MINERÍA DE DATOS Y MODELIZACIÓN PREDICTIVA SERIES TEMPORALES Y MODELO ARIMA

IPC HOSTELERÍA Y TURISMO EN CANARIAS

Jacobo Álvarez Gutiérrez

En el presente trabajo se ha seleccionado la serie temporal correspondiente al **Índice de Precios de Consumo (IPC) de Hostelería y Turismo en Canarias**, debido a la relevancia de este sector en la economía del archipiélago.

El **Índice de Precios de Consumo (IPC)** es un indicador económico que mide la evolución de los precios de una cesta de bienes y servicios representativa del consumo de los hogares. Dentro de este índice, se encuentran diferentes categorías que reflejan la variación de precios en sectores específicos, como la alimentación, el transporte o la hostelería y el turismo.

La serie temporal analizada en este estudio tiene una frecuencia **mensual** y toma como **año base 2021** (índice 100). Esto significa que los valores reflejan la evolución de los precios en comparación con el nivel medio de precios en 2021. Por ejemplo, si el índice para **enero de 2025** es de **123,47**, esto indica que los precios en el sector han aumentado un **23,47%** respecto al nivel de 2021.

Esta serie permite analizar la tendencia de los precios en un sector clave para la economía canaria, caracterizado por su marcada **estacionalidad** debido a la fuerte influencia del turismo en la región.

La fuente de datos es el siguiente enlace: https://www.ine.es/consul/serie.do?d=true&s=IPC260788&c=2&

In [2]:
```
# Librerías necesarias
import os
import warnings
```

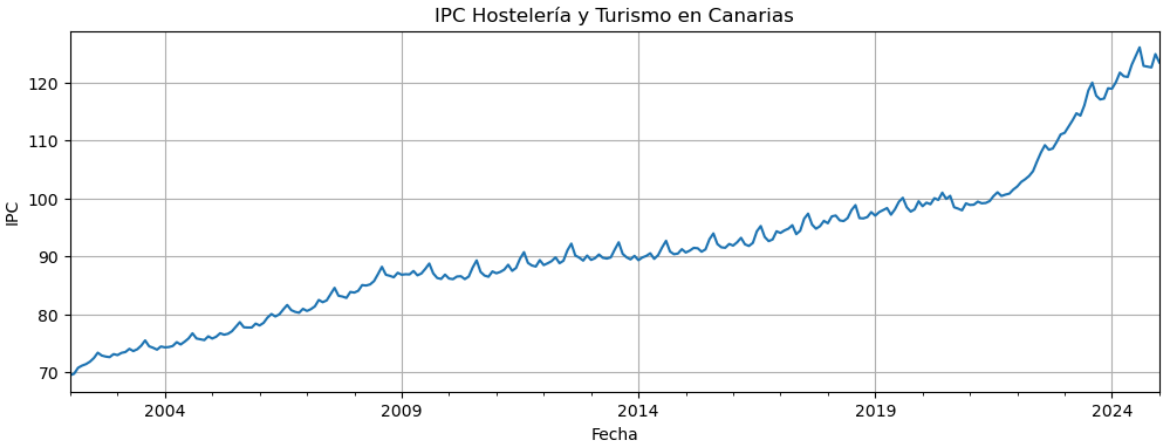
```
import pandas as pd
import numpy as np
import datetime as dt
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
from tabulate import tabulate
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing, Holt
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import pmdarima as pm
```

In [3]:
```
# Definimos directorio
os.chdir(r'C:\Users\jacob\Desktop\master_ucm\mod7_3_mineria_modpredic\Tarea')

# Para solucionar diversas advertencias
warnings.filterwarnings("ignore", category=UserWarning, module="openpyxl")
warnings.filterwarnings("ignore", category=UserWarning, module='statsmodels')

# Importación de los datos
datos = pd.read_excel('ipc_hosteleria&turismo_canarias.xlsx', skiprows = 6) # La
datos = datos[['PERIODO', 'VALOR']] # Solo nos interesan esas columnas de datos

# Damos el formato correcto a la fecha
datos['PERIODO'] = datos['PERIODO'].str.replace('M', '-').str.strip()
datos['PERIODO'] = pd.to_datetime(datos['PERIODO'], format='%Y-%m')

# Usamos la fecha como índice de los registros
datos = datos.set_index('PERIODO')
datos = datos.sort_index()
datos.head()
```

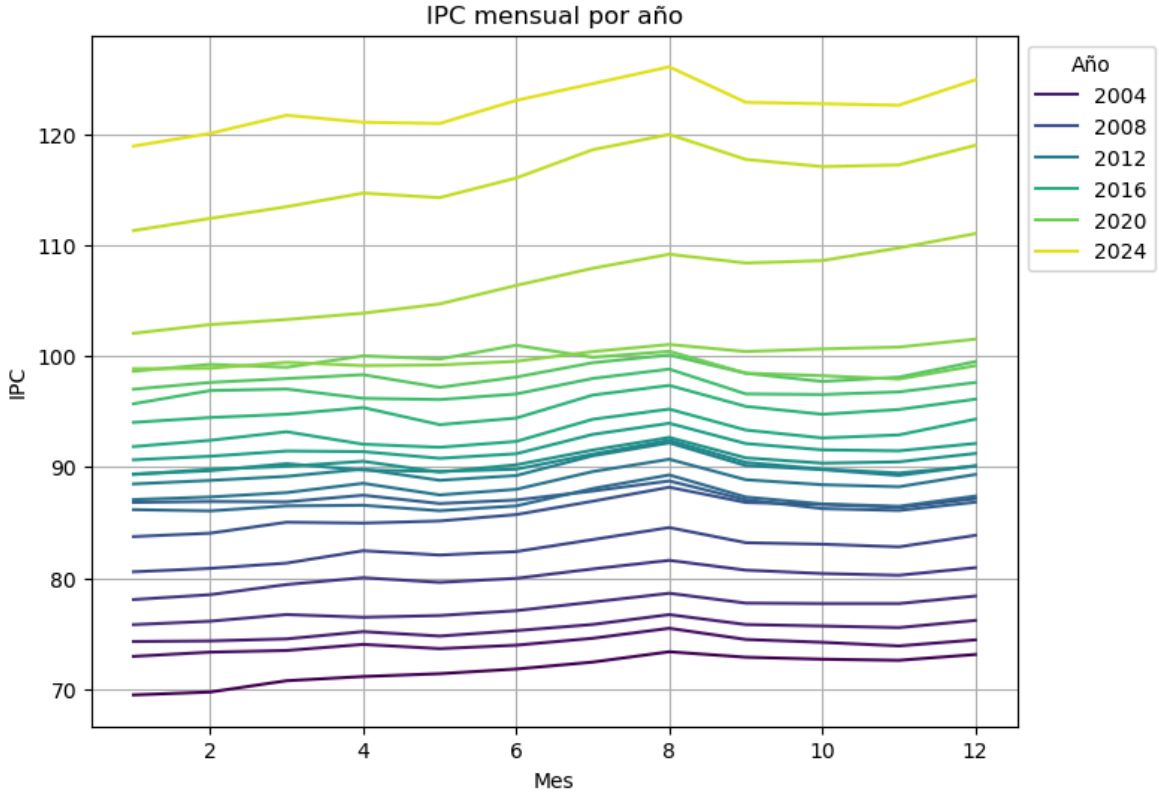
Out[3]:

PERIODO	VALOR
2002-01-01	69.466
2002-02-01	69.724
2002-03-01	70.745
2002-04-01	71.123
2002-05-01	71.380

In [4]:
```
# Separamos los valores numéricos del IPC para tratarlos como un objeto de tipo
serie = datos['VALOR']

# Representamos la serie
serie.plot(figsize= (12,4))
plt.title('IPC Hostelería y Turismo en Canarias')
plt.xlabel('Fecha')
plt.ylabel('IPC')
plt.grid()
plt.show()
```



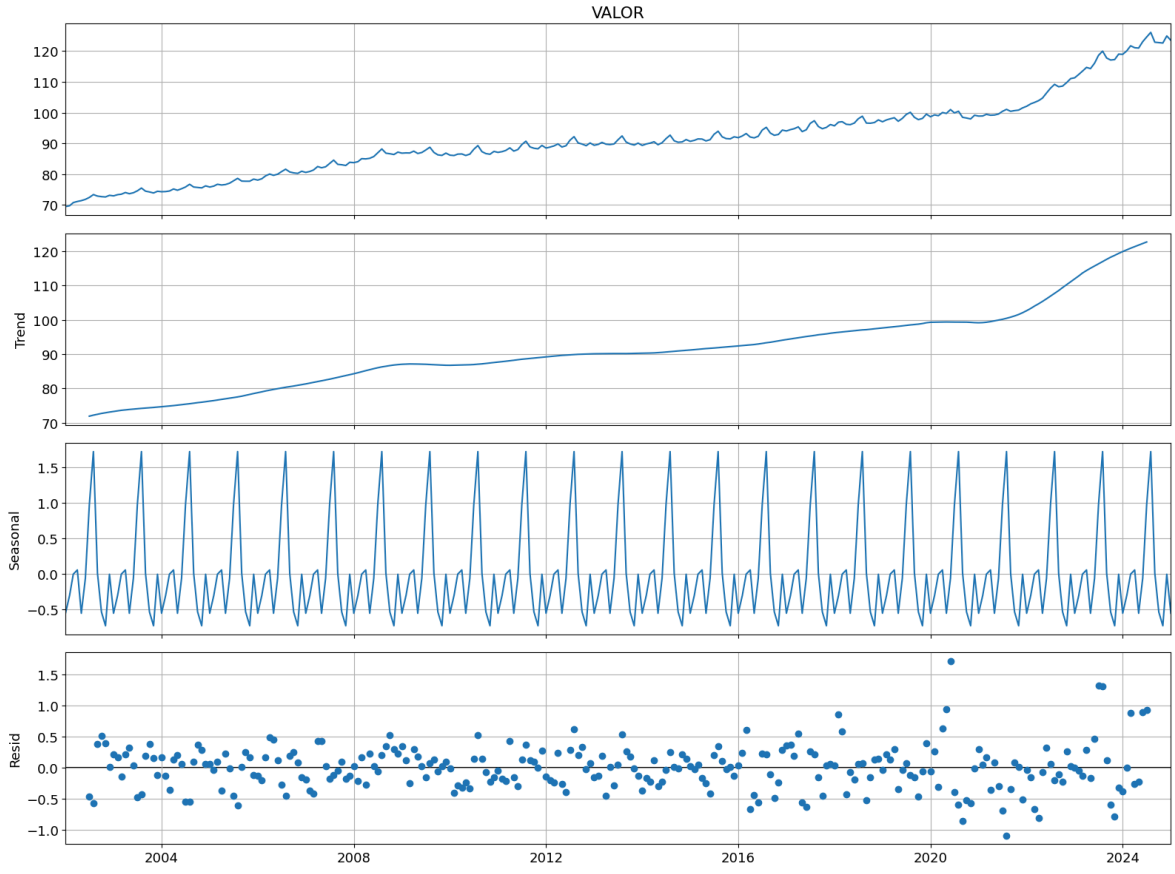
In [5]:
```
# Representación de cada año de forma independiente
datos['AÑO'] =datos.index.year

plt.figure(figsize=(8, 6))
sns.lineplot(x=datos.index.month, y=datos.VALOR,
             hue = datos['AÑO'], palette='viridis')
plt.xlabel('Mes')
plt.ylabel('IPC')
plt.title('IPC mensual por año')
plt.legend(title='Año',loc='upper left', bbox_to_anchor=(1, 1))
plt.grid()
plt.show()
```



