UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE ESTUDIOS ESTADÍSTICOS

TAREA EVALUABLE

MÓDULO: MINERÍA DE DATOS Y MODELIZACIÓN PREDICTIVA - SERIES TEMPORALES

Jorge González Perea 51553561G

Máster en Big Data, Data Science & Inteligencia Artificial

Curso académico 2024-2025

Índice

1. Introducción.

En esta práctica se va a tratar una serie temporal de datos. Este conjunto de datos representa la cantidad de energía eléctrica generada en un parque eólico de Alemania entre 2011 y 2021. Esta serie temporal se puede utilizar para estudiar tendencias en la producción de energía eólica de un parque con el fin de predecir dicha variable en el futuro, ya que las fuentes de energías renovables son cada vez más estudiadas y aseguran la producción de energía con un impacto medioambiental mucho menor que otras fuentes. La lectura inicial de los datos se encuentra en el siguiente listado:

```
datos = pd.read_csv('data.csv')
datos.columns = ['Date', 'Power']
display(datos.head())
```

Listing 1: Lectura de datos.

Nota: las librerías importadas y sus nombres asignados se recogen en la sección 6.1 del anexo.

	Date	Power
0	2011-01-01 00:00:00	3416.0
1	2011-01-01 00:15:00	4755.0
2	2011-01-01 00:30:00	4939.0
3	2011-01-01 00:45:00	4939.0
4	2011-01-01 01:00:00	4998.0

Figura 1: Vista inicial del conjunto de datos.

Se puede observar que las dos variables que se tienen son:

- **Date**: hora de la medida. Cada fila está separada 15 minutos de la anterior.
- **Power**: cantidad de energía generada por segundo (potencia) en MW.

Como se puede observar en el *DataFrame* generado, cada fila está separada de la siguiente por 15 minutos, además de que se tiene un total de **385.566** observaciones. Este número tan elevado de puntos no permite visualizar correctamente las gráficas, lo que supone una dificultad añadida para la extracción de información. Para solucionarlo, basta con agrupar las fechas por mes y sumar las medidas de las filas agrupadas, no sin antes convertir los datos de la columna **Date** a formato de fecha, así como de establecer dicha variable como el índice del *DF*. Así se obtiene una serie temporal **mensual** con un total de **132** puntos:

```
datos['Date'] = pd.to_datetime(datos['Date'], format='%Y-%m-%d %H:%M:%S')
datos.index = datos['Date']
del datos['Date']
datos = datos.resample('ME').agg({'Power': 'sum'})
display(datos.head())
```

Listing 2: Preparación de los datos.

	Power
Date	
2011-01-31	4753535.0
2011-02-28	4502270.0
2011-03-31	4948915.0
2011-04-30	4795377.0
2011-05-31	5590865.0

Figura 2: Vista final del conjunto de datos.

2. Representación gráfica de la serie temporal.

La résprentación de los datos se hace de manera muy sencilla con ayuda de las librerías *Seaborn* y *MatPlotLib*:

```
sns.lineplot(x=datos.index, y = datos['Power'], color = 'mediumorchid', label = 'Power')
plt.xticks(rotation=30, ha='right')
plt.title('Serie temporal.')
plt.grid()
plt.ylabel('Energ a producida por unidad de tiempo (MW).')
plt.legend()
plt.show()
```

Listing 3: Representación de la serie temporal.

Se obtiene así el siguiente gráfico:

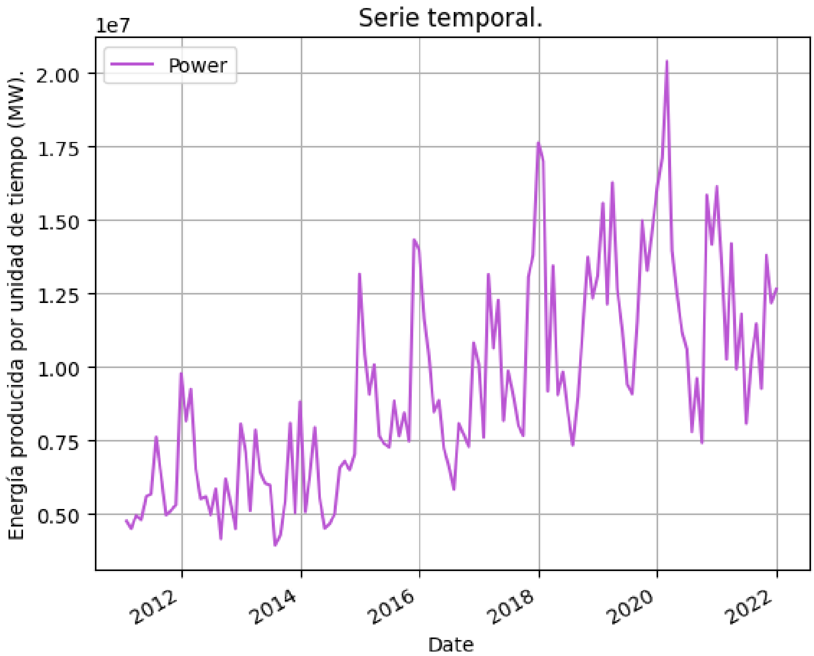


Figura 3: Serie temporal.

De la imagen 3 se obtienen las siguientes conclusiones:

- La serie presenta **estacionalidad**, ya que hay períodos de tiempo en los que el valor de la serie sigue un ciclo. Estos ciclos no son temporalmente homogéneos, ya que su duración no parece siempre la misma a primera vista.

- No es una serie estacionaria, luego presenta **tendencia** (la media aumenta con el tiempo). Se puede observar cómo la energía producida ha aumentado considerablemente en promedio durante los últimos años.

- Es una serie **heterocedástica**, ya que la varianza no es constante en el tiempo. La variación con respecto a la media, al igual que esta, aumenta en los últimos años. Es decir, hay mayor oscilación en la serie temporal.

3. Descomposición de la serie temporal.

Para descomponer la serie temporal en sus componentes de tendencia, estacionalidad, residual y observado se emplea la función *seasonal_decompose* de la librería *statsmodels*. Tal y como se ha comentado en el apartado anterior, se trata de una serie heterocedástica y con ciclos estacionales no homogéneos, por lo que para se emplea un esquema **multiplicativo** para su descomposición:

```
resultado = seasonal_decompose(datos['Power'], model = 'multiplicative')

plt.figure(figsize=(12, 12))
plt.subplot(4, 1, 1)
plt.plot(resultado.observed, label='Observado', color = 'orchid')
plt.grid()
plt.legend(loc='upper left')
plt.subplot(4, 1, 2)
plt.plot(resultado.trend, label='Tendencia', color = 'mediumorchid')
plt.grid()
plt.legend(loc='upper left')
plt.subplot(4, 1, 3)
plt.plot(resultado.seasonal, label='Estacionalidad', color = 'darkorchid')
plt.grid()
plt.legend(loc='upper left')
plt.subplot(4, 1, 4)
plt.plot(resultado.resid, label='Residual', color = 'purple')
plt.grid()
plt.legend(loc='upper left')
plt.show()
```

Listing 4: Descomposición de la serie y representación de los componentes.

Los componentes tienen el siguiente aspecto:

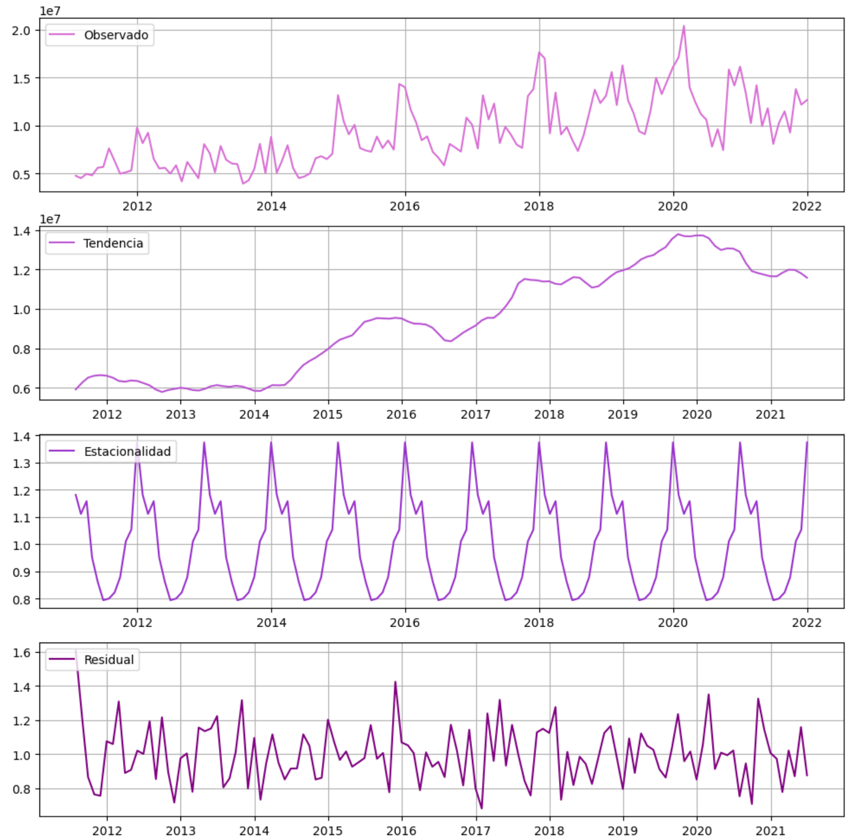


Figura 4: Descomposición de la serie temporal.

Se puede comprobar que la serie tiene una estacionalidad anual, que cuadra bastante bien con la estacionalidad esperada para una cierta producción de energía. Además, la gráfica del componente de tendencia confirma que la media es creciente. Por otro lado, los meses con menor y mayor coeficiente estacional son junio y diciembre de 2011, con los siguientes coeficientes:

Fecha	Coeficiente estacional
Junio 2011	0.795
Diciembre 2011	1.374

Tabla 1: Menor y mayor coeficiente estacional.

Es decir, en diciembre de 2011 se registró un 37,4 % más de energía producida con respecto a la media global, y en junio de ese mismo año, un 20,5 % menos. En la imagen 5 se encuentra una comparación gráfica entre los coeficientes y los errores calculados:

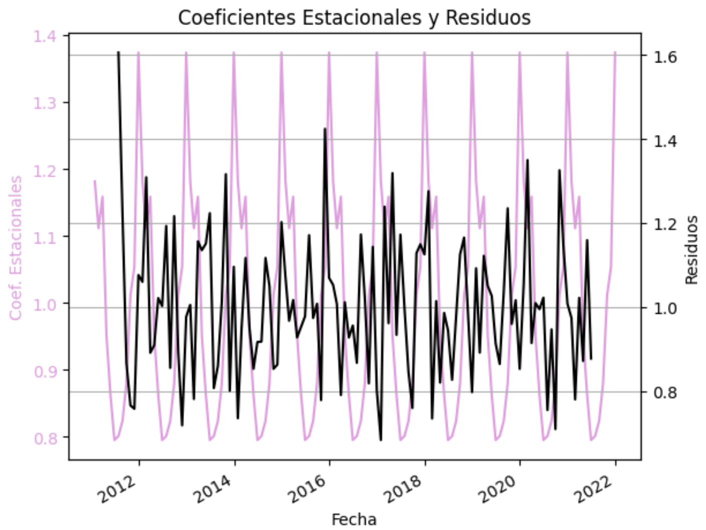


Figura 5: Coeficientes estacionales y residuos.

También conviene generar una comparación de la energía producida agrupando por años y la suma de toda la energía producida en cada año. Es evidente la tendencia mencionada:

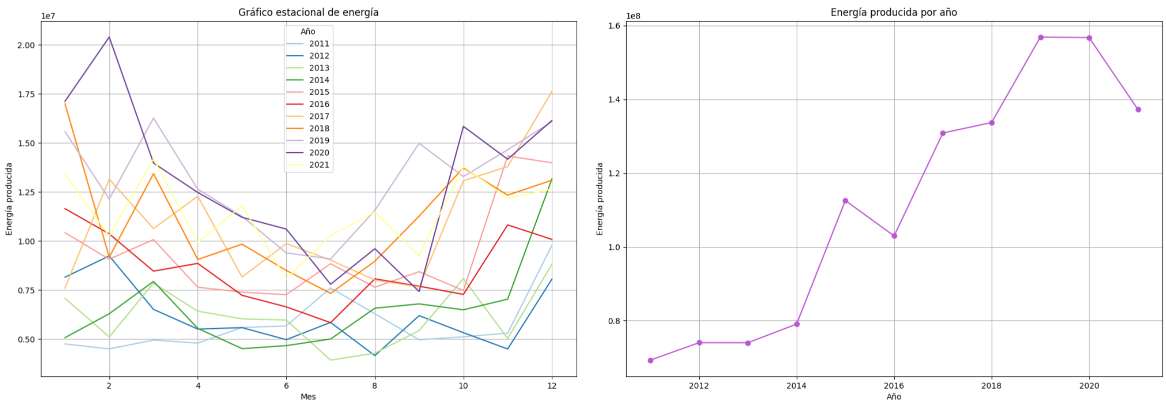


Figura 6: Izquierda: energía producida por años. Derecha: sumatorio de energía producida por años.

Se puede observar que, al agrupar los datos de cada año se genera una nueva serie temporal en la que la tendencia es clara (ver figura 6 derecha). Sin embargo, en esta serie resulta imposible distinguir la estacionalidad o la heterocedasticidad de la serie original, y por ello se eligió agrupar por meses y no años o semanas.

En la siguiente imagen se encuentra un gráfico con la serie temporal, la serie ajustada estacionalmente y la tendencia de esta.

```
plt.figure(figsize=(12, 8))
plt.plot(resultado.observed, label='Observado', color='grey')
plt.plot(resultado.observed/resultado.seasonal, label='Ajustada Estacionalmente', color=
'mediumorchid')
plt.plot(resultado.trend, label='Tendencia', color='red')
plt.legend()
plt.grid()
plt.title('Descomposici n Estacional de la Serie Temporal')
plt.xlabel('Fecha')
plt.ylabel('Cantidad')
plt.show()
```

Listing 5: Ajuste estacional.

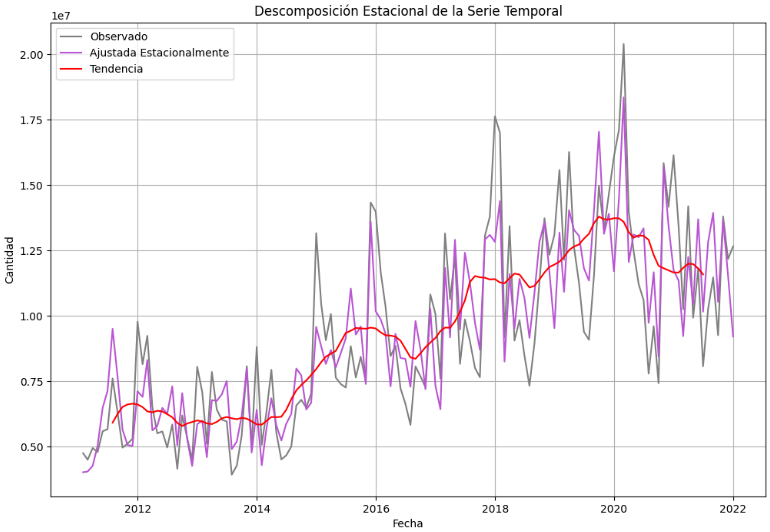


Figura 7: Ajuste estacional y tendencia de la serie.

Donde se pueden observar claramente la estacionalidad y tendencia de la serie original. Además, puede ser de utilidad dividir los datos de la serie original entre sus correspondientes coeficientes estacionales (En el caso de un esquema multiplicativo). El resultado es una nueva serie temporal desestacionalizada.

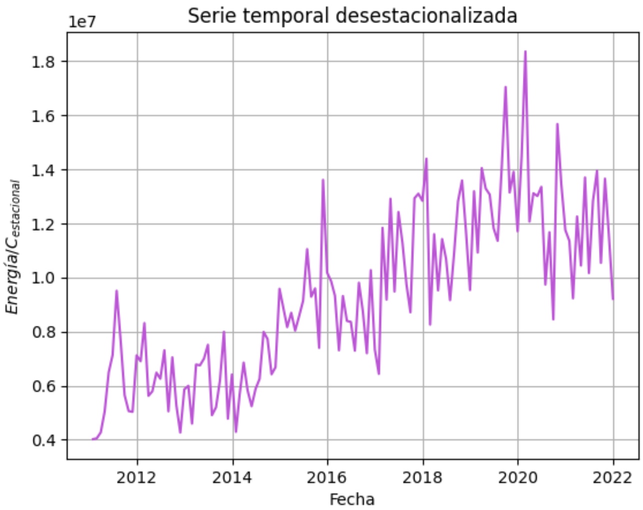


Figura 8: Serie desestacionalizada.

4. Modelos de predicción.

En este apartado se van a desarrollar modelos para poder predecir la cantidad de energía producida en un cierto tiempo futuro. Para ello se reservan algunos de los puntos del conjunto de datos original. El nuevo *DF* llamado **datos_TR** contiene, por tanto, todos los datos originales excepto aquellos reservados (aproximadamente un ciclo estacional). Estos datos se usarán para comparar las predicciones con los datos reales con el fin de estimar la bondad de los modelos.

4.1. Modelo con suavizado *Holt-Winters*.

Como la serie presenta tendencia y estacionalidad se aplica un suavizado de tipo *Holt-Winters* (también llamado suavizado triple exponencial) con las siguientes instrucciones:

```
datos_TR = datos.loc[:'2021-01-31']
datos_rest = datos['2021-01-31':]

modelo_holt_winters = sm.tsa.ExponentialSmoothing(datos_TR['Power'], trend='add',
seasonal='additive', seasonal_periods=12).fit()
predicciones_hw = modelo_holt_winters.forecast(steps=12)
modelo_holt_winters.summary()
```

Listing 6: Suavizado Holt-Winters.

El resumen del modelo dado por *Pyhton* se puede ver en la siguiente imagen, donde aparecen estadísticos como los criterios AIC o BIC:

ExponentialSmoothing Model Results

Dep. Variable:	Power	No. Observations:	120
Model:	ExponentialSmoothing	SSE	497129083278471.375
Optimized:	True	AIC	3518.285
Trend:	Additive	BIC	3562.885
Seasonal:	Additive	AICC	3525.058
Seasonal Periods:	12	Date:	Tue, 25 Feb 2025
Box-Cox:	False	Time:	17:43:05
Box-Cox Coeff.:	None		

Figura 9: Resumen del modelo con alisado HW.

Por otro lado, los resultados del modelo se pueden ver en la imagen 10, donde se disponen los valores para ciertos coeficientes de interés:

	coeff	code	optimized
smoothing_level	0.1817857	alpha	True
smoothing_trend	0.0001	beta	True
smoothing_seasonal	0.1573489	gamma	True
initial_level	6.3022e+06	l.0	True
initial_trend	19990.881	b.0	True
initial_seasons.0	1.0377e+06	s.0	True
initial_seasons.1	7.561e+05	s.1	True
initial_seasons.2	1.3725e+06	s.2	True
initial_seasons.3	-5.028e+05	s.3	True
initial_seasons.4	-1.0201e+06	s.4	True
initial_seasons.5	-1.3222e+06	s.5	True
initial_seasons.6	-6.4289e+05	s.6	True
initial_seasons.7	-1.0216e+06	s.7	True
initial_seasons.8	-6.0181e+05	s.8	True
initial_seasons.9	-2.7779e+05	s.9	True
initial_seasons.10	-1.1132e+06	s.10	True
initial_seasons.11	3.3359e+06	s.11	True

Figura 10: Resultados del modelo con alisado HW.

Donde se puede comprobar que el coeficiente $\alpha = 0{,}18$ tiene un valor cercano a 0, por lo que el modelo tendrá más en cuenta los datos más antiguos (históricos) en lugar de los más recientes. Además, el coeficiente $\beta = 1 \times 10^{-4}$ indica que el modelo mantendrá una tendencia más estable. El último parámetro, $\gamma = 0{,}16$, provoca que la estacionalidad del modelo no cambie mucho a lo largo del tiempo.

Gracias a estas funciones se obtiene un conjunto de datos de la longitud del ciclo seleccionado que actúa como predicción. Estos datos se pueden representar junto a la serie temporal original, haciendo distinción entre los datos normales, los que se han reservado y los calculados. De esta forma se puede ver mejor la calidad de la predicción del modelo:

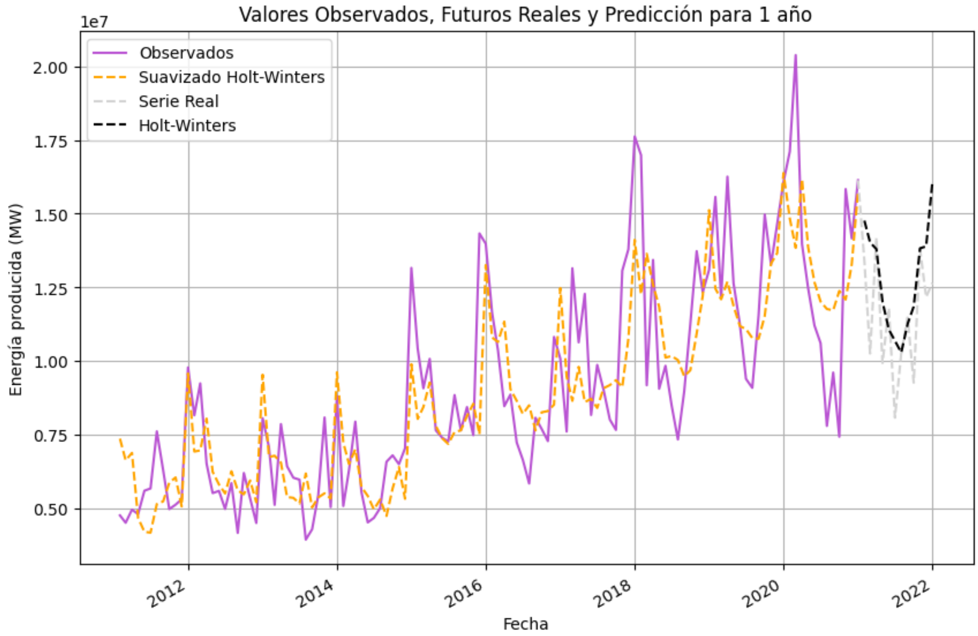


Figura 11: Representación de las predicciones del modelo con alisado HW.

Los errores de la predicción se calculan de forma muy sencilla con los datos originales que no se han usado. Estos cálculos vienen en la tabla 2.

```
errores = abs(datos_rest['Power']-predicciones_hw)
errores = errores.dropna()
display(errores)
```

Listing 7: Errores de la predicción de Holt-Winters.

Fecha	Error (10^6 MW)	Error absoluto (%)
2021-01-31	1.3636	10.2
2021-02-28	3.7736	36.8
2021-03-31	0.3919	2.8
2021-04-30	2.0840	21.0
2021-05-31	0.7315	6.2
2021-06-30	2.6098	32.3
2021-07-31	0.0170	0.2
2021-08-31	0.2187	1.9
2021-09-30	2.5886	28.0
2021-10-31	0.0281	0.2
2021-11-30	1.7198	14.1
2021-12-31	3.4392	27.2

Tabla 2: Tabla de errores por fecha.

Como se puede observar, en cinco de los puntos (cinco meses) se tiene un error absoluto de aproximadamente el 10 % o menor (en algunos incluso menor del 2 %), por lo que las predicciones se ajustan en cierta medida a los datos reales. Como se puede ver en la imagen 11, las predicciones cumplen con la tendencia dentro del ciclo y con la estacionalidad del mismo.

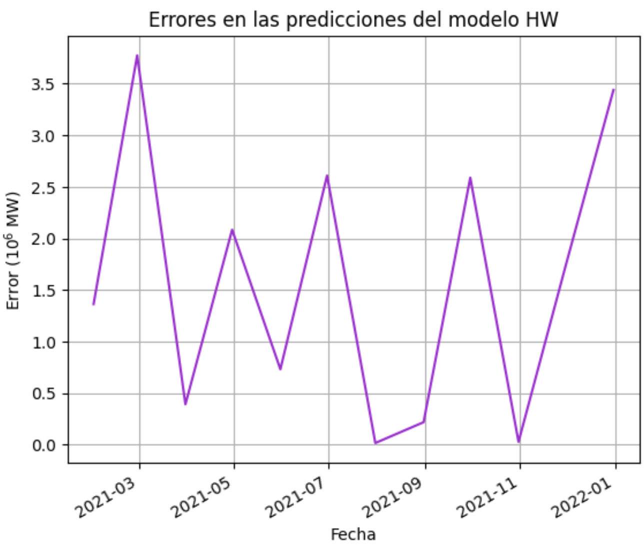


Figura 12: Representación de los errores del modelo con alisado HW.

La **media** de los errores en el cálculo de las predicciones tiene un valor de $1{,}5805 \times 10^6$. Sin embargo, un segundo intento de predicción tomando el logaritmo de los datos puede dar lugar a resultados mejores:

```
datos_log = np.log(datos_TR['Power'])
datos_log_diff = datos_log.diff().dropna()
modelo_hw_diff = sm.tsa.ExponentialSmoothing(datos_log_diff, trend='add', seasonal='add', seasonal_periods=12).fit()
pred_diff = modelo_hw_diff.forecast(steps=12)
last_log_value = datos_log.iloc[-1]
pred_log = last_log_value + pred_diff.cumsum()
predicciones_log = np.exp(pred_log)
modelo_hw_diff.summary()
```

Listing 8: Predicción con logaritmos y suavizado HW.

Los resultados de este nuevo modelo son los siguientes:

ExponentialSmoothing Model Results			
Dep. Variable:	Power	No. Observations:	119
Model:	ExponentialSmoothing	SSE	6.599
Optimized:	True	AIC	-312.171
Trend:	Additive	BIC	-267.705
Seasonal:	Additive	AICC	-305.331
Seasonal Periods:	12	Date:	Tue, 25 Feb 2025
Box-Cox:	False	Time:	18:48:54
Box-Cox Coeff.:	None		

	coeff	code	optimized
smoothing_level	1.4902e-08	alpha	True
smoothing_trend	1.4901e-08	beta	True
smoothing_seasonal	1.557e-12	gamma	True
initial_level	0.0211206	l.0	True
initial_trend	-0.0001778	b.0	True
initial_seasons.0	-0.0551702	s.0	True
initial_seasons.1	0.0281760	s.1	True
initial_seasons.2	-0.1715132	s.2	True
initial_seasons.3	-0.0993826	s.3	True
initial_seasons.4	-0.0482723	s.4	True
initial_seasons.5	-0.0505083	s.5	True
initial_seasons.6	0.0436905	s.6	True
initial_seasons.7	0.0567226	s.7	True
initial_seasons.8	0.1327447	s.8	True
initial_seasons.9	0.0355005	s.9	True
initial_seasons.10	0.2720572	s.10	True
initial_seasons.11	-0.1617529	s.11	True

Figura 13: Resumen y resultados del modelo con alisado HW con **logaritmos**.

De esta forma se obtiene la siguiente predicción:

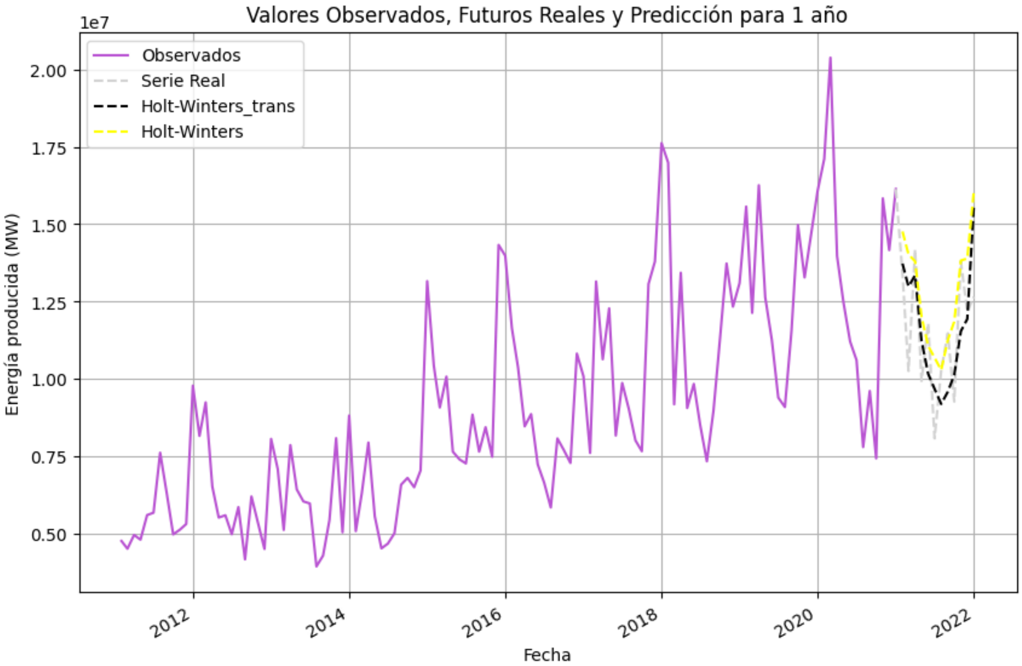


Figura 14: Representación de las predicciones del modelo con alisado HW con **logaritmos**.

Donde se puede observar que toda la curva de las nuevas predicciones (curva negra) se ajusta un poco mejor a los datos reales (gris). De nuevo, se calculan los errores para estos cálculos con respecto a los datos reales, así como sus errores absolutos.

```
    errores_log = abs(datos_rest['Power']-predicciones_log)
    errores_log = errores_log.dropna()
    errores_absolutos_log = 100*errores_log/datos_rest['Power']
    errores_absolutos_log = errores_absolutos_log.dropna()
    df_errores_log = pd.DataFrame({'Fecha': errores_log.index,
                                   'Error': errores_log,
                                   'Error absoluto': errores_absolutos_log})
    df_errores_log.index = df_errores_log['Fecha']
    df_errores_log = df_errores_log.drop('Fecha', axis = 1)
```

Listing 9: Errores de la predicción de Holt-Winters con logaritmos.

Fecha	**Error** (10^6 **MW**)	**Error absoluto** (%)
2021-01-31	0.3228	2.4
2021-02-28	2.7314	26.6
2021-03-31	0.8416	5.9
2021-04-30	1.3068	13.2
2021-05-31	1.6331	13.8
2021-06-30	1.6002	19.8
2021-07-31	1.0818	10.5
2021-08-31	1.8905	16.5
2021-09-30	0.8630	9.3
2021-10-31	2.2574	16.4
2021-11-30	0.2374	2.0
2021-12-31	2.9771	23.5

Tabla 3: Tabla de errores con logaritmos.

Además, estos errores se pueden observar gráficamente para un mejor entendimiento de su magnitud y sus diferencias. Tal y como se puede observar en la gráfica de la figura 15 o en la tabla 3, estos se han reducido en casi todos los puntos (para 7 de 12 meses los nuevos cálculos son menores). Por otro lado, el **promedio** de estas nuevas diferencias tiene un valor de $1{,}4786 \times 10^6$, que es menor que la media para el cálculo anterior. Es decir, se ha reducido el error general en la predicción a costa de que el modelo apenas reaccione ante nuevos datos debido a los valores de la figura 13.

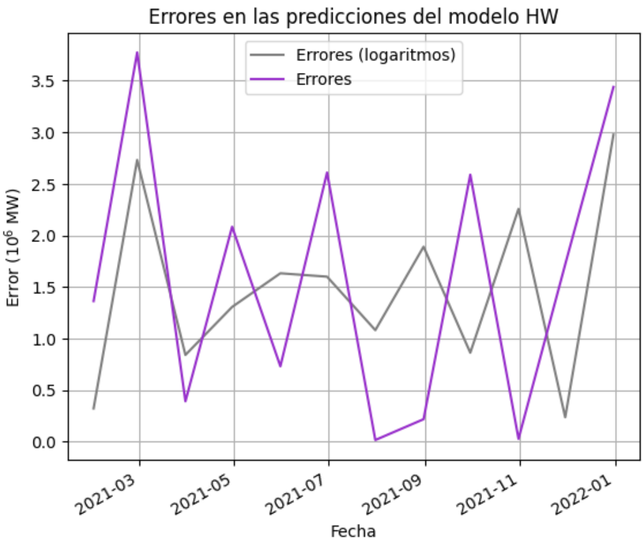


Figura 15: Representación de los errores del modelo con alisado HW con **logaritmos**.

Tal y como se puede observar en la imagen 10, el nuevo modelo logarítmico tiene unos parámetros $\alpha \approx 0{,}18$, $\beta \approx 1 \times 10^{-4}$ y $\gamma \approx 16$, por lo que se pueden sacar las siguientes conclusiones sobre el modelo:

- Los datos del pasado serán de gran importancia debido al valor de α, pero el modelo reaccionará a datos nuevos moderadamente ($\alpha > 0$).
- La tendencia cambiará muy poco ($\beta \approx 1 \times 10^{-4}$). Es decir, esta seguirá existiendo (la serie segurá teniendo tendencia), pero no cambiará de forma notable (la tendencia no cambiará, pero el crecimiento y/o decrecimiento de la serie se mantendrá).
- La estacionalidad no sufrirá variaciones bruscas, pero sí notables ($\gamma \approx 0{,}16$).

Por tanto y teniendo en cuenta las características estacionales de la serie, la expresión algebraica del modelo será la siguiente:

$$\hat{X}_t = (L_t + b_t) + S_{t+1-s} \tag{1}$$

Donde se ha empleado un esquema multiplicativo para la descomposición, pero aditivo para el suavizado (ya que la estacionalidad no aumenta con el tiempo), por lo que los términos L_t, b_t y S_t tienen la siguiente forma:

$$L_t = \alpha(X_t - S_{t-s}) + (1-\alpha)(L_{t-1} + b_{t-1}) \approx 0{,}18(X_t - S_{t-s}) + 0{,}82(L_{t-1} + b_{t-1}) \tag{2}$$

La componente de tendencia se puede aproximar (gracias al valor de beta) de forma que la tendencia se mantiene constante y no depende de los valores que toma la serie en ningún momento:

$$b_t = \beta(L_t - L_{t-1}) + (1-\beta)b_{t-1} \approx b_{t-1} \tag{3}$$

Y por último:

$$S_t = \gamma(X_t - L_t) + (1-\gamma)S_{t-s} \approx 0{,}16(X_t - L_t) + 0{,}84S_{t-s} \tag{4}$$

4.2. Funciones de autocorrelación (ACF) y autocorrelación parcial (PACF).

La función de autocorrelación simple (ACF) permite comprobar de manera sencilla si la serie presenta correlación consigo misma. Por otro lado, la función de autocorrelación parcial (PACF) es de gran ayuda para poder identificar el modelo autorregresivo más correcto para la serie temporal. Los gráficos de estas funciones se genera con facilidad gracias a la librería *Statsmodels*:

```
fig, ax = plt.subplots(figsize=(10, 8))
plot_acf(datos_TR['Power'],lags=48, alpha=0.05, ax = ax, color = 'mediumorchid', vlines_kwargs={"colors": 'mediumorchid'})
for item in ax.collections:
    if type(item)==PolyCollection:
        item.set_facecolor('mediumorchid')
plt.title('Funci n de correlaci n simple (ACF)')
plt.show()

fig, ax = plt.subplots(figsize=(10, 8))
plot_pacf(datos_TR['Power'], lags=48, alpha=0.05, ax = ax, color = 'mediumorchid', vlines_kwargs={"colors": 'mediumorchid'})
for item in ax.collections:
    if type(item)==PolyCollection:
        item.set_facecolor('mediumorchid')
plt.title('Funci n de correlaci n parcial (PACF)')
plt.show()
```

Listing 10: Código para generar los gráficos de ACF y PACF.

Tal y como se ve en la imagen 16, la serie es estacionaria ya que el correlograma de la ACF decrece lentamente (con cambios de tendencia de la propia ACF). Además, se tienen varios marcadores que se extienden por encima del intervalo sombreado, indicando que en esos puntos existe autocorrelación significativa entre dichos puntos y el valor asociado al instante de tiempo anterior (es decir, al mes anterior). Por otro lado, el crecimiento y decrecimiento mencionados indican un comportamiento estacional.

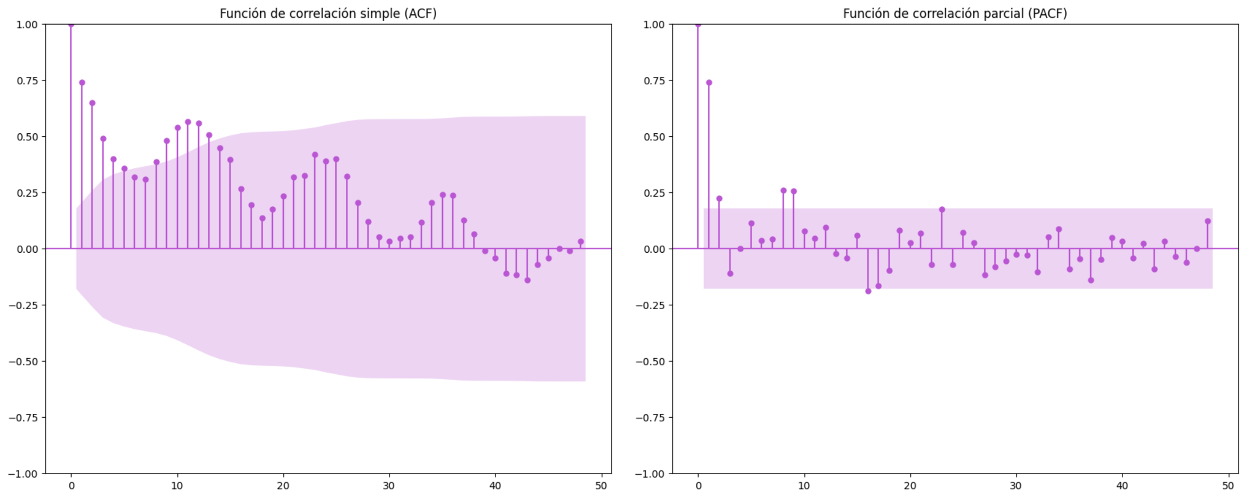


Figura 16: Funciones de autocorrelación simple (ACF) y autocorrelación parcial (PACF) de izquierda a derecha respectivamente.

La PACF indica correlación entre dos puntos para instantes de tiempo distintos. En este caso, hay un gran número de puntos en los que no hay correlación significativa ya que se encuentran en el intervalo sombreado. Por tanto esos puntos no serán tan relevantes a la hora de desarrollar el modelo AR (no hace falta incluirlos).

Como la serie no es estacionaria en la media y tampoco en la varianza, el modelo autoregresivo a elegir es el modelo ARIMA estacional (sARIMA), con tres parámetros no estacionales y cuatro estacionales. Es decir, será un modelo $sARIMA(d,p,q)(D,P,Q)_s$.

4.3. Modelo autorregresivo. Parámetros no estacionales.

Antes de comenzar con el desarrollo del modelo es necesario llevar a cabo ciertas pruebas que proporcionen valores estadísticos fiables que den cierta noción sobre la estacionariedad de la serie. Se pueden llevar a cabo pruebas como la prueba Dickey-Fuller o Kwiatkowski-Phillips-Schmidt-Shin (KPSS). A continuación se muestra el código para llevar a cabo dichas pruebas sobre la serie original y la serie diferenciada a orden I:

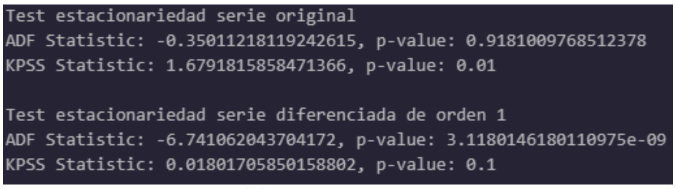
```
power = datos_TR['Power']
datos_diff_1 = power.diff().dropna()
sns.lineplot(datos_diff_1*1e-6, color = 'mediumorchid')
plt.title('Serie temporal diferenciada')
plt.grid()
plt.xlabel('Fecha')
plt.xticks()
plt.ylabel(r'Power: serie diferenciada ($10^6$)')
plt.show()

print('Test estacionariedad serie original')
print(f'ADF Statistic: {adfuller(datos_TR['Power'])[0]}, p-value: {adfuller(datos_TR['
Power'])[1]}')
print(f'KPSS Statistic: {kpss(datos_TR['Power'])[0]}, p-value: {kpss(datos_TR['Power'])
[1]}')

print('\nTest estacionariedad serie diferenciada de orden 1')
print(f'ADF Statistic: {adfuller(datos_diff_1)[0]}, p-value: {adfuller(datos_diff_1)[1]}
')
print(f'KPSS Statistic: {kpss(datos_diff_1)[0]}, p-value: {kpss(datos_diff_1)[1]}')
```

Listing 11: Pruebas de Dickey-Fuller y KPSS.

Y se obtiene que, en efecto, la serie original NO es estacionaria pero la diferenciación sí lo es:



```
Test estacionariedad serie original
ADF Statistic: -0.35011218119242615, p-value: 0.9181009768512378
KPSS Statistic: 1.6791815858471366, p-value: 0.01

Test estacionariedad serie diferenciada de orden 1
ADF Statistic: -6.741062043704172, p-value: 3.1180146180110975e-09
KPSS Statistic: 0.01801705850158802, p-value: 0.1
```

Figura 17: Prueba de Dickey-Fuller y KPSS para la serie original y la serie diferenciada de orden I.

Donde la diferenciación no estacional tiene la siguiente forma:

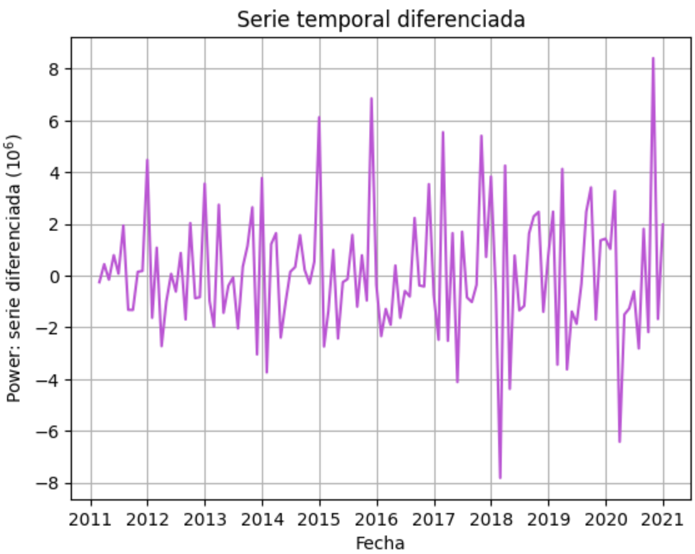


Figura 18: Serie diferenciada de orden I.

Las funciones ACF y PACF para la serie diferenciada son las siguientes, donde se puede comprobar que basta con una diferenciación no estacional para que la serie se haga estacionaria:

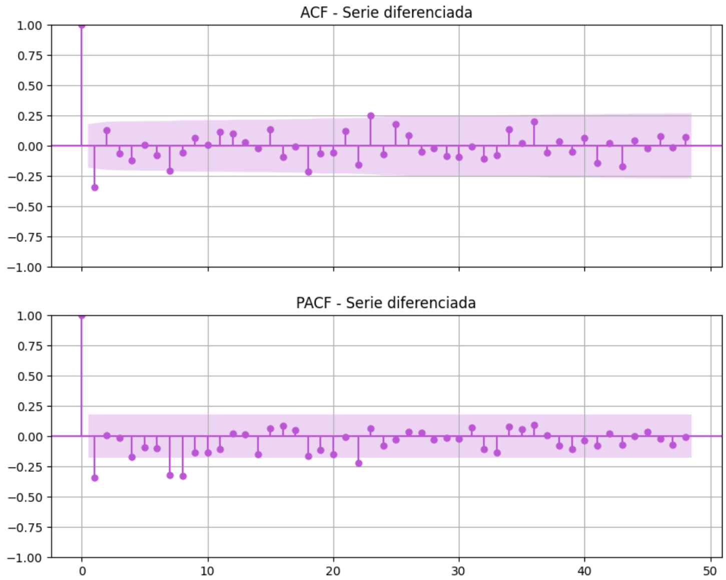


Figura 19: ACF y PACF de la serie diferenciada.

Si se quiere desarrollar un modelo ARIMA con parámetros no estacionales d, p y q, es necesario determinar primero el parámetro d, que hace referencia al orden de diferenciación al que se ha llevado la serie para hacerla estacionaria. Como una diferenciación ha sido suficiente, se toma el valor $d = 1$. Además, en la PACF no hay autocorrelaciones significativas a partir del valor 2, por lo que se toma $p = 1$. Como en la ACF ocurre algo parecido, se toma el parámetro $q = 1$. Toda la información necesaria se puede observar en la imagen de la descomposición estacional de la serie diferenciada, donde por ejemplo, se observa que la tendencia no tiene un crecimiento o decrecimiento monótono ya que la nueva serie es estacionaria.

Figura 20: Descomposición estacional de la serie diferenciada a orden I.

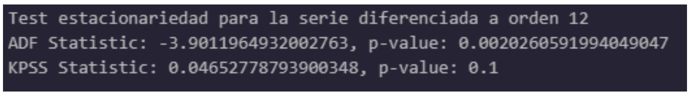
4.4. Modelo autorregresivo. Parámetros estacionales.

Para la parte estacional del modelo ARIMA se toma el valor s = 12 debido a la estacionalidad de la serie. El resto de parámetros (D, P y Q) se determinan a partir de la diferenciación estacional de la serie. Para ello bastaría con diferenciar la serie de forma no estacional(como en el anterior ejemplo con *datos_ diff_ 1*) y combinarla con una nueva diferenciación estacional (*datos_ diff_ 1_ 12*).

```
datos_diff_1_12 = datos_diff.diff(12).dropna()
print('Test estacionariedad serie de orden 12')
print('\nTest estacionariedad serie diferenciada de orden 12')
print(f'ADF Statistic: {adfuller(datos_diff_1_12)[0]}, p-value: {adfuller(datos_diff_1_12)[1]}')
print(f'KPSS Statistic: {kpss(datos_diff_1_12)[0]}, p-value: {kpss(datos_diff_1_12)[1]}')
```

Listing 12: Diferenciación estacional de la serie temporal.

Los resultados de las pruebas de *Dickey-Fuller* y KPSS son:



```
Test estacionariedad para la serie diferenciada a orden 12
ADF Statistic: -3.9011964932002763, p-value: 0.0020260591994049047
KPSS Statistic: 0.04652778793900348, p-value: 0.1
```

Figura 21: Prueba de *Dickey-Fuller* y KPSS para la serie diferenciada de orden XII (diferenciación estacional).

Donde se puede observar que no es estacionaria. Esto también se puede comprobar en la ACF y PACF de la diferenciación de orden XII sobre la diferenciación de orden I:

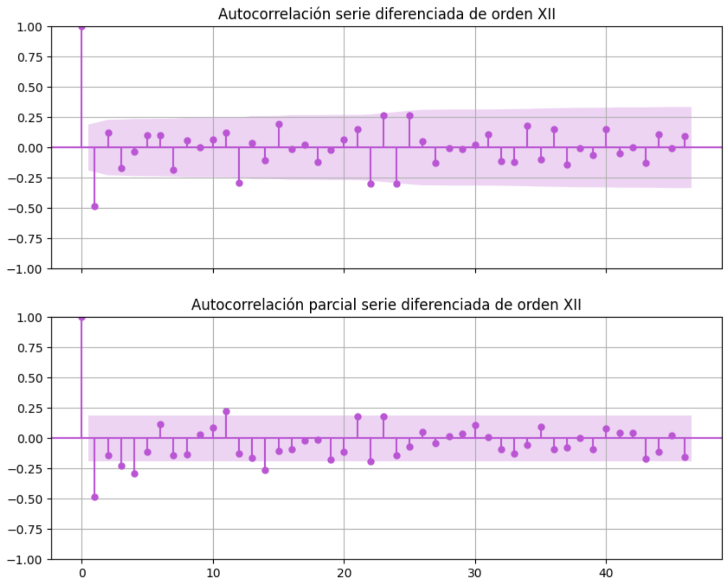


Figura 22: ACF y PACF de la serie tras la combinación de diferenciaciones de orden I y XII.

Como la serie diferenciada no presenta comportamiento estacional significativo, el valor del parámetro D es $D = 1$, ya que sólo ha sido necesario diferenciarla una vez. Como no hay cortes significativos en la PACF estacional a partir del primer ciclo estacional (12 meses) se puede considerar que $P = 1$ aunque haya ciertos valores significativos posteriores (hay aproximadamente uno cada 12 marcas debido a la estacionalidad). Como en la ACF ocurre algo parecido, se toma el parámetro $Q = 0$.

Por tanto, los parámetros elegidos para el modelo se recogen en la siguiente tabla:

Parámetro	**Valor**
d	1
p	1
q	1
D	1
P	1
Q	0
s	12

Tabla 4: Parámetros del modelo ARIMA estacional

Por lo que el modelo elegido es un modelo $sARIMA(1,1,1)(1,1,0)_{12}$. La expresión algebraica que representa a un modelo cualquiera es la siguiente:

$$\left(1-\sum_{n=1}^{P}\Phi_n B^{ns}\right)\left(1-\sum_{n=1}^{p}\phi_n B^{n}\right)(1-B^s)^D(1-B)^d X_t=\left(1-\sum_{n=1}^{Q}\Theta_n B^{ns}\right)\left(1-\sum_{n=1}^{q}\theta_n B^{n}\right)Z_t \quad (5)$$

Y aplicada al modelo $sARIMA(1,1,1)(1,1,0)_{12}$ se tiene la siguiente expresión:

$$(1-\Phi_1 B^{12})(1-\phi_1 B)(1-B^{12})(1-B)X_t=(1-\theta_1 B)Z_t \quad (6)$$

4.5. Predicciones del modelo $sARIMA$.

Con la función de la librería *statsmodels* es posible recrear el modelo para los parámetros elegidos en el apartado anterior.

```
warnings.filterwarnings("ignore", category=UserWarning, message='Non-invertible|Non-stationary')
modelo = SARIMAX(endog = datos, order = (1, 1, 1), seasonal_order = (1, 1, 0, 12))
modelo_res = modelo.fit(disp=0)
warnings.filterwarnings("default")
modelo_res.summary()
```

Listing 13: Modelo sARIMA.

Los resultados obtenidos son los siguientes:

SARIMAX Results

Dep. Variable:	Power	No. Observations:	120
Model:	SARIMAX(1, 1, 1)x(1, 1, [], 12)	Log Likelihood	-1728.038
Date:	Tue, 11 Mar 2025	AIC	3464.076
Time:	21:32:15	BIC	3474.767
Sample:	01-31-2011	HQIC	3468.410
	- 12-31-2020		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.0836	0.237	-0.353	0.724	-0.548	0.381
ma.L1	-0.6866	0.160	-4.301	0.000	-1.000	-0.374
ar.S.L12	-0.5128	0.118	-4.351	0.000	-0.744	-0.282
sigma2	8.076e+12	2.3e-15	3.51e+27	0.000	8.08e+12	8.08e+12

Ljung-Box (L1) (Q):	0.01	Jarque-Bera (JB):	10.85
Prob(Q):	0.92	Prob(JB):	0.00
Heteroskedasticity (H):	2.18	Skew:	0.50
Prob(H) (two-sided):	0.02	Kurtosis:	4.19

Figura 23: Resultados del modelo sARIMA.

Se puede realizar el test de Ljung-Box para comprobar la bondad del modelo, obteniendo las medidas estadísticas de la tabla 5, donde se puede comprobar que el valor del estadístico p supera el umbral de 0,05, por lo que la serie temporal se considera un ruido blanco:

Estadístico LB	**Valor p**
14.4789	0.2712

Tabla 5: Resultados del test de Ljung-Box.

En el siguiente listado se encuentra el código para generar las predicciones del modelo para un período de 48 observaciones (4 años):

```
predicciones_statsmodels = modelo_res.get_forecast(steps=48).predicted_mean
predicciones_statsmodels.name = 'predicciones_statsmodels'
display(predicciones_statsmodels.head(4))

fig, ax = plt.subplots(figsize=(9, 5))
datos.plot(ax=ax, label='Serie', color = 'mediumorchid')
datos_rest.plot(ax=ax, label='Reales', color = 'gray')
predicciones_statsmodels.plot(ax=ax, label='sARIMA', color = 'orange')
ax.set_title(r'Predicciones del modelo $sARIMA(1, 1, 1)(1, 1, 0)_{12}$')
plt.grid()
ax.legend(['Serie original', 'Datos reales', 'ARIMA']);
plt.show()
```

Listing 14: Modelo sARIMA.

Los resultados se representan en la siguiente imagen junto a la serie original y los datos reservados para las predicciones:

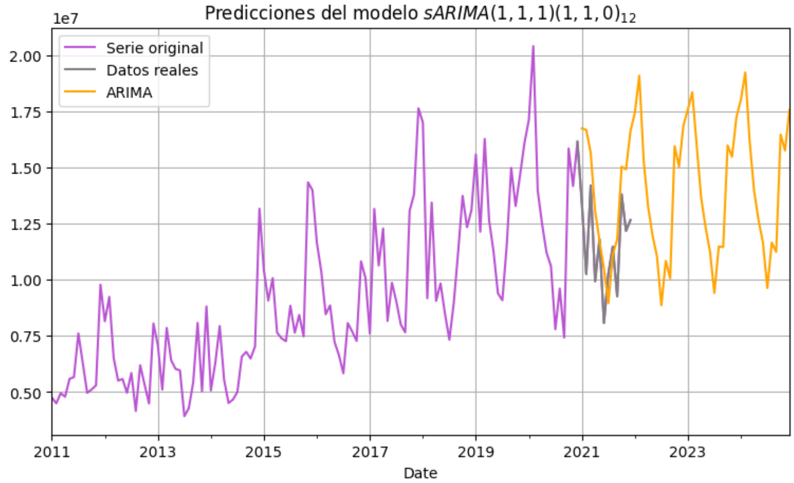


Figura 24: Predicciones del modelo sARIMA.

Como se puede observar, según el modelo la producción de energía del parque eólico seguirá una tendencia positiva, pero menor que la serie original. Además, la estacionalidad se conserva y, aunque la varianza no se ha estabilizado del todo, la heterocedasticidad de la serie generada por el modelo es menos pronunciada que en la figura 3.

Para las predicciones del primer año generadas por el modelo es posible realizar una comparación con los valores reales reservados fuera del *DataFrame* asignado a la variable *datos_ TR*. Tal y como se ha hecho en el apartado 4.1, los errores son los siguientes:

```
errores_sarima = abs(datos_rest['Power'][1:] - predicciones_statsmodels[0:12])
errores_absolutos = 100*errores_sarima/datos_rest['Power']
errores_absolutos = errores_absolutos.dropna()
df_errores_sarima = pd.DataFrame({'Fecha': errores_sarima.index,
                    'Error': errores_sarima,
                    'Error absoluto': errores_absolutos})
```

```
    df_errores_sarima.index = df_errores_sarima['Fecha']
    df_errores_sarima = df_errores_sarima.drop('Fecha', axis = 1)
    display(df_errores_sarima)
    plt.plot(errores_sarima*1e-6, label = 'Errores sARIMA', color = 'mediumorchid')
    plt.plot(errores*1e-6, label='Errores HW (logaritmos)', linestyle='-', color='gray')
    plt.plot(errores_log*1e-6, label='Errores HW', linestyle='-', color='black')
    plt.legend()
    plt.xticks(rotation=30, ha='right')
    plt.grid()
    plt.xlabel('Fecha')
    plt.ylabel(r'Error ($10^6$ MW)')
    plt.title('Errores en las predicciones del modelo HW')
    plt.show()
```

Listing 15: Modelo sARIMA.

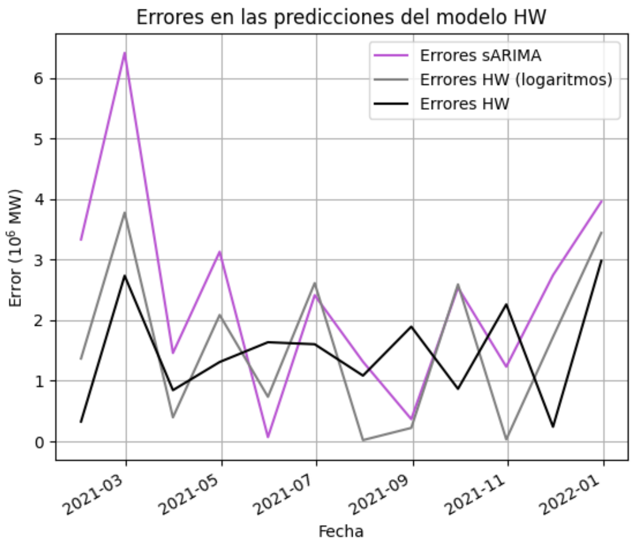


Figura 25: Comparación de errores entre los tres modelos desarrollados.

Se puede observar que, por lo general, la diferencia entre la predicción calculada por el modelo $sARIMA(1,1,1)(1,1,0)_{12}$ y los datos reales es mayor que aquella para los modelos de suavizado *Holt-Winters*. Además, el error es mayor sobre todo para los datos más próximos en el tiempo (las primeras predicciones) y parecen tener una tendencia a disminuir. Para comprobar esto sería necesario una serie temporal con más registros para poder utilizar un número adecuado de datos para entrenar los modelos y un número mayor para comparar con las predicciones. El valor numérico de las diferencias, así como su valor absoluto se puede observar en la siguiente tabla:

Fecha	Error (10^6 MW)	Error absoluto (%)
2021-01-31	3.3297	24.8416
2021-02-28	6.4110	62.5268
2021-03-31	1.4553	10.2563
2021-04-30	3.1276	31.4994
2021-05-31	0.0663	0.5621
2021-06-30	2.4095	29.8434
2021-07-31	1.3112	12.7702
2021-08-31	0.3635	3.1689
2021-09-30	2.5356	27.3779
2021-10-31	1.2290	8.9070
2021-11-30	2.7408	22.5182
2021-12-31	3.9545	31.2505

Tabla 6: Errores mensuales en millones de MW

Se pueden observar ciertos puntos en los que la precisión es aceptable. Sin embargo, la media de error en la predicción para estas doce observaciones es entre un 50 % y un 60 % mayor que en los casos anteriores:

Modelo HW	Modelo HW (log)	Modelo sARIMA
1.5805	1.4786	2.4112

Tabla 7: Comparación de la media de errores entre las predicciones de los modelos con respecto a los datos reales en 10^6 MW.

Por tanto se podría afirmar que, para esta serie, el modelo que mejor predice los datos es aquel que emplea el suavizado triple exponencial o de *Holt-Winters*.

5. Conclusiones

Tras el análisis de la serie temporal de producción de energía eólica, se han obtenido las siguientes conclusiones:

- La serie temporal presenta **tendencia creciente**, lo que indica un aumento en la producción de energía a lo largo del tiempo.
- Se ha identificado un **componente estacional** anual, con máximos en diciembre y mínimos en junio.
- La serie es **heterocedástica**, es decir, la varianza no es constante y aumenta con el tiempo.

En cuanto a los modelos de predicción desarrollados:

- Se han probado tres enfoques: **Holt-Winters**, **Holt-Winters con transformación logarítmica** y **sARIMA**.
- El modelo **Holt-Winters con logaritmos** ha obtenido los mejores resultados, presentando el menor error absoluto.
- El modelo **sARIMA** mostró un error significativamente mayor, especialmente en predicciones a corto plazo, lo que indica que no es la mejor opción para esta serie.

Como conclusión general, el modelo **Holt-Winters** es el más adecuado para predecir la producción de energía eólica en este caso, ya que sigue correctamente la tendencia y estacionalidad de la serie con un margen de error aceptable dentro de los tres casos posibles.

6. Anexo.

En esta sección se encuentran listados de código, tablas o imágenes que no son de gran relevancia para la práctica.

6.1. Anexo - Librerías y funciones.

Las librerías o funciones importadas para el estudio de los datos de la serie temporal se recogen en el siguiente listado.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from datetime import datetime
from statsmodels.tsa.seasonal import seasonal_decompose
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

Listing 16: Librerías y funciones.

6.2. Anexo - Gráficos.

Listado con el código para generar algunos de los gráficos vistos:

```
# FIGURA 5.
fig, ax1 = plt.subplots()
ax1.set_xlabel('Fecha')
ax1.set_ylabel('Coef. Estacionales', color = '#DDA0DD')
ax1.plot(resultado.seasonal, color = '#DDA0DD')
ax1.tick_params(axis='y', labelcolor = '#DDA0DD')
plt.xticks(rotation=30, ha='right')
ax2 = ax1.twinx()
ax2.set_ylabel('Residuos', color = '#000000')
ax2.plot(resultado.resid, color = '#000000')
ax2.tick_params(axis='y', labelcolor = '#000000')
plt.title('Coeficientes Estacionales y Residuos')
plt.grid()
plt.show()


# FIGURA 6.
datos['A o'] = pd.to_datetime(datos.index, format='%Y-%m-%d').year
sns.set_palette("Paired", 20)
plt.figure(figsize=(12, 8))
for a o, datos_a o in datos.groupby('A o'):
    plt.plot(datos_a o.index.month, datos_a o['Power'], label=str(a o))
plt.legend(title='A o')
plt.title('Gr fico estacional de energ a')
plt.xlabel('Mes')
plt.ylabel('Energ a producida')
plt.grid()
plt.show()

sumatorio_por_a o = datos.groupby('A o')['Power'].sum()
datos = datos.drop('A o', axis = 1)
plt.figure(figsize=(12, 8))
plt.plot(sumatorio_por_a o.index, sumatorio_por_a o, marker='o', color = 'mediumorchid
')
plt.title('Energ a producida por a o')
plt.xlabel('A o')
plt.ylabel('Energ a producida')
plt.grid()
plt.show()

# FIGURA 8.
serie_desestacionalizada = np.divide(datos['Power'].tolist(), list(resultado.seasonal))
```

```python
    plt.plot(datos.index, serie_desestacionalizada, color = 'mediumorchid')
    plt.grid()
    plt.title('Serie temporal desestacionalizada')
    plt.xlabel('Fecha')
    plt.ylabel(r'$Energ a/C_{estacional}$')
    plt.show()


    # FIGURA 11.
    plt.figure(figsize=(10, 6))
    plt.plot(datos_TR.index, datos_TR['Power'], label='Observados', linestyle='-', color='mediumorchid')
    plt.plot(datos_TR.index, modelo_holt_winters.fittedvalues, label='Suavizado Holt-Winters', linestyle='--', color='orange')
    plt.plot(datos_rest.index, datos_rest['Power'], label='Serie Real', linestyle='--', color='lightgray')
    plt.plot(predicciones_hw.index, predicciones_hw, label='Holt-Winters', linestyle='--', color='black')
    plt.xlabel('Fecha')
    plt.ylabel('Energ a producida (MW)')
    plt.title('Valores Observados, Futuros Reales y Predicci n para 1 a o')
    plt.legend()
    plt.xticks(rotation=30, ha='right')
    plt.grid()
    plt.show()

    # FIGURA 12.
    plt.figure(figsize=(10, 6))
    plt.plot(datos_TR.index, datos_TR['Power'], label='Observados', linestyle='-', color='mediumorchid')
    plt.plot(datos_rest.index, datos_rest['Power'], label='Serie Real', linestyle='--', color='lightgray')
    plt.plot(predicciones_hw.index, predicciones_hw, label='Holt-Winters', linestyle='--', color='black')
    plt.xlabel('Fecha')
    plt.ylabel('Energ a producida (MW)')
    plt.title('Valores Observados, Futuros Reales y Predicci n para 1 a o')
    plt.legend()
    plt.xticks(rotation=30, ha='right')
    plt.grid()
    plt.show()

    # FIGURA 13.
    plt.plot(errores*1e-6, label='errores', linestyle='-', color='darkorchid')
    plt.xticks(rotation=30, ha='right')
    plt.grid()
    plt.xlabel('Fecha')
    plt.ylabel(r'Error ($10^6$ MW)')
    plt.title('Errores en las predicciones del modelo HW')
    plt.show()

    # FIGURA 14.
    plt.figure(figsize=(10, 6))
    plt.plot(datos_TR.index, datos_TR['Power'], label='Observados', linestyle='-', color='mediumorchid')
    plt.plot(datos_rest.index, datos_rest['Power'], label='Serie Real', linestyle='--', color='lightgray')
    plt.plot(predicciones_log.index, predicciones_log, label='Holt-Winters_trans', linestyle='--',color='black')
    plt.plot(predicciones_hw.index, predicciones_hw, label='Holt-Winters', linestyle='--', color='yellow')
    plt.xlabel('Fecha')
    plt.ylabel('Energ a producida (MW)')
    plt.title('Valores Observados, Futuros Reales y Predicci n para 1 a o')
    plt.legend()
    plt.xticks(rotation=30, ha='right')
    plt.show()

    # FIGURA 16.
    plt.plot(errores_log*1e-6, label='Errores (logaritmos)', linestyle='-', color='gray')
```

```python
    plt.plot(errores*1e-6, label='Errores', linestyle='-', color='darkorchid')
    plt.legend()
    plt.xticks(rotation=30, ha='right')
    plt.grid()
    plt.xlabel('Fecha')
    plt.ylabel(r'Error ($10^6$ MW)')
    plt.title('Errores en las predicciones del modelo HW')
    plt.show()

    # FIGURA 19.
    fig, axs=plt.subplots(nrows=2,ncols=1,figsize=(10,8),sharex=True)

    plot_acf(datos_TR, ax=axs[0], lags=48, alpha=0.05, color = 'mediumorchid', vlines_kwargs={"colors": 'mediumorchid'})
    axs[0].set_title('ACF - Serie original')
    for item in axs[0].collections:
        if type(item)==PolyCollection:
            item.set_facecolor('mediumorchid')


    plot_acf(datos_diff_1, ax=axs[1], lags=48, alpha=0.05, color = 'mediumorchid', vlines_kwargs={"colors": 'mediumorchid'})
    axs[1].set_title('ACF - Serie diferenciada')
    for item in axs[1].collections:
        if type(item)==PolyCollection:
            item.set_facecolor('mediumorchid')

    # FIGURA 20.
    fig, axs=plt.subplots(nrows=2,ncols=1,figsize=(10,8),sharex=True)

    plot_acf(datos_diff_1, ax=axs[0], lags=48, alpha=0.05, color = 'mediumorchid', vlines_kwargs={"colors": 'mediumorchid'})
    axs[0].set_title('ACF - Serie diferenciada')
    for item in axs[0].collections:
        if type(item)==PolyCollection:
            item.set_facecolor('mediumorchid')

    plot_pacf(datos_diff_1, ax=axs[1], lags=48, alpha=0.05, color = 'mediumorchid', vlines_kwargs={"colors": 'mediumorchid'})
    axs[1].set_title('PACF - Serie diferenciada')
    for item in axs[1].collections:
        if type(item)==PolyCollection:
            item.set_facecolor('mediumorchid')

    # FIGURA 22:
    fig, axs = plt.subplots(nrows=2, ncols=1, figsize=(10, 8), sharex=True)
    plt.grid()

    plot_acf(datos_diff_1_12, ax=axs[0], lags=46, alpha=0.05, color = 'mediumorchid', vlines_kwargs={"colors": 'mediumorchid'})
    axs[0].set_title('Autocorrelaci n serie diferenciada de orden XII')
    for item in axs[0].collections:
        if type(item)==PolyCollection:
            item.set_facecolor('mediumorchid')

    plot_pacf(datos_diff_1_12, ax=axs[1], lags=46, alpha=0.05, color = 'mediumorchid', vlines_kwargs={"colors": 'mediumorchid'})
    axs[1].set_title('Autocorrelaci n parcial serie diferenciada de orden XII')
    for item in axs[1].collections:
        if type(item)==PolyCollection:
            item.set_facecolor('mediumorchid')
    axs[0].grid()
```

Listing 17: Código para generar los distintos gráficos.