	114	1.913080
2025-04-25	38.893333	2.098756	115	1.888880
2025-04-26	39.036247	2.068220	116	1.861398
2025-04-27	39.188290	2.037651	117	1.833886
2025-04-28	39.323902	2.009004	118	1.808104
2025-04-29	39.452286	1.982884	119	1.784595
2025-04-30	39.620487	1.958984	120	1.763086
2025-05-01	39.835983	1.936322	121	1.742690
2025-05-02	40.057840	1.914709	122	1.723238
2025-05-03	40.255417	1.894800	123	1.705320
2025-05-04	40.432910	1.877667	124	1.689901
2025-05-05	40.574974	1.863226	125	1.676904
2025-05-06	40.685740	1.851167	126	1.666050
2025-05-07	40.785915	1.840789	127	1.656710
2025-05-08	40.871530	1.832339	128	1.649106
2025-05-09	40.940178	1.825908	129	1.643317
2025-05-10	40.984474	1.822202	130	1.639982
2025-05-11	41.008965	1.820451	131	1.638406
2025-05-12	41.006640	1.820947	132	1.638852
2025-05-13	40.995660	1.822458	133	1.640213
2025-05-14	40.977085	1.824844	134	1.642360
2025-05-15	40.951927	1.827982	135	1.645184
2025-05-16	40.921165	1.831765	136	1.648588
2025-05-17	40.879850	1.836494	137	1.652845

2025-05-18	40.832100	1.841828	138	1.657645
2025-05-19	40.781586	1.847537	139	1.662783
2025-05-20	40.723026	1.854195	140	1.668776
2025-05-21	40.660110	1.861639	141	1.675475
2025-05-22	40.585464	1.869947	142	1.682952
2025-05-23	40.484535	1.879316	143	1.691384
2025-05-24	40.383175	1.888778	144	1.699900
2025-05-25	40.280040	1.898104	145	1.708294
2025-05-26	40.175390	1.907314	146	1.716582
2025-05-27	40.072950	1.916073	147	1.724466
2025-05-28	39.963470	1.925232	148	1.732709
2025-05-29	39.834248	1.935503	149	1.741953
2025-05-30	39.698322	1.946319	150	1.751687
2025-05-31	39.554115	1.957793	151	1.762014
2025-06-01	39.398730	1.970129	152	1.773116
2025-06-02	39.221786	1.984070	153	1.785663
2025-06-03	39.028755	1.999241	154	1.799317
2025-06-04	38.816520	2.015875	155	1.814288
2025-06-05	38.638610	2.030153	156	1.827138
2025-06-06	38.522926	2.039880	157	1.835892
2025-06-07	38.332195	2.057835	158	1.852052
2025-06-08	38.122215	2.077560	159	1.869804
2025-06-09	37.905476	2.097517	160	1.887765
2025-06-10	37.672400	2.118664	161	1.906797
2025-06-11	37.414960	2.141768	162	1.927591
2025-06-12	37.127155	2.167394	163	1.950655
2025-06-13	36.804573	2.195958	164	1.951570
2025-06-14	36.464302	2.226079	165	1.953205
2025-06-15	36.097200	2.258473	166	1.956129
2025-06-16	35.710136	2.290604	167	1.958097
2025-06-17	35.337460	2.320717	168	1.957637
2025-06-18	34.968230	2.351958	169	1.957436
2025-06-19	34.578860	2.384722	170	1.957780
2025-06-20	34.191980	2.419793	171	1.959252
2025-06-21	33.730034	2.456314	172	1.961089
2025-06-22	33.330868	2.491753	173	1.961251
2025-06-23	33.031790	2.525627	174	1.959398
2025-06-24	32.744980	2.558687	175	1.956157
2025-06-25	32.484646	2.590956	176	1.951575
2025-06-26	32.277756	2.623406	177	1.946398
2025-06-27	32.082325	2.655622	178	1.940317
2025-06-28	31.893240	2.686867	179	1.932811
2025-06-29	31.710447	2.716983	180	1.923800
2025-06-30	31.533915	2.745854	181	1.913240
2025-07-01	31.339233	2.776015	182	1.902914
2025-07-02	31.168163	2.803537	183	1.890127
2025-07-03	30.967470	2.829322	184	1.875567
2025-07-04	30.772520	2.852145	185	1.858494
2025-07-05	30.593351	2.872884	186	1.839572

2025-07-06	30.429129	2.891881	187	1.819087
2025-07-07	30.271496	2.909812	188	1.797513
2025-07-08	30.116714	2.927217	189	1.775216
2025-07-09	29.966352	2.943925	190	1.752110
2025-07-10	29.818323	2.960135	191	1.728337
2025-07-11	29.667347	2.976331	192	1.704190
2025-07-12	29.501510	2.993572	193	1.680263
2025-07-13	29.298592	3.013814	194	1.657598
2025-07-14	29.040926	3.038928	195	1.671410
2025-07-15	28.719078	3.069772	196	1.688374
2025-07-16	28.354294	3.104398	197	1.707419
2025-07-17	27.996687	3.139275	198	1.726601
2025-07-18	27.696710	3.171835	199	1.744509
2025-07-19	27.429237	3.202152	200	1.761184
2025-07-20	27.183851	3.230921	201	1.777006
2025-07-21	26.953926	3.258355	202	1.792095
2025-07-22	26.735060	3.284513	203	1.806482
2025-07-23	26.525349	3.309237	204	1.820080
2025-07-24	26.324705	3.332212	205	1.832717
2025-07-25	26.134205	3.353027	206	1.844165
2025-07-26	25.955551	3.371224	207	1.854173
2025-07-27	25.795546	3.385277	208	1.861902
2025-07-28	25.661797	3.393510	209	1.866431
2025-07-29	25.558718	3.396875	210	1.868281
2025-07-30	25.475359	3.396100	211	1.867855
2025-07-31	25.412834	3.390869	212	1.864978
2025-08-01	25.371658	3.380884	213	1.859486
2025-08-02	25.349075	3.367051	214	1.851878
2025-08-03	25.387949	3.349384	215	1.842161
2025-08-04	25.459000	3.326865	216	1.829776
2025-08-05	25.555323	3.299943	217	1.814969
2025-08-06	25.677505	3.268034	218	1.797419
2025-08-07	25.821737	3.232776	219	1.778027
2025-08-08	25.997877	3.192143	220	1.755678
2025-08-09	26.189110	3.150165	221	1.732591
2025-08-10	26.396494	3.106883	222	1.708786
2025-08-11	26.624872	3.062699	223	1.684484
2025-08-12	26.913391	3.016081	224	1.658844
2025-08-13	27.184298	2.970876	225	1.633982
2025-08-14	27.488285	2.922263	226	1.607245
2025-08-15	27.828396	2.871901	227	1.579545
2025-08-16	28.198130	2.818762	228	1.550319
2025-08-17	28.584530	2.762931	229	1.519612
2025-08-18	29.010853	2.702344	230	1.486289
2025-08-19	29.484154	2.635806	231	1.449693
2025-08-20	30.083857	2.558599	232	1.407229
2025-08-21	30.938440	2.453865	233	1.349626
2025-08-22	32.108482	2.294323	234	1.261878
2025-08-23	33.817860	2.119132	235	1.165523

2025-08-24	36.309510	1.956094	236	1.075852
2025-08-25	38.228905	1.827205	237	1.004963
2025-08-26	39.798870	1.726327	238	0.949480
2025-08-27	40.739532	1.661231	239	0.913677
2025-08-28	41.315570	1.622506	240	0.892378
2025-08-29	41.546170	1.604763	241	0.882620
2025-08-30	41.562977	1.595671	242	0.877619
2025-08-31	41.441517	1.594790	243	0.877135
2025-09-01	41.247177	1.596481	244	0.878065
2025-09-02	41.043380	1.595979	245	0.877788
2025-09-03	40.863674	1.593725	246	0.876549
2025-09-04	40.705303	1.590476	247	0.874762
2025-09-05	40.662216	1.583225	248	0.870774
2025-09-06	40.724590	1.571436	249	0.864290
2025-09-07	40.810955	1.556931	250	0.856312
2025-09-08	40.881493	1.542188	251	0.848204
2025-09-09	40.904790	1.527954	252	0.840375
2025-09-10	40.891110	1.514534	253	0.832994
2025-09-11	40.855587	1.501746	254	0.825960
2025-09-12	40.810870	1.489543	255	0.819249
2025-09-13	40.790527	1.476303	256	0.811966
2025-09-14	40.795715	1.461880	257	0.804034
2025-09-15	40.846684	1.447479	258	0.796113
2025-09-16	40.941162	1.433820	259	0.788601
2025-09-17	41.080200	1.421297	260	0.781713
2025-09-18	41.246610	1.408845	261	0.774865
2025-09-19	41.417732	1.397175	262	0.768446
2025-09-20	41.631620	1.384864	263	0.761675
2025-09-21	41.868435	1.372795	264	0.755037
2025-09-22	42.036324	1.363975	265	0.750186
2025-09-23	42.138157	1.361508	266	0.748829
2025-09-24	42.300700	1.359056	267	0.747481
2025-09-25	42.509200	1.356442	268	0.746043
2025-09-26	42.745010	1.353876	269	0.744632
2025-09-27	42.988434	1.351594	270	0.743377
2025-09-28	43.231810	1.348905	271	0.741898
2025-09-29	43.471554	1.345545	272	0.740050
2025-09-30	43.692352	1.342615	273	0.738438
2025-10-01	43.950172	1.337794	274	0.735787
2025-10-02	44.190125	1.333272	275	0.733300
2025-10-03	44.435600	1.328347	276	0.730591
2025-10-04	44.730694	1.322617	277	0.727439
2025-10-05	45.079770	1.315389	278	0.723464
2025-10-06	45.522950	1.305326	279	0.717929
2025-10-07	46.103832	1.291258	280	0.710192
2025-10-08	46.846134	1.272703	281	0.699986
2025-10-09	48.106834	1.240471	282	0.682259
2025-10-10	49.946148	1.189289	283	0.654109
2025-10-11	52.279850	1.116233	284	0.613928

2025-10-12	54.577076	1.060170	285	0.583093
2025-10-13	56.880230	1.021155	286	0.561635
2025-10-14	59.420692	0.976383	287	0.537011
2025-10-15	61.441826	0.933052	288	0.513178
2025-10-16	62.835888	0.897564	289	0.493660
2025-10-17	63.231700	0.880248	290	0.484136
2025-10-18	63.539913	0.866365	291	0.476501
2025-10-19	63.724370	0.857389	292	0.471564
2025-10-20	63.836285	0.850469	293	0.467758
2025-10-21	63.876656	0.846426	294	0.465534
2025-10-22	63.875370	0.844125	295	0.464269
2025-10-23	63.856790	0.842388	296	0.463313
2025-10-24	63.799580	0.841029	297	0.462566
2025-10-25	63.719147	0.840149	298	0.462082
2025-10-26	63.632780	0.839151	299	0.461533
2025-10-27	63.543884	0.838218	300	0.461020
2025-10-28	63.449013	0.838571	301	0.461214
2025-10-29	63.367393	0.839384	302	0.461661
2025-10-30	63.296997	0.839816	303	0.461899
2025-10-31	63.240025	0.839935	304	0.461964
2025-11-01	63.198240	0.839731	305	0.461852
2025-11-02	63.170677	0.839391	306	0.461665
2025-11-03	63.159084	0.838828	307	0.461355
2025-11-04	63.164013	0.837978	308	0.460888
2025-11-05	63.183826	0.836916	309	0.460304
2025-11-06	63.219110	0.835538	310	0.459546
2025-11-07	63.274517	0.833609	311	0.458485
2025-11-08	63.351818	0.830988	312	0.457043
2025-11-09	63.448032	0.827697	313	0.455233
2025-11-10	63.561653	0.823743	314	0.453059
2025-11-11	63.689130	0.819301	315	0.450616
2025-11-12	63.826336	0.814569	316	0.448013
2025-11-13	63.969086	0.809740	317	0.445357
2025-11-14	64.112564	0.804968	318	0.442732
2025-11-15	64.246120	0.800277	319	0.440152
2025-11-16	64.377075	0.795682	320	0.437625
2025-11-17	64.504265	0.791122	321	0.435117
2025-11-18	64.627240	0.786542	322	0.432598
2025-11-19	64.744090	0.782168	323	0.430193
2025-11-20	64.859640	0.777907	324	0.427849
2025-11-21	64.987230	0.773617	325	0.425490
2025-11-22	65.109010	0.769626	326	0.423294
2025-11-23	65.222000	0.765905	327	0.421248
2025-11-24	65.329410	0.762058	328	0.419132
2025-11-25	65.440190	0.757702	329	0.416736
2025-11-26	65.574970	0.752426	330	0.413834
2025-11-27	65.708374	0.747414	331	0.411078
2025-11-28	65.833800	0.743005	332	0.408653
2025-11-29	65.943540	0.739225	333	0.406574

2025-11-30	66.036920	0.735919	334	0.404755
2025-12-01	66.112840	0.732882	335	0.403085
2025-12-02	66.174126	0.730858	336	0.401972
2025-12-03	66.213880	0.729799	337	0.401390
2025-12-04	66.233470	0.729562	338	0.401259
2025-12-05	66.240960	0.729911	339	0.401451
2025-12-06	66.239030	0.730748	340	0.401912
2025-12-07	66.221730	0.732183	341	0.402700

## Cálculo Necesidad de Riego

[ [Volver al índice](#) ]

Para transformar el dato de **ETc** (Evapotranspiración del cultivo) en una cantidad de agua aplicable mediante el sistema de riego, seguiremos este proceso de tres pasos:

### 1. Cálculo de las Necesidades Netas ( $NN$ )

El objetivo es reponer el agua consumida por el árbol, descontando la aportación natural de la lluvia.

$$NN = ETc - P_e$$

- $ETc$ : Evapotranspiración del cultivo calculada (mm/día).
- $P_e$  (**Precipitación efectiva**): Agua de lluvia que realmente aprovecha el cultivo. Se suele estimar como el **75% - 80%** de la precipitación registrada (siempre que esta sea **> 5 mm**).

### 2. Ajuste por Eficiencia de Riego: Necesidades Brutas ( $NB$ )

Debido a las pérdidas por evaporación o lixiviación, debemos aplicar más agua de la que el árbol consume estrictamente. Dividimos las necesidades netas por el coeficiente de eficiencia ( $E_a$ ):

$$NB = \frac{NN}{E_a}$$

**Valores típicos de eficiencia ( $E_a$ ):**

- **Goteo:** 0.90 - 0.95
- **Microaspersión:** 0.80 - 0.85

### 3. Conversión a Litros por Árbol (A falta de datos, no llevaremos a cabo el siguiente cálculo, pero se especifica por si en un futuro se implementa)

La  $ETc$  se expresa en  $mm$ , lo que equivale a  $l/m^2$ . Para obtener el volumen por planta, multiplicamos por el marco de plantación y el coeficiente de sombreado.

**Fórmula:**

$$Litros = ETc \times \text{Marco de plantación} \times K_r$$

- **Marco de plantación:** Superficie asignada a cada árbol (ej.  $6m \times 7m = 42m^2$ ).
- **$K_r$  (Coeficiente de sombreado):** Ajuste para árboles jóvenes. Si el árbol es adulto y cubre gran parte del marco, se usa 1. Si es joven, se calcula en función del diámetro de la copa para evitar regar suelo desnudo.

**Ejemplo práctico:** Para una  $ETc$  de 5 mm en un marco de  $6 \times 7 m$  ( $42 m^2$ ):

$$5 \text{ mm} \times 42 \text{ m}^2 = 210 \text{ litros/árbol al día.}$$

```
# --- Configuración de parámetros ---
EFICIENCIA_RIEGO = 0.95 # Ajustar según sistema (0.95 para goteo)
COEF_PRECIPITACION = 0.75 # Porcentaje de lluvia aprovechable (75%)

# Definimos la Precipitación Efectiva (Pe)
# Se suele considerar efectiva si la lluvia supera un umbral mínimo (3 mm), como indica la FAO-56
def calcular_pe(precipitacion):
    if precipitacion > 3:
        return precipitacion * COEF_PRECIPITACION
    return 0

MODELO_ELEGIDO["Pe"] =
MODELO_ELEGIDO["Precipitaciones"].apply(calcular_pe)

# Calculamos Necesidades Netas (NN = ETc - Pe)
MODELO_ELEGIDO["NN"] = (MODELO_ELEGIDO["ETc"] -
MODELO_ELEGIDO["Pe"]).clip(lower=0)

# Calculamos Necesidades Brutas (NB = NN / Eficiencia)
MODELO_ELEGIDO["NB"] = MODELO_ELEGIDO["NN"] / EFICIENCIA_RIEGO

# Visualizar resultados
display(MODELO_ELEGIDO[["Precipitaciones", "ETc", "Pe", "NN", "NB"]].head())
```

	Precipitaciones	ETc	Pe	NN	NB
2025-04-12	0.834385	1.131830	0	1.131830	1.191400
2025-04-13	0.908292	1.134035	0	1.134035	1.193721
2025-04-14	0.894316	1.179228	0	1.179228	1.241292

2025-04-15	0.817538	1.261722	0	1.261722	1.328128
2025-04-16	0.720192	1.394440	0	1.394440	1.467831

## Cálculo Riego Deficitario Controlado

[ [Volver al índice](#) ]

Basado en las investigaciones especificadas en el apartado 5 de la memoria complementaria de este proyecto. Aplicaremos una estrategia de **RDC** durante la etapa de **Mediados** (específicamente en su segunda fase).

### Justificación Técnica

Según la bibliografía consultada, el almendro presenta una resistencia diferenciada al estrés hídrico dependiendo de su estado fenológico:

1. **Periodo Crítico:** Floración, cuajado y llenado rápido del grano. En estas fases, el déficit hídrico penaliza severamente la cosecha.
2. **Periodo de Resistencia (Fase II de Mediados):** Coincide con el **endurecimiento de la cáscara**. Durante este intervalo, el crecimiento del embrión es lento, lo que permite reducir drásticamente el aporte de agua sin afectar significativamente el peso final de la pepita o la inducción floral del año siguiente.

### Implementación del Estrés Hídrico

Se ha definido un nivel de **estrés del 80%** (reducción del 80% de la ETc), lo que significa que el árbol recibirá únicamente el **20% de sus necesidades hídricas teóricas**.

**Fórmula de aplicación para el periodo RDC:**

$$ETc_{RDC} = ETc \times 0.20$$

**Nota de aplicación:** Esta reducción se aplicará exclusivamente en la ventana temporal detectada (22 de junio al 10 de agosto) tras el endurecimiento de la cáscara y antes del inicio de la maduración (periodo crítico de post-cosecha).

```
# Calculamos ETc_RDC y lo aplicamos al dataframe en su propia columna
MODELO_ELEGIDO["ETc_RDC"] = MODELO_ELEGIDO.apply(lambda fila:
calcular_ETc(fila["día"], fila["ET0"]) * 0.2, axis=1)

# Calculamos Necesidades Netas (NN = ETc - Pe)
MODELO_ELEGIDO["NN_RDC"] = (MODELO_ELEGIDO["ETc_RDC"] -
MODELO_ELEGIDO["Pe"]).clip(lower=0)
```

```

# Calculamos Necesidades Brutas (NB = NN / Eficiencia)
MODELO_ELEGIDO["NB_RDC"] = MODELO_ELEGIDO["NN_RDC"] / EFICIENCIA_RIEGO

# Visualizar resultados
display(MODELO_ELEGIDO[["Precipitaciones", "ETc_RDC", "Pe", "NN_RDC",
"NB_RDC"]].head())

```

	Precipitaciones	ETc_RDC	Pe	NN_RDC	NB_RDC
2025-04-12	0.834385	0.226366	0	0.226366	0.238280
2025-04-13	0.908292	0.226807	0	0.226807	0.238744
2025-04-14	0.894316	0.235846	0	0.235846	0.248258
2025-04-15	0.817538	0.252344	0	0.252344	0.265626
2025-04-16	0.720192	0.278888	0	0.278888	0.293566

## Gráficas comparativas: Impacto de la Estrategia RDC

A continuación, se presentan dos visualizaciones clave para evaluar la viabilidad de la estrategia de **Riego Deficitario Controlado (RDC)** aplicada durante la segunda fase de la etapa de "Mediados".

- Dinámica Temporal de Riego:** En la primera gráfica comparamos la evolución diaria de las **Necesidades Brutas (NB)**. Esto permite identificar visualmente el "ahorro" diario de agua y observar cómo la transpiración del cultivo se ve recortada drásticamente durante el periodo de estrés controlado (junio-agosto).
- Balance Hídrico Acumulado:** El gráfico de barras cuantifica el impacto total de la estrategia en términos de volumen. Convertimos los milímetros acumulados a **metros cúbicos por hectárea ( $m^3/ha$ )**.

```

# Configuración de estilo
sns.set_theme(style="whitegrid")
plt.rcParams['figure.figsize'] = (12, 6)

# --- 1. Gráfica Comparativa de Evolución Temporal ---
plt.figure()
plt.plot(MODELO_ELEGIDO.index, MODELO_ELEGIDO['NB'], label='Sin RDC  
(100% ETc)', color='tab:blue', linewidth=2)
plt.plot(MODELO_ELEGIDO.index, MODELO_ELEGIDO['NB_RDC'], label='Con  
RDC (20% ETc en periodo)', color='tab:orange', linestyle='--', linewidth=2)

plt.title('Comparativa de Necesidades Brutas de Riego (NB)', fontsize=14, fontweight='bold')
plt.xlabel('Fecha')

```

```

plt.ylabel('Necesidad de Riego (mm/día)')
plt.legend()
plt.tight_layout()
plt.show()

# --- 2. Gráfico de Barras: Consumo Total Acumulado ---
# Calculamos el total (sumando mm)
total_sin_rdc = MODELO_ELEGIDO['NB'].sum()
total_con_rdc = MODELO_ELEGIDO['NB_RDC'].sum()

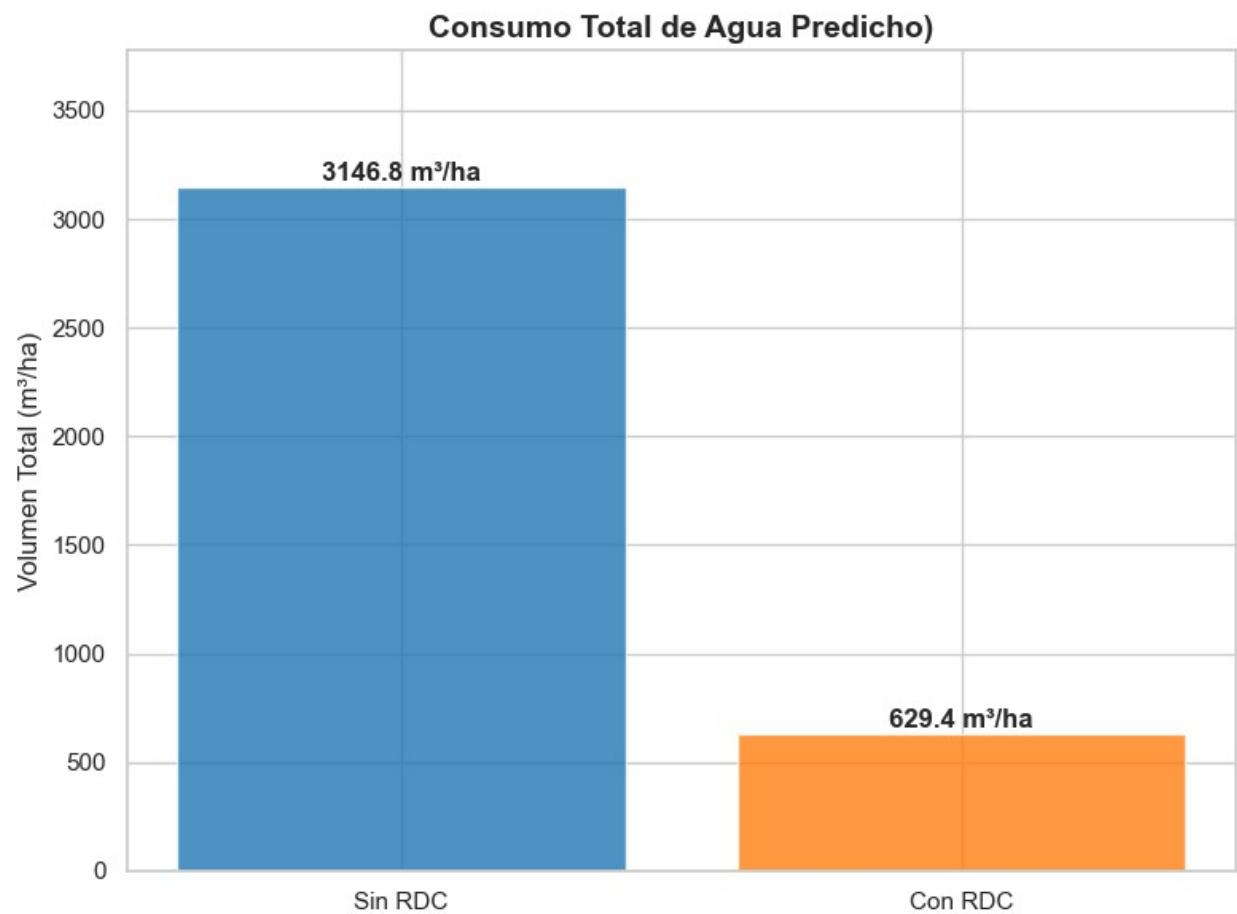
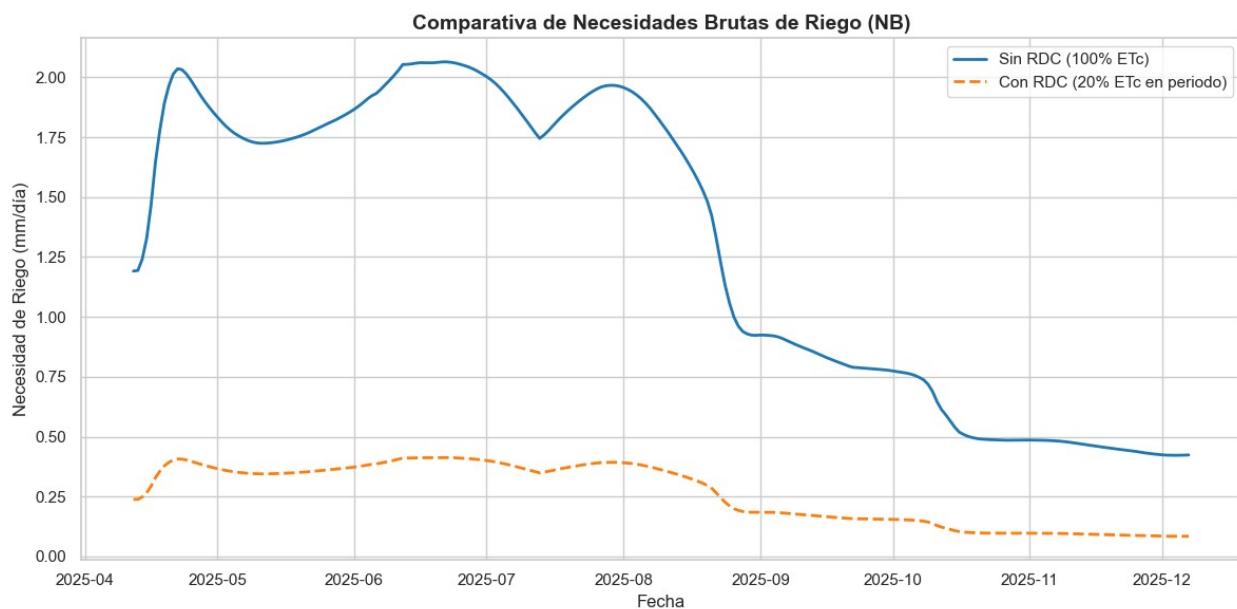
# Opcional: Convertir mm a m3/ha (1 mm = 10 m3/ha)
labels = ['Sin RDC', 'Con RDC']
totales_m3 = [total_sin_rdc * 10, total_con_rdc * 10]
ahorro = ((total_sin_rdc - total_con_rdc) / total_sin_rdc) * 100

plt.figure(figsize=(8, 6))
bars = plt.bar(labels, totales_m3, color=['tab:blue', 'tab:orange'],
alpha=0.8)

# Añadir etiquetas de valor sobre las barras
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 10,
f'{round(yval, 1)} m³/ha', ha='center', va='bottom',
fontweight='bold')

plt.title(f'Consumo Total de Agua Predicho', fontsize=14,
fontweight='bold')
plt.ylabel('Volumen Total (m³/ha)')
plt.ylim(0, max(totales_m3) * 1.2) # Espacio para la etiqueta
plt.tight_layout()
plt.show()

```



# 7. Conclusiones

[ [Volver al índice](#) ]

---

El proyecto ha desarrollado con éxito la integración de **minería de datos y estrategias de Riego Deficitario Controlado (RDC)** en el cultivo intensivo del almendro. Se ha logrado transformar datos brutos de sensores en una herramienta de decisión capaz de optimizar el consumo de agua en el cultivo del almendro.

A pesar de la correcta implementación técnica, la precisión de las predicciones se vio condicionada por la **capacidad limitada de hardware** para procesar ciclos anuales completos y la **escasez de datos históricos**.

## Líneas de trabajo futuras

### Integración en la Universidad de Burgos (UBU)

Se plantea integrar el motor de predicción desarrollado como un **módulo especializado** dentro de las plataformas agrícolas que están desarrollando proyectos en colaboración con la **UBU**. Esto permitirá que el algoritmo de RDC opere sobre infraestructuras consolidadas, beneficiándose de una gestión de datos preexistente mientras aporta su capacidad analítica específica.

### Otras líneas de trabajo futuras sobre el propio ecosistema:

1. **Meteorología en tiempo real:** Integrar la API de **AEMET** para ajustar el riego según previsiones de lluvia.
2. **Arquitectura Multicultivo:** Flexibilizar el sistema para adaptarse a diferentes fincas y variedades agrícolas.
3. **Computación de alto rendimiento:** Migrar el entrenamiento a un **clúster distribuido** para procesar series temporales extensas.
4. **Edge Computing:** Implementar el modelo en dispositivos locales para la **automatización directa** del riego.
5. **IA Explicativa:** Incorporar modelos que generen recomendaciones contextualizadas en lenguaje natural.