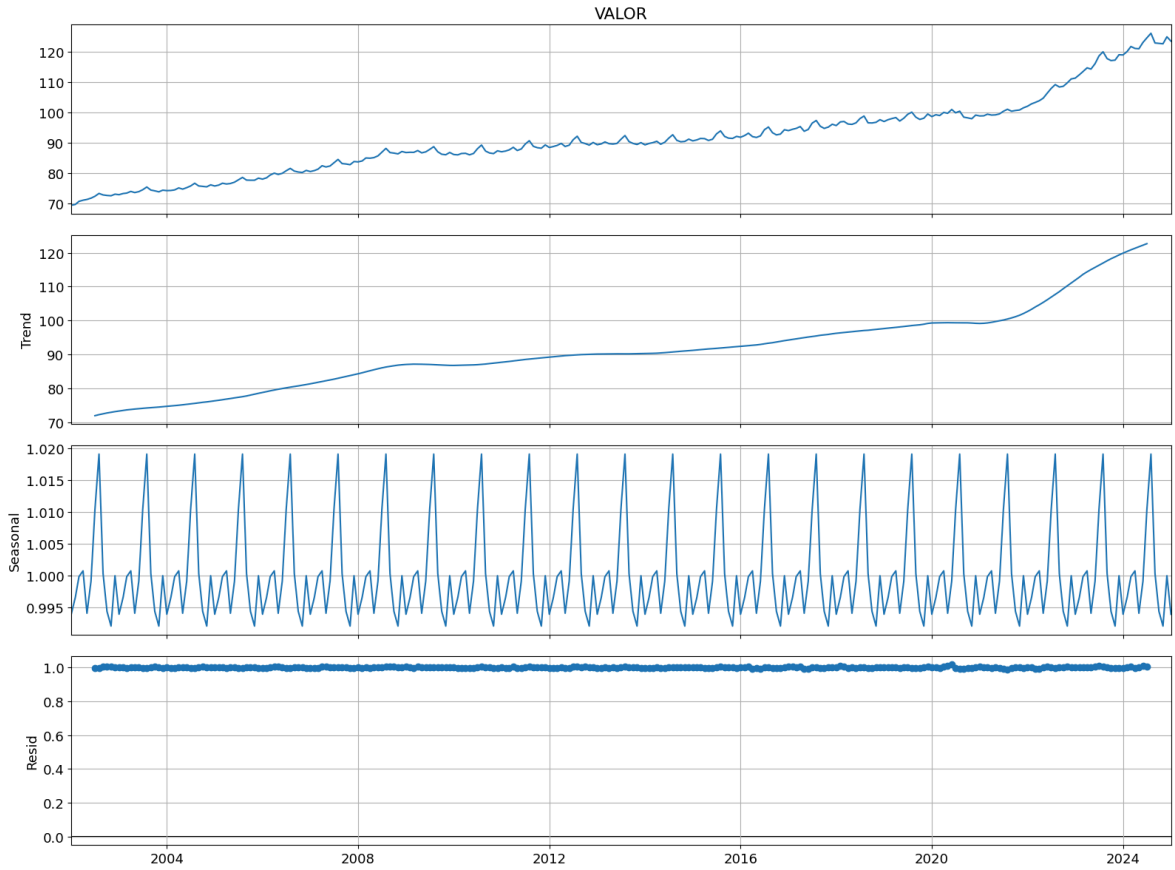
Tras observar la representación gráfica de la serie anterior, vemos que existe una cierta tendencia al alza en el IPC a lo largo de los últimos años. No obstante, se confirma también algo que ya augurábamos por tratarse de un indicativo del sector turístico, y es una cierta recursividad en las fluctuaciones del valor del IPC, probablemente asociadas a las temporadas altas y bajas de turismo en el archipiélago.

Además, nótese que el IPC nunca adopta valores nulos o negativos, lo cual es un indicio de que el modelo multiplicativo se adaptará mejor a esta serie. Independientemente, realizaremos la descomposición estacional con ambos tipos de modelización (aditiva y multiplicativa) para ver cual describe mejor la serie.

In [7]:
```
# Descomposición estacional de la serie (Modelo Aditivo)
add_decomposition = seasonal_decompose(serie, model = 'aditive', period = 12)
plt.rc("figure", figsize=(16, 12))
plt.rc("font", size=13)
fig = add_decomposition.plot()
for ax in fig.axes:
    ax.grid()
plt.show()
```



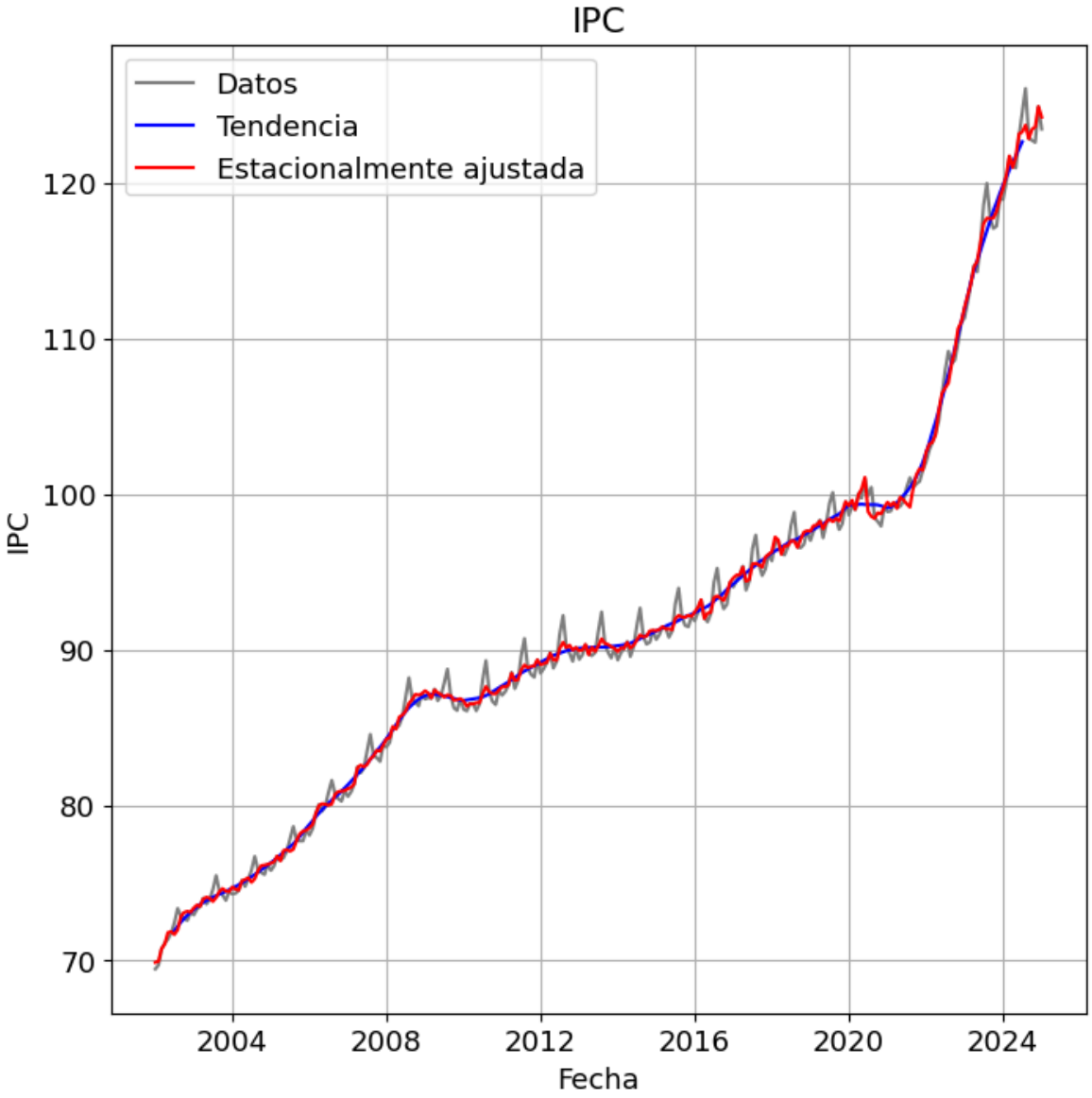
In [8]:
```
# Descomposición estacional de la serie (Modelo Multiplicativo)
mult_decomposition = seasonal_decompose(serie, model = 'multiplicative', period
plt.rc("figure", figsize=(16, 12))
plt.rc("font", size=13)
fig = mult_decomposition.plot()
for ax in fig.axes:
    ax.grid()
plt.show()
```



Tal y como se puede observar de las dos descomposiciones estacionales obtenidas, vemos que la amplitud de la estacionalidad parece ser independiente de esa tendencia al alza observada. Esto podría dar a intuir que un modelo aditivo sería más adecuado. Sin embargo, nótese que los residuos son prácticamente insignificantes en el modelo multiplicativo, así que parece que este último se ajusta mejor a la hora de describir las características generales de la serie objeto de estudio. Será por tanto este último el que tomaremos como referencia para la posterior generación de modelos predictivos.

Aún así, antes de pasar a los modelos de suavizado exponencial, vamos a realizar una representación gráfica de la serie temporal y la serie temporal pero estacionalmente ajustada para hacer una comparación entre ellas.

In [10]:
```
# Comparación serie y serie estacionalmente ajustada
serie_ajustada_estacionalmente=serie/mult_decomposition.seasonal
# Graficar la serie original y las componentes
plt.figure(figsize=(8, 8))
# Serie original
plt.plot(serie, label='Datos', color='gray')
# Tendencia
plt.plot(mult_decomposition.trend, label='Tendencia', color='blue')
# Estacionalmente ajustada
plt.plot(serie_ajustada_estacionalmente, label='Estacionalmente ajustada', color
plt.xlabel('Fecha')
plt.ylabel('IPC')
plt.title('IPC')
plt.legend()
plt.grid()
plt.show()
```



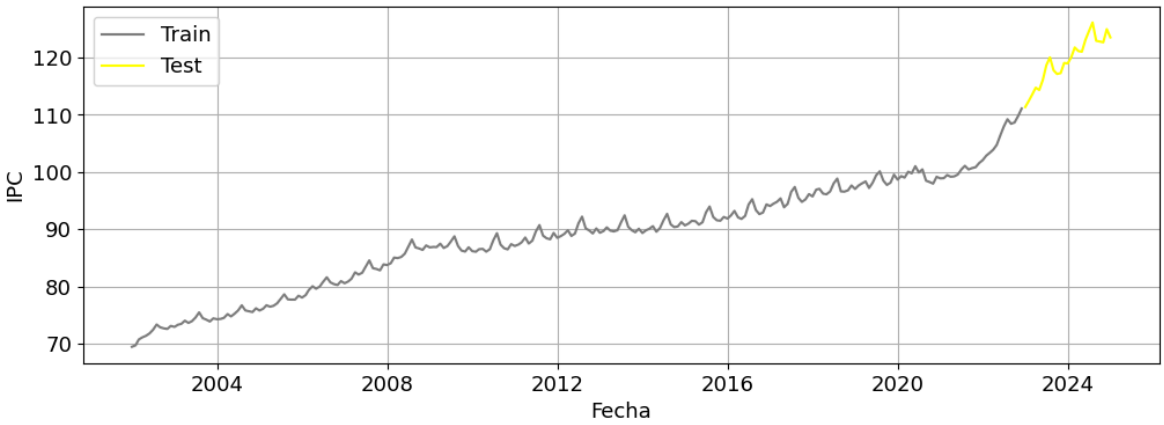
Vemos claramente que, al corregir los efectos de estacionalidad a la serie original, esta sigue casi de manera perfecta la tendencia global observada. Esto refleja que la estacionalidad es el principal factor que añade variabilidad a la serie original.

Con todo el estudio realizado hasta ahora, podemos concluir que la estacionalidad juega un papel clave en la variabilidad del IPC, aunque la tendencia a largo plazo sigue un crecimiento sostenido (más acentuado después de 2020, lo cual puede deberse a efectos económicos recientes como la inflación post-pandemia o el simple aumento de la demanda turística). Además, la serie ajustada estacionalmente sigue muy de cerca a la tendencia global, lo cual es un indicio de que la descomposición ha funcionado bien. En resumen, parece que el modelo multiplicativo ha capturado la dinámica esencial de esta serie temporal.

Llegados a este punto podemos empezar con la implementación de modelos predictivos de suavizado exponencial. No obstante, previo a esto, vamos a dividir la serie en unos datos de entrenamiento y otros datos test para la posterior comprobación de la efectividad predictiva de los modelos. Estos datos test no se tomarán de forma aleatoria debido a la influencia del factor cronológico en la serie. En su lugar, los datos test se corresponderán con las lecturas de los dos últimos años (2023 y 2024), así como la del

primer mes de este año 2025. Esto se hace así procurando mantener una proporción aproximada del 90% de los datos como entrenamiento y un 10% de estos como test.

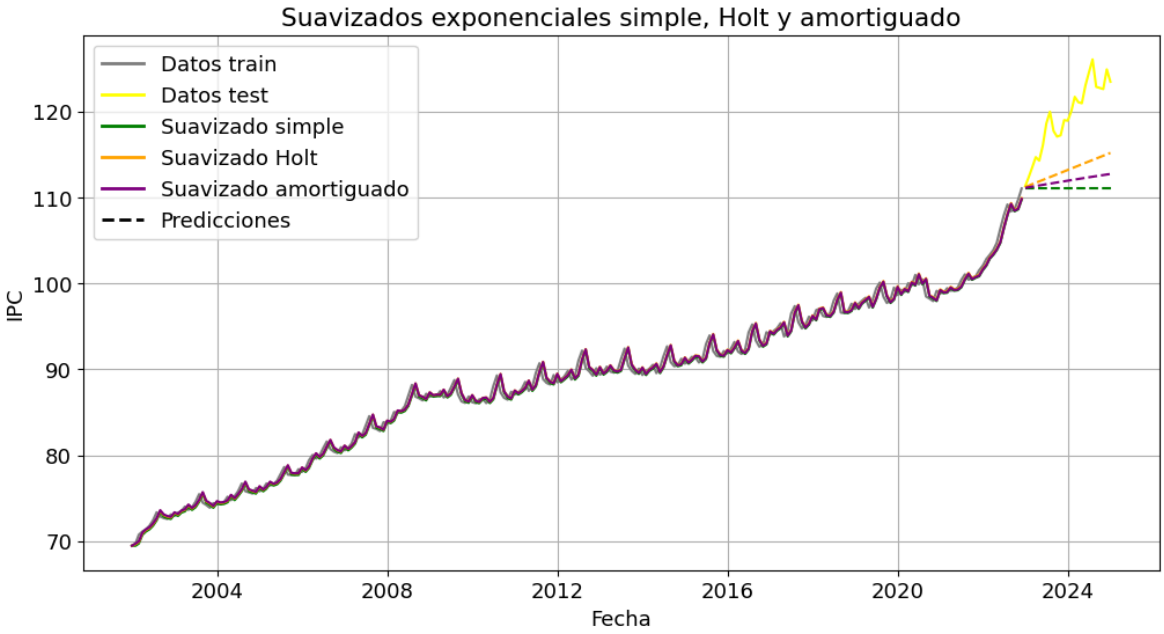
In [12]:
```
# Partición de los datos originales en train y test
serie_test = serie[-25:]
serie_train = serie[:-25]

plt.figure(figsize=(12, 4))
plt.plot(serie_train, label='Train', color='gray')
plt.plot(serie_test, label='Test', color='yellow')
plt.legend()
plt.xlabel('Fecha')
plt.ylabel('IPC')
plt.grid()
plt.show()
```



Tal y como comentábamos, empezaremos a continuación con la implementación de los distintos modelos de suavizado exponencial que se han trabajado a lo largo de la teoría. Esto comprende el método de alisado simple, el método de alisado doble de Holt, el método de tendencia amortiguada y el método Holt-Winters. Además, este último método es el más apropiado para series temporales con estacionalidad (como es este caso), por lo que auguramos que será el método más idóneo para este ejemplo. Representaremos conjuntamente el resultado de la aplicación de los tres primeros métodos y, posteriormente, la del método de Holt-Winters.

In [14]:
```
# Método de alisado simple
modelo_simple = SimpleExpSmoothing(serie_train, initialization_method = 'estimat
fcast_simple = modelo_simple.forecast(25)

# Método de Holt
modelo_holt = Holt(serie_train, initialization_method = 'estimated').fit()
fcast_holt = modelo_holt.forecast(25)

# Método de tendencia amortiguada
modelo_amort = Holt(serie_train, damped_trend = True, initialization_method = 'e
fcast_amort = modelo_amort.forecast(25)

plt.figure(figsize = (12, 6))

plt.plot(serie_train, label = 'Datos-train', color = 'gray')
plt.plot(serie_test, label = 'Datos-test', color = 'yellow')

plt.plot(modelo_simple.fittedvalues, label = 'Suavizado simple', color = 'green'
```

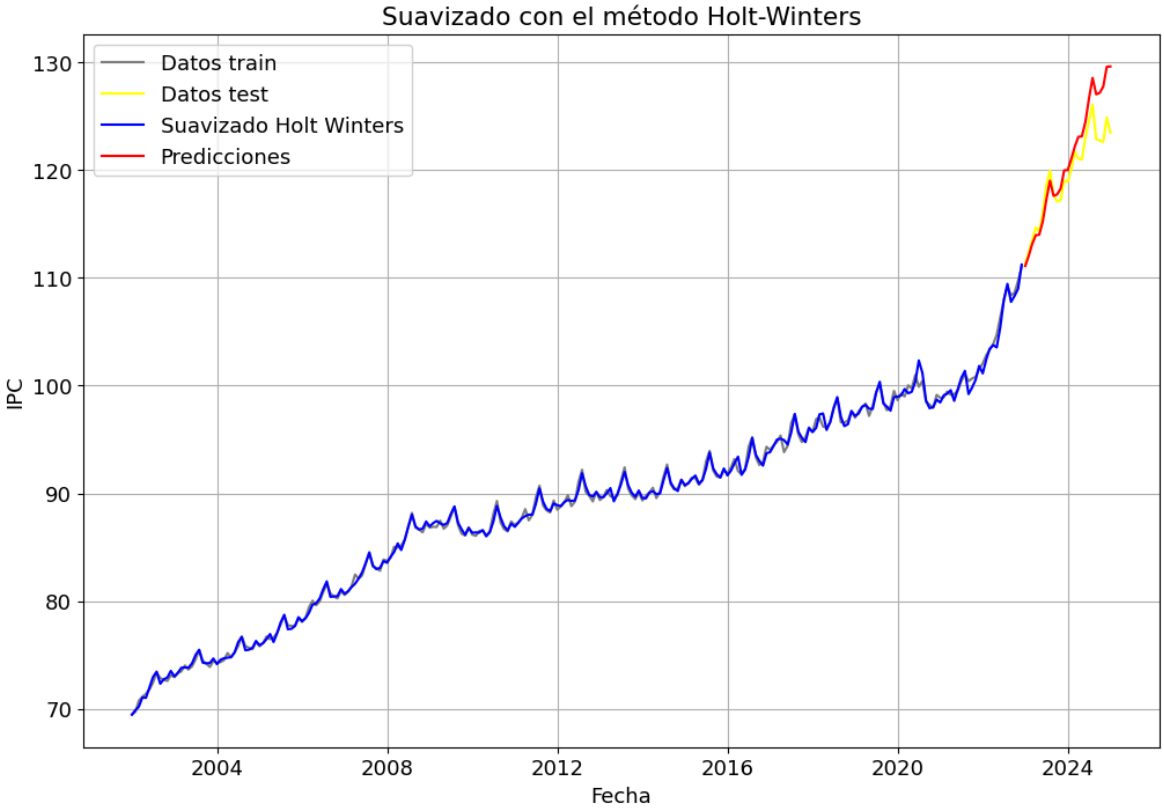
```
plt.plot(modelo_holt.fittedvalues, label = 'Suavizado Holt', color = 'orange')
plt.plot(modelo_amort.fittedvalues, label = 'Suavizado amortiguado', color = 'pu

plt.plot(fcast_simple, color = 'green', linestyle = '--')
plt.plot(fcast_holt, color = 'orange', linestyle = '--')
plt.plot(fcast_amort, color = 'purple', linestyle = '--')

plt.xlabel('Fecha')
plt.ylabel('IPC')
plt.title('Suavizados exponenciales simple, Holt y amortiguado')

# Personalizar la leyenda con líneas manuales
legend_elements = [
    Line2D([0], [0], color="gray", lw=2, label="Datos train"),
    Line2D([0], [0], color="yellow", lw=2, label="Datos test"),
    Line2D([0], [0], color="green", lw=2, label="Suavizado simple"),
    Line2D([0], [0], color="orange", lw=2, label="Suavizado Holt"),
    Line2D([0], [0], color="purple", lw=2, label="Suavizado amortiguado"),
    Line2D([0], [0], color="black", linestyle="--", lw=2, label="Predicciones")
]

# Agregar la leyenda personalizada
plt.legend(handles=legend_elements, loc="best")
plt.grid()
plt.show()
```



Vemos que los suavizados prácticamente se superponen entre sí y, además, guardando una relación estricta con los valores de la serie original. Sin embargo, en lo que respecta a la capacidad predictiva de los modelos generados, vemos que ninguno de ellos se ajusta adecuadamente a los registros reales. A continuación, vamos a implementar el método de Holt-Winters, el cuál resulta especialmente útil cuando la serie presenta estacionalidad. Veamos el resultado de dicha aplicación:

In [16]:
```
# Método de Holt-Winters
modelo_holt_winters = ExponentialSmoothing(serie_train, seasonal_periods = 12, t
                                           initialization_method = 'estimated').
fcast_holt_winters = modelo_holt_winters.forecast(25)
```

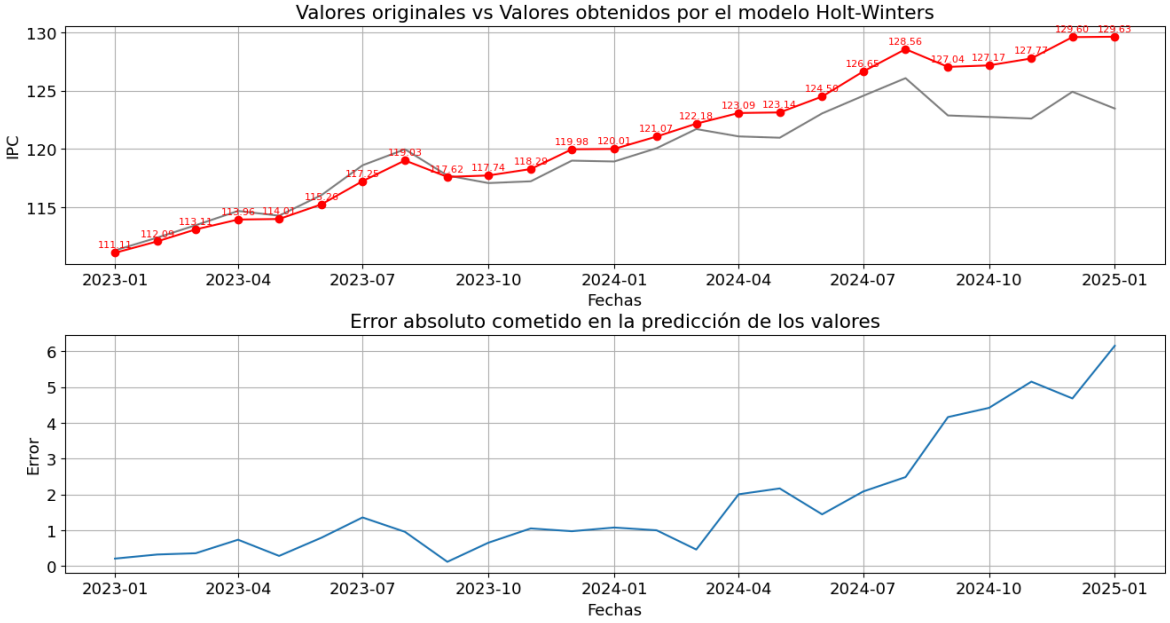
```
plt.figure(figsize = (12, 8))
plt.plot(serie_train, label = 'Datos train', color = 'gray')
plt.plot(serie_test, label = 'Datos test', color = 'yellow')
plt.plot(modelo_holt_winters.fittedvalues, label = 'Suavizado Holt Winters', col
plt.plot(fcast_holt_winters, label = 'Predicciones', color = 'red')
plt.xlabel('Fecha')
plt.ylabel('IPC')
plt.title('Suavizado con el método Holt-Winters')
plt.legend()
plt.grid()
plt.show()
```



Con este último método de suavizado (Holt-Winters) vemos que la capacidad predictiva si es significativa. Para valorar esta bondad de ajuste vamos a hacer una representación gráfica que haga explícita la diferencia entre los valores originales y los que ha obtenido el modelo:

In [18]:
```
fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(16, 8))
axes[0].plot(serie_test, label = 'Datos originales', color = 'gray')
axes[0].plot(fcast_holt_winters, label = 'Datos modelados', color = 'red', marke
for i, txt in enumerate(fcast_holt_winters):
    axes[0].annotate(f"{txt:.2f}",
                     (fcast_holt_winters.index[i], fcast_holt_winters.iloc[i]),
                     textcoords="offset points", xytext=(0, 4),  # 🔹 Separació
                     ha='center', fontsize=8, color="red")  # 🔹 Reducimos tama
axes[0].set_xlabel('Fechas')
axes[0].set_ylabel('IPC')
axes[0].set_title('Valores originales vs Valores obtenidos por el modelo Holt-Wi
axes[0].grid()
axes[1].plot(abs(serie_test-fcast_holt_winters), label = 'Error')
axes[1].set_xlabel('Fechas')
axes[1].set_ylabel('Error')
axes[1].set_title('Error absoluto cometido en la predicción de los valores')
```

```
axes[1].grid()
plt.subplots_adjust(hspace=0.3)
```



Recordamos de teoría que este modelo se obtiene aplicando las siguientes fórmulas:

$$L_t = \alpha \frac{x_t}{S_{t-s}} + (1-\alpha)(L_{t-1} + b_{t-1})$$

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

$$S_t = \gamma \frac{x_t}{(L_{t-1} + b_{t-1})} + (1-\gamma)S_{t-s}$$

$$\hat{x}_{t+h} = (L_t + b_t h)S_{t-s+h}$$

Los resultados obtenidos para cada uno de los parámetros se observa explícitamente en la siguiente tabla:

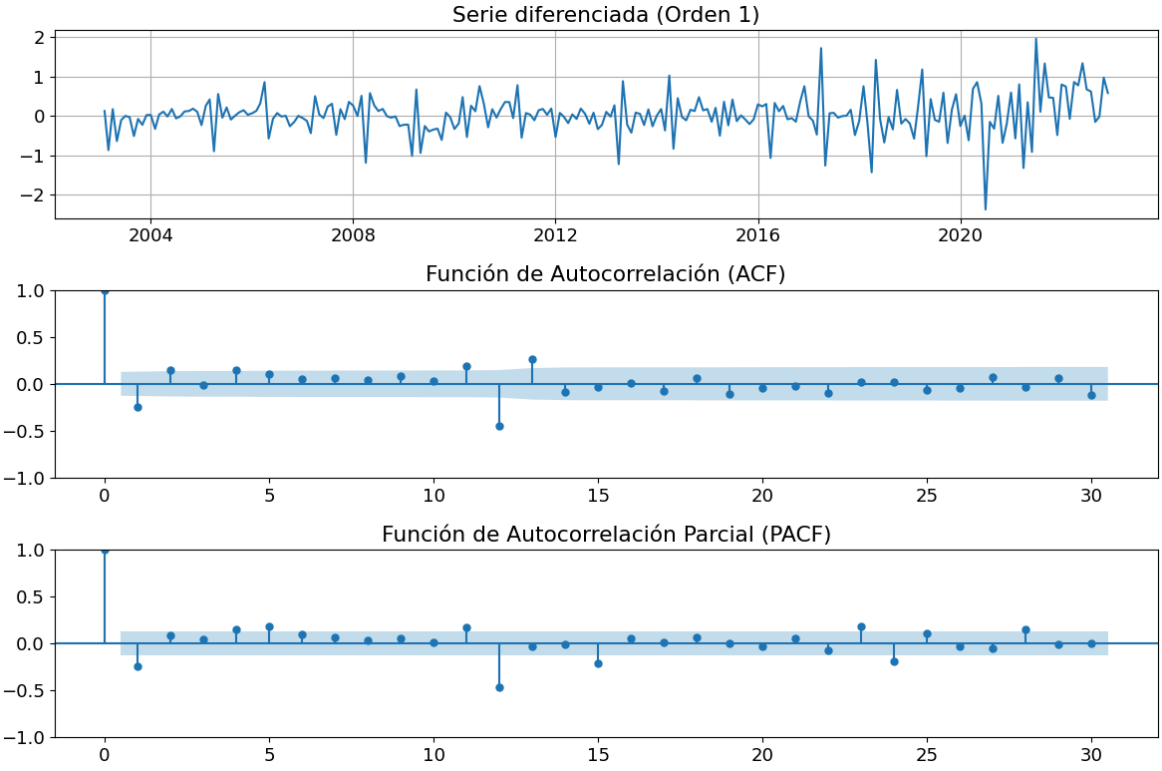
In [20]:
```
# Parámetros del modelo Holt-Winters
headers = ['Nombre', 'Parámetro', 'Valor', 'Optimizado']
table_str = tabulate(modelo_holt_winters.params_formatted, headers, tablefmt = '
print(table_str)
```

Nombre	Parámetro	Valor	Optimizado
smoothing_level	alpha	0.824273	True
smoothing_trend	beta	0.120093	True
smoothing_seasonal	gamma	0.0260372	True
initial_level	l.0	63.7368	True
initial_trend	b.0	1.00392	True
initial_seasons.0	s.0	1.08545	True
initial_seasons.1	s.1	1.08784	True
initial_seasons.2	s.2	1.09065	True
initial_seasons.3	s.3	1.09198	True
initial_seasons.4	s.4	1.08551	True
initial_seasons.5	s.5	1.09043	True
initial_seasons.6	s.6	1.10201	True
initial_seasons.7	s.7	1.11157	True
initial_seasons.8	s.8	1.09139	True
initial_seasons.9	s.9	1.08575	True
initial_seasons.10	s.10	1.08378	True
initial_seasons.11	s.11	1.09214	True

A continuación, nos proponemos el objetivo de elaborar un modelo ARIMA ajustado manualmente. Para ello, puesto que ya conocemos la serie y sabemos que existe una cierta tendencia y una estacionalidad, vamos a tener que diferenciarla al menos un orden para intentar hacerla estacionaria. Así, el primer paso será diferenciar la serie y representarla junto con los correlogramas asociados.

In [22]:
```
# Serie con diferenciación regular (para corregir tendencia)
serie_diff = serie_train.diff().dropna()
serie_seasonal_diff = serie_diff.diff(12).dropna()

# Plot de la serie y los correlogramas
fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize = (12, 8))
ax1.plot(serie_seasonal_diff)
ax1.set_title('Serie diferenciada (Orden 1)')
ax1.grid()
plot_acf(serie_seasonal_diff, lags = 30, ax = ax2)
ax2.set_title('Función de Autocorrelación (ACF)')
plot_pacf(serie_seasonal_diff, lags = 30, ax = ax3)
ax3.set_title('Función de Autocorrelación Parcial (PACF)')
```

```
plt.tight_layout()
plt.show()
```



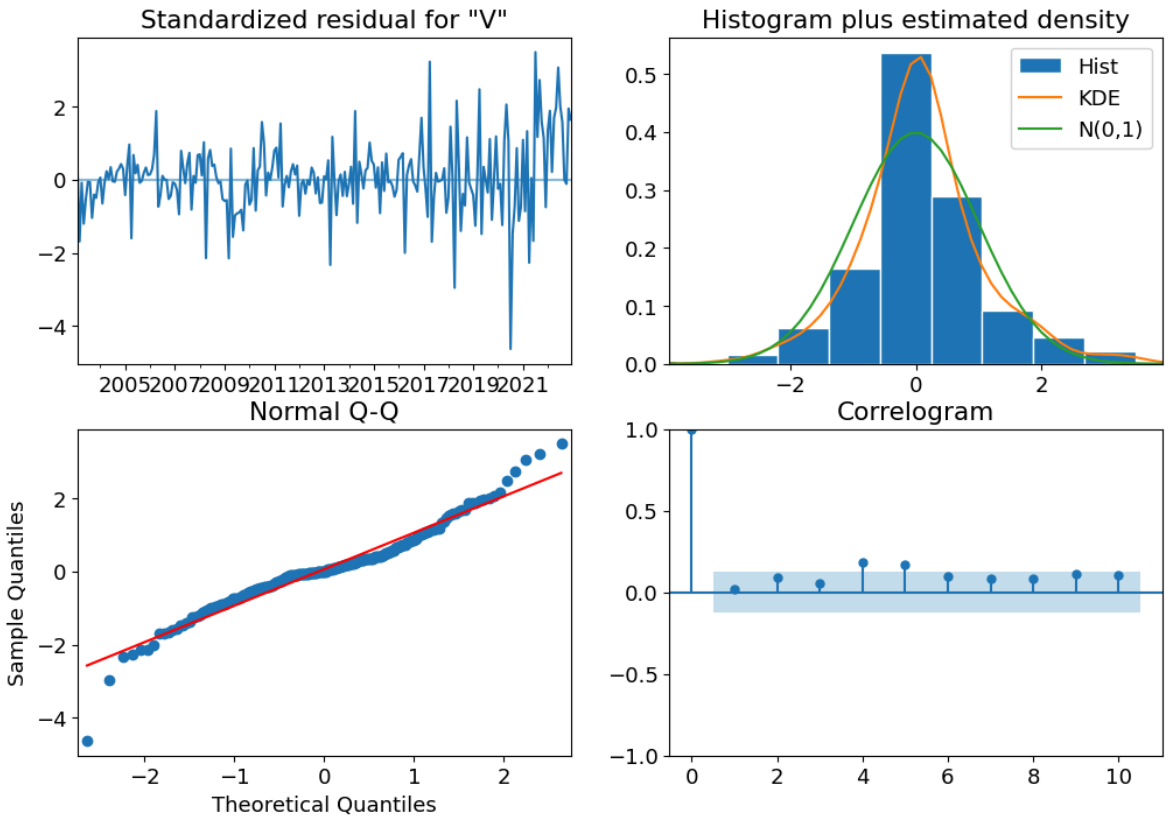
Tras haber hecho una diferenciación regular y otra estacional, vemos que la serie adopta una versión mucho más estacionaria que la original. Por otra parte, vemos que el primer punto en PACF y el 12 están fuera de la región donde se admite su nulidad, así que parece relevante tener en cuenta un factor autoregresivo de orden 1 ($p = 1$). Esto sugiere una elección de parámetros tal que $ARIMA(1,1,0)(0,1,0)_{12}$. Siguiendo la metodología Box-Jenkins, vamos a aplicar dicho modelo para evaluar sus residuos y valorar la bondad de ajuste del mismo.

In [24]:
```
# Generación del modelo
modelo_arima = sm.tsa.ARIMA(serie_train, order = (1,1,0), seasonal_order = (0,1,
resultados = modelo_arima.fit()
print(resultados.summary())
```

```
                                SARIMAX Results
==========================================================================================
Dep. Variable:                              VALOR   No. Observations:
252
Model:             ARIMA(1, 1, 0)x(0, 1, 0, 12)   Log Likelihood                -
172.737
Date:                          Tue, 25 Feb 2025   AIC
349.473
Time:                                  10:52:36   BIC
356.426
Sample:                              01-01-2002   HQIC
352.275
                                   - 12-01-2022
Covariance Type:                             opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.2431      0.049     -4.967      0.000      -0.339      -0.147
sigma2         0.2484      0.015     16.613      0.000       0.219       0.278
===================================================================================
==
Ljung-Box (L1) (Q):                   0.07   Jarque-Bera (JB):                77.
66
Prob(Q):                              0.79   Prob(JB):                         0.
00
Heteroskedasticity (H):               3.99   Skew:                            -0.
07
Prob(H) (two-sided):                  0.00   Kurtosis:                         5.
79
===================================================================================
==

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-st
ep).
```

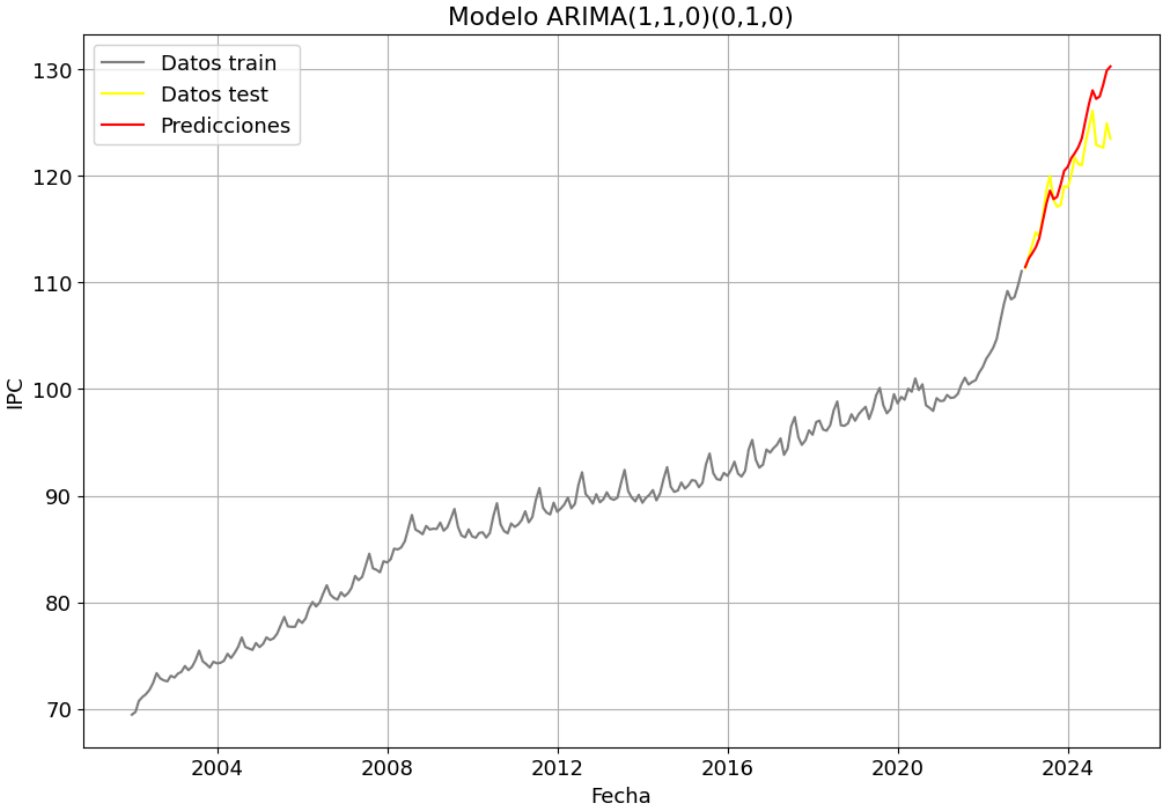
Nótese que el p-valor asociado al estadístico de Ljung-Box es 0.79, que es significativamente mayor que 0.05. Esto ya es una evidencia significativa de que los residuos del modelo están incorrelados. No obstante, sabemos que para hacer una correcta diagnosis del modelo no basta con esto, sino que debemos verificar que los residuos se comportan como un ruido blanco. Hagamos entonces un análisis un poco más exhaustivo de los residuos.

In [26]:
```
# Análisis de los residuos del modelo anterior
resultados.plot_diagnostics(figsize = (12,8))
plt.show()
```



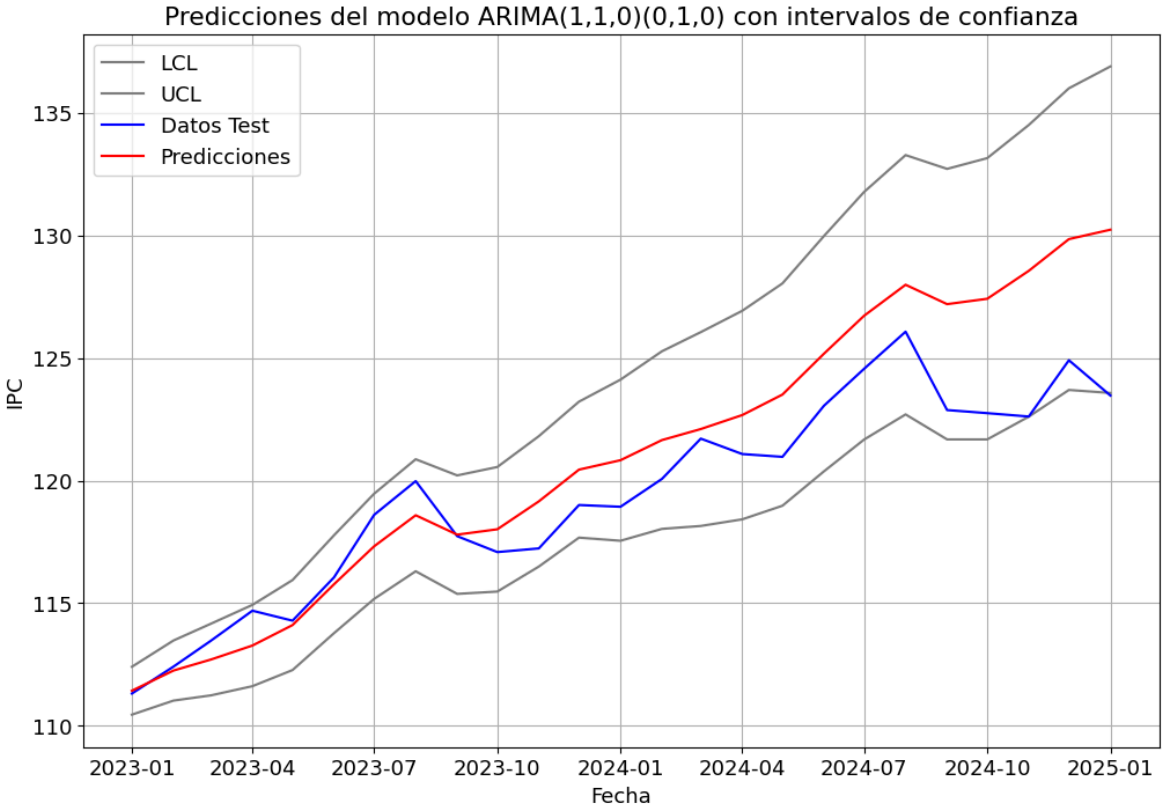
Verificamos con las gráficas anteriores que los residuos siguen aproximadamente una distribución normal con un valor medio próximo a cero, una varianza relativamente constante y además están incorrelados, así que, efectivamente, parece que podemos empezar a realizar alguna predicción con este modelo.

In [28]:
```
# Predicciones del modelo ARIMA(1,1,0)(0,1,0)

predicciones = resultados.get_forecast(steps = 25)
predi_test = predicciones.predicted_mean
plt.figure(figsize = (12,8))
plt.plot(serie_train, label = 'Datos train', color = 'gray')
plt.plot(serie_test, label = 'Datos test', color = 'yellow')
plt.plot(predi_test, label = 'Predicciones', color = 'red')
plt.xlabel('Fecha')
plt.ylabel('IPC')
plt.title('Modelo ARIMA(1,1,0)(0,1,0)')
plt.legend()
plt.grid()
plt.show()
```



Vamos a efectuar un análisis correspondiente solo a la parte de los datos test y las predicciones. Concretamente, vamos a representar estos valores en los márgenes de un intervalo de confianza del 95% para las predicciones.

In [30]:
```
# Análisis predicciones con intervalos de confianza
intervalos_confianza = predicciones.conf_int()
plt.figure(figsize = (12,8))
plt.plot(intervalos_confianza['lower VALOR'], label = 'LCL', color = 'gray')
plt.plot(intervalos_confianza['upper VALOR'], label = 'UCL', color = 'gray')
plt.plot(serie_test, label = 'Datos Test', color = 'blue')
plt.plot(predi_test, label = 'Predicciones', color = 'red')
plt.xlabel('Fecha')
plt.ylabel('IPC')
plt.title('Predicciones del modelo ARIMA(1,1,0)(0,1,0) con intervalos de confian
plt.legend()
plt.grid()
plt.show()
```



En esta última gráfica se puede observar que las predicciones obtenidas con el modelo $ARIMA(1,1,0)(0,1,0)_{12}$ se ajustan de manera aproximada a los datos reales a lo largo de todo el periodo del conjunto Test. Además, nótese que las desviaciones entre predicciones y valores reales son tanto por exceso como por defecto, lo cual podría sugerir que no se está cometiendo algún error sistemático en la modelización. Por último, obsérvese que los intervalos de confianza cubren adecuadamente la mayoría de los valores reales y las predicciones, lo cual es un indicio de que el modelo captura de manera razonable la incertidumbre en las predicciones.

Es importante destacar el hecho de que la partición del conjunto de datos en datos Train y datos Test tiene lugar justamente en la transición de los efectos de la COVID-19, lo cual añade cierto error inevitable al modelado.

Finalmente, para concluir con este proyecto, vamos a implementar un modelo ARIMA con ajuste automático y a analizar el resultado obtenido de manera similar a como se ha hecho con este de ajuste manual.

In [32]:
```
# Modelo ARIMA con ajuste automatizado
modelo_auto = pm.auto_arima(serie_train, start_p=0, start_q=0, max_p=3, max_q=3,
                            m=12, start_P=0, start_Q=0, max_P=2, max_Q=2, seasona
                            error_action='ignore', suppress_warnings=True, stepwi
print(modelo_auto.summary())
```

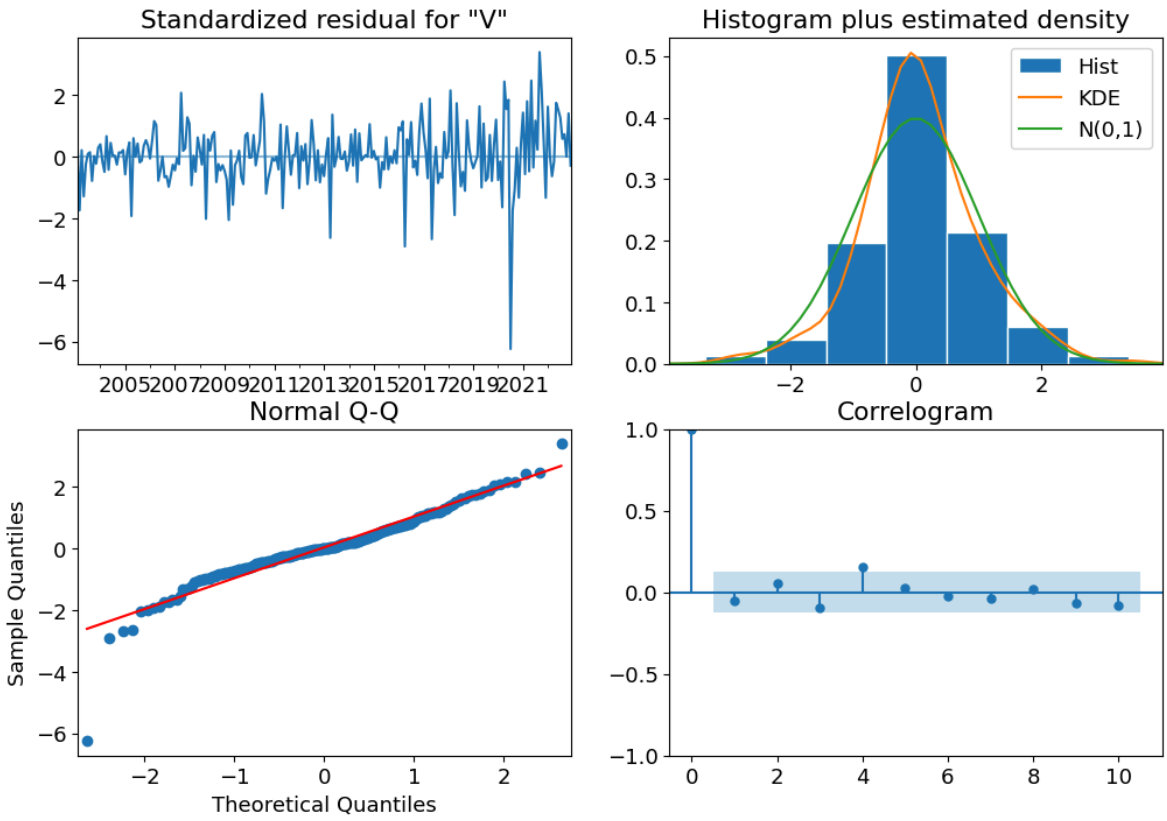
```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,1,0)[12]             : AIC=362.019, Time=0.03 sec
 ARIMA(1,1,0)(1,1,0)[12]             : AIC=302.780, Time=0.13 sec
 ARIMA(0,1,1)(0,1,1)[12]             : AIC=284.937, Time=0.20 sec
 ARIMA(0,1,1)(0,1,0)[12]             : AIC=352.421, Time=0.04 sec
 ARIMA(0,1,1)(1,1,1)[12]             : AIC=286.339, Time=0.21 sec
 ARIMA(0,1,1)(0,1,2)[12]             : AIC=286.287, Time=0.48 sec
 ARIMA(0,1,1)(1,1,0)[12]             : AIC=303.723, Time=0.09 sec
 ARIMA(0,1,1)(1,1,2)[12]             : AIC=287.942, Time=0.84 sec
 ARIMA(0,1,0)(0,1,1)[12]             : AIC=284.056, Time=0.14 sec
 ARIMA(0,1,0)(1,1,1)[12]             : AIC=285.194, Time=0.20 sec
 ARIMA(0,1,0)(0,1,2)[12]             : AIC=285.123, Time=0.26 sec
 ARIMA(0,1,0)(1,1,0)[12]             : AIC=304.466, Time=0.08 sec
 ARIMA(0,1,0)(1,1,2)[12]             : AIC=287.080, Time=0.63 sec
 ARIMA(1,1,0)(0,1,1)[12]             : AIC=284.699, Time=0.21 sec
 ARIMA(1,1,1)(0,1,1)[12]             : AIC=283.483, Time=0.30 sec
 ARIMA(1,1,1)(0,1,0)[12]             : AIC=349.828, Time=0.06 sec
 ARIMA(1,1,1)(1,1,1)[12]             : AIC=284.484, Time=0.43 sec
 ARIMA(1,1,1)(0,1,2)[12]             : AIC=284.397, Time=0.64 sec
 ARIMA(1,1,1)(1,1,0)[12]             : AIC=299.653, Time=0.19 sec
 ARIMA(1,1,1)(1,1,2)[12]             : AIC=286.916, Time=1.41 sec
 ARIMA(2,1,1)(0,1,1)[12]             : AIC=271.625, Time=0.45 sec
 ARIMA(2,1,1)(0,1,0)[12]             : AIC=340.691, Time=0.11 sec
 ARIMA(2,1,1)(1,1,1)[12]             : AIC=272.387, Time=0.55 sec
 ARIMA(2,1,1)(0,1,2)[12]             : AIC=272.411, Time=0.93 sec
 ARIMA(2,1,1)(1,1,0)[12]             : AIC=284.484, Time=0.31 sec
 ARIMA(2,1,1)(1,1,2)[12]             : AIC=274.371, Time=1.88 sec
 ARIMA(2,1,0)(0,1,1)[12]             : AIC=284.237, Time=0.21 sec
 ARIMA(3,1,1)(0,1,1)[12]             : AIC=272.127, Time=0.77 sec
 ARIMA(2,1,2)(0,1,1)[12]             : AIC=256.793, Time=1.11 sec
 ARIMA(2,1,2)(0,1,0)[12]             : AIC=328.289, Time=0.26 sec
 ARIMA(2,1,2)(1,1,1)[12]             : AIC=275.250, Time=0.77 sec
 ARIMA(2,1,2)(0,1,2)[12]             : AIC=275.253, Time=1.09 sec
 ARIMA(2,1,2)(1,1,0)[12]             : AIC=274.512, Time=0.71 sec
 ARIMA(2,1,2)(1,1,2)[12]             : AIC=258.457, Time=2.64 sec
 ARIMA(1,1,2)(0,1,1)[12]             : AIC=266.825, Time=0.50 sec
 ARIMA(3,1,2)(0,1,1)[12]             : AIC=275.092, Time=0.77 sec
 ARIMA(2,1,3)(0,1,1)[12]             : AIC=inf, Time=1.38 sec
 ARIMA(1,1,3)(0,1,1)[12]             : AIC=263.372, Time=0.58 sec
 ARIMA(3,1,3)(0,1,1)[12]             : AIC=inf, Time=1.49 sec
 ARIMA(2,1,2)(0,1,1)[12] intercept   : AIC=261.285, Time=1.30 sec

Best model:  ARIMA(2,1,2)(0,1,1)[12]
Total fit time: 24.407 seconds
                                      SARIMAX Results
==========================================================================================
===========
Dep. Variable:                                     y   No. Observations:
252
Model:             SARIMAX(2, 1, 2)x(0, 1, [1], 12)   Log Likelihood
-122.396
Date:                              Tue, 25 Feb 2025   AIC
256.793
Time:                                      10:53:01   BIC
277.651
Sample:                                  01-01-2002   HQIC
265.198
                                       - 12-01-2022
Covariance Type:                                 opg
==========================================================================================
```

```
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.6569      0.076     21.802      0.000       1.508       1.806
ar.L2         -0.6934      0.081     -8.529      0.000      -0.853      -0.534
ma.L1         -1.8116      0.051    -35.787      0.000      -1.911      -1.712
ma.L2          0.9060      0.051     17.867      0.000       0.807       1.005
ma.S.L12      -0.6528      0.054    -12.072      0.000      -0.759      -0.547
sigma2         0.1582      0.009     17.648      0.000       0.141       0.176
===================================================================================
==
Ljung-Box (L1) (Q):                   0.65   Jarque-Bera (JB):               439.
56
Prob(Q):                              0.42   Prob(JB):                         0.
00
Heteroskedasticity (H):               3.51   Skew:                            -0.
89
Prob(H) (two-sided):                  0.00   Kurtosis:                         9.
40
===================================================================================
==

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-st
ep).
```

Como vemos, parece que el mejor modelo ARIMA que ha encontrado la función auto_arima es aquel que se corresponde con la siguiente elección de parámetros: $ARIMA(2,1,2)(0,1,1)_{12}$. Nótese, que todos los p-valores asociados a los distintos coeficientes del modelo son inferiores a 0.05, lo cual es un indicativo de que todos los parámetros son significativos. Además, atendiendo al valor del p-valor para el estadístico de Ljung-Box, vemos que los residuos están incorrelados.

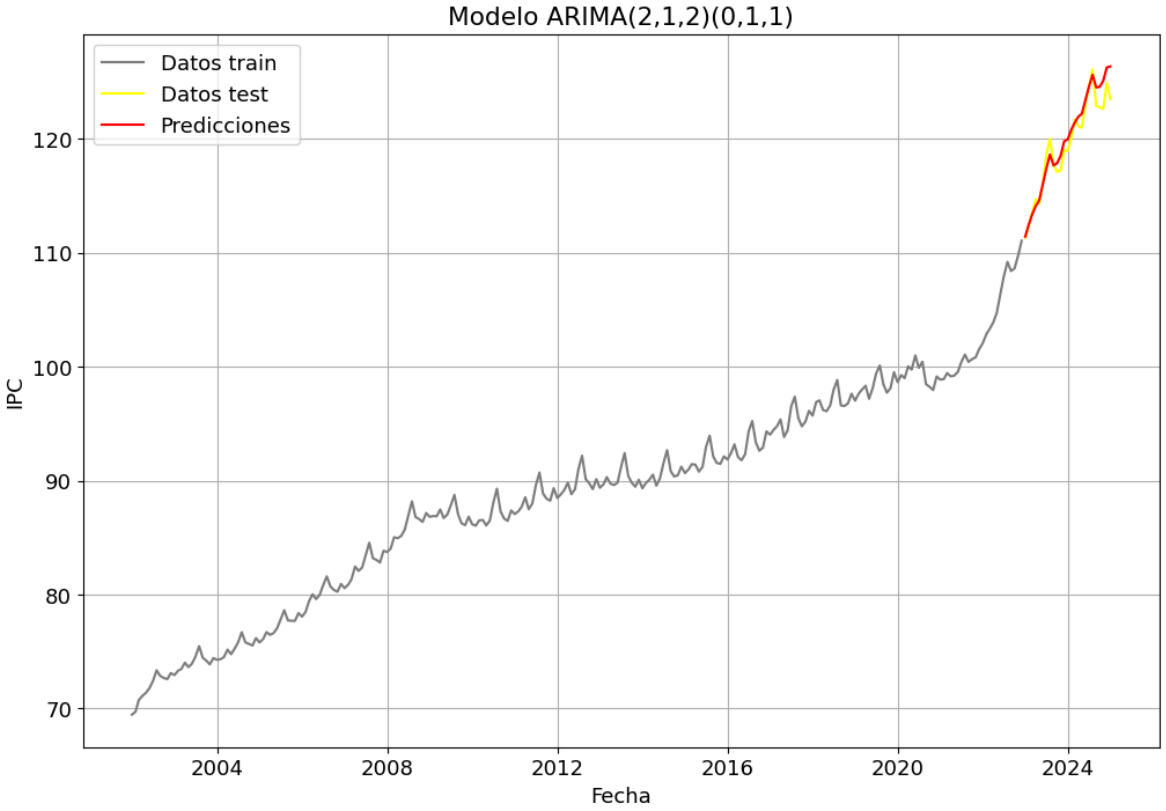
A continuación, de manera análoga a como se procedió con el modelo manual, vamos a representar de forma explícita algunas características adicionales de los residuos, así como las predicciones obtenidas con este modelo. Además, se hará una comparación entre estas predicciones y las obtenidas con el modelo manual.

In [34]:
```
# Correlación residuos
best_arima = sm.tsa.ARIMA(serie_train, order = (2,1,2), seasonal_order = (0,1,1,
resultados_auto = best_arima.fit()
resultados_auto.plot_diagnostics(figsize = (12,8))
plt.show()
```

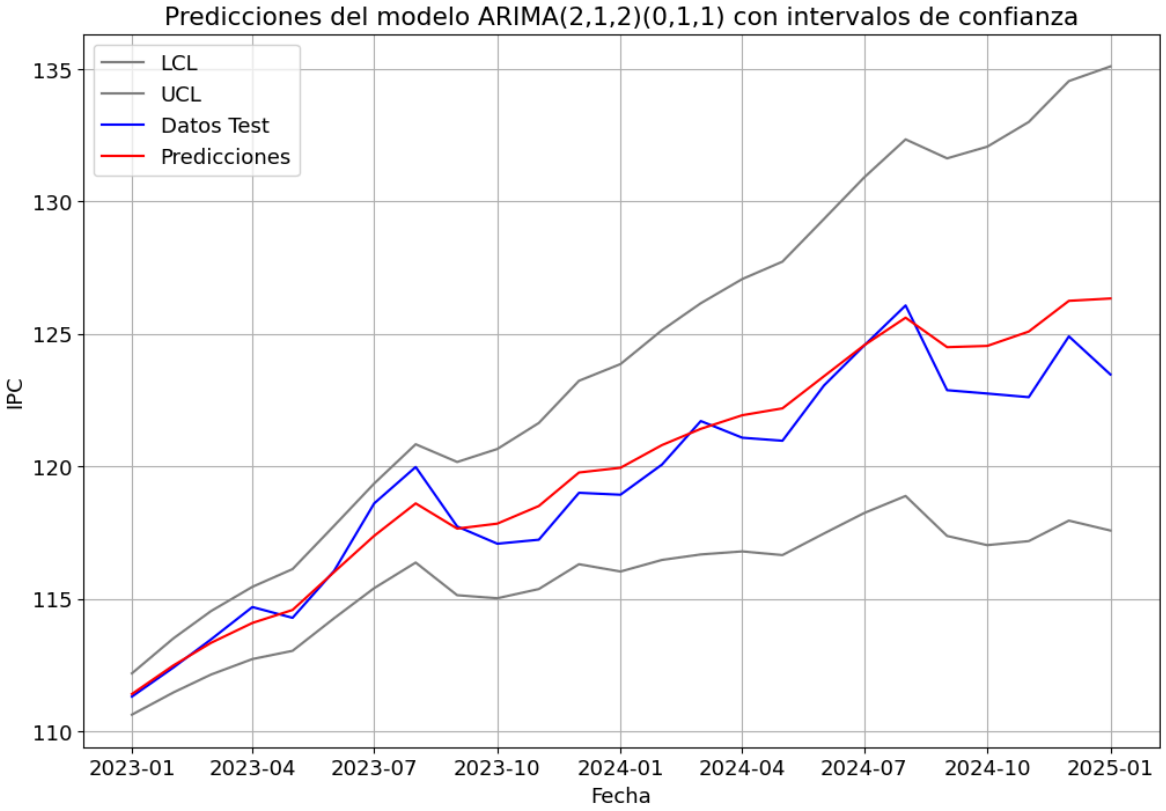


Comprobamos con los gráficos anteriores que los residuos del modelo no solo están incorrelados, sino que además se aproximan a una distribución normal con media cero y desviación aproximadamente constante.

In [36]:
```
# Predicciones modelo auto-ARIMA
predicciones_auto = resultados_auto.get_forecast(steps = 25)
predi_test_auto = predicciones_auto.predicted_mean
plt.figure(figsize = (12,8))
plt.plot(serie_train, label = 'Datos train', color = 'gray')
plt.plot(serie_test, label = 'Datos test', color = 'yellow')
plt.plot(predi_test_auto, label = 'Predicciones', color = 'red')
plt.xlabel('Fecha')
plt.ylabel('IPC')
plt.title('Modelo ARIMA(2,1,2)(0,1,1)')
plt.legend()
plt.grid()
plt.show()
```

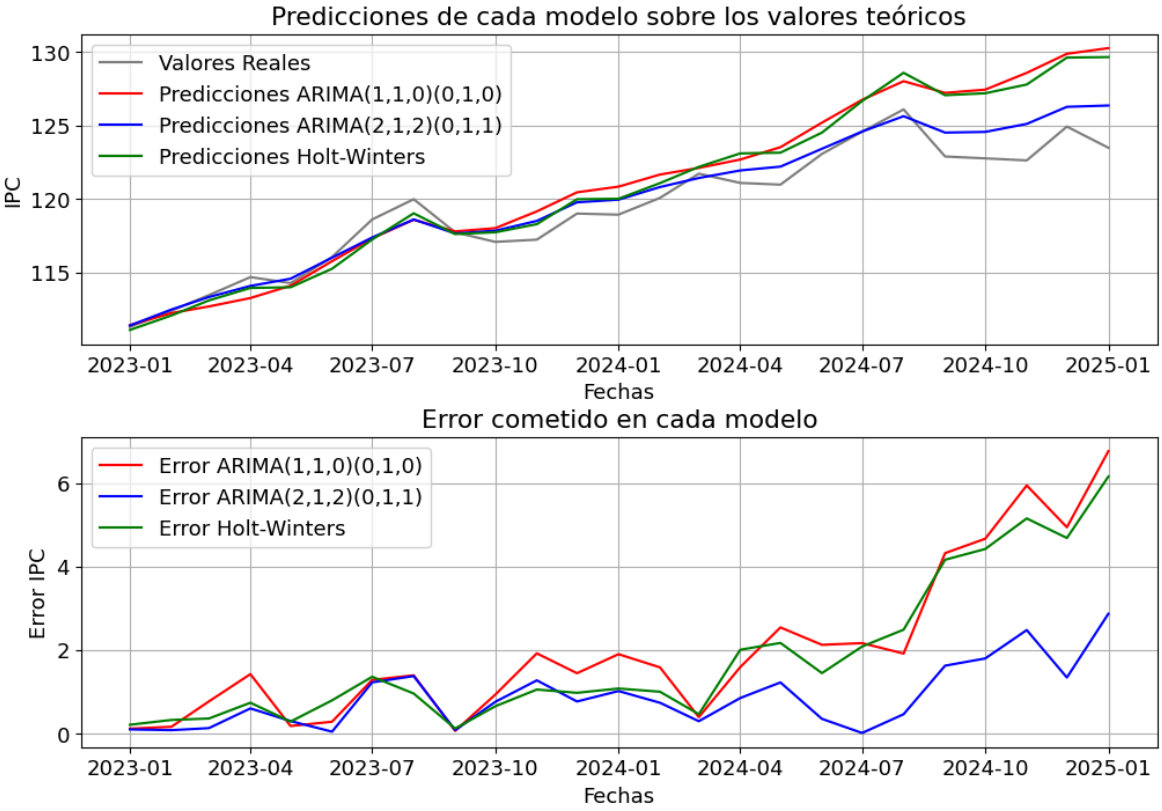


In [37]:
```
# Análisis de predicciones del modelo auto-ARIMA con intervalos de confianza
intervalos_confianza_auto = predicciones_auto.conf_int()
plt.figure(figsize = (12,8))
plt.plot(intervalos_confianza_auto['lower VALOR'], label = 'LCL', color = 'gray'
plt.plot(intervalos_confianza_auto['upper VALOR'], label = 'UCL', color = 'gray'
plt.plot(serie_test, label = 'Datos Test', color = 'blue')
plt.plot(predi_test_auto, label = 'Predicciones', color = 'red')
plt.xlabel('Fecha')
plt.ylabel('IPC')
plt.title('Predicciones del modelo ARIMA(2,1,2)(0,1,1) con intervalos de confian
plt.legend()
plt.grid()
plt.show()
```



Con los dos gráficos anteriores ya tenemos suficientes indicativos como para poder asegurar que este modelo auto-ARIMA de coeficientes $ARIMA(2,1,2)(0,1,1)_{12}$ es significativamente mejor que el obtenido de forma manual. Esto lo podemos argumentar viendo la cercanía de las predicciones con los valores teóricos. No obstante, vamos a realizar un análisis comparativo explícito entre las predicciones de estos dos modelos ARIMA y el modelo obtenido con el método de suavizado de Holt-Winters para elaborar una conclusión final que determine el modelo idóneo para la predicción de esta serie.

In [39]:
```
# Análisis comparativo de los modelos suavizado Holt-Winters, ARIMA(1,1,0)(0,1,0
fig, (ax1, ax2) = plt.subplots(2, 1, figsize = (12, 8))
ax1.plot(serie_test, label = 'Valores Reales', color = 'gray')
ax1.plot(predi_test, label = 'Predicciones ARIMA(1,1,0)(0,1,0)', color = 'red')
ax1.plot(predi_test_auto, label = 'Predicciones ARIMA(2,1,2)(0,1,1)', color = 'b
ax1.plot(fcast_holt_winters, label = 'Predicciones Holt-Winters', color = 'green
ax1.set_title('Predicciones de cada modelo sobre los valores teóricos')
ax1.legend()
ax1.set_xlabel('Fechas')
ax1.set_ylabel('IPC')
ax1.grid()

ax2.set_title('Error cometido en cada modelo')
ax2.plot(abs(serie_test-predi_test), label = 'Error ARIMA(1,1,0)(0,1,0)', color
ax2.plot(abs(serie_test-predi_test_auto), label = 'Error ARIMA(2,1,2)(0,1,1)', c
ax2.plot(abs(serie_test-fcast_holt_winters), label = 'Error Holt-Winters', color
ax2.set_xlabel('Fechas')
ax2.set_ylabel('Error IPC')
ax2.grid()
ax2.legend()
plt.subplots_adjust(hspace=0.3)
plt.show()
```



Tal y como augurábamos, se confirma que el modelo obtenido de forma automática obtiene una predicciones mejores que el modelo ajustado de forma manual o el obtenido con el método de suavizado de Holt-Winters. En efecto, en las primeras predicciones parece que todos los modelos se comportan de manera muy similar pero, sin embargo, vemos que el modelo ARIMA automático ajusta notablemente mejor los últimos valores de la serie temporal.

Concluimos por tanto que, el mejor modelo predictivo para modelizar la serie temporal sobre el IPC asociado al turismo y la hostelería en Canarias, responde a la siguiente expresión algebraica:

Sea la serie original Y_t, tenemos la serie diferenciada regularmente $Z_t = Y_t - Y_{t-1}$ y, a su vez, la serie diferenciada regular y estacionalmente
$W_t = Z_t - Z_{t-12} = (Y_t - Y_{t-1}) - (Y_{t-12} - Y_{t-13})$. Por ende, teniendo en cuenta que el modelo ARIMA tiene la forma
$W_t = \phi_1 W_{t-1} + \phi_2 W_{t-2} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \Theta_1 \epsilon_{t-12} + \epsilon_t$ (donde ϕ_i, θ_i, Θ_i y ϵ_t son, respectivamente, los coeficientes autoregresivos, los coeficientes de media móvil, el coeficiente de media móvil estacional y los residuos) y adoptando los valores de los parámetros suministrados por el summary que ofrece la consola de python tendríamos la siguiente expresión:

$$(Y_t - Y_{t-1}) - (Y_{t-12} - Y_{t-13}) = 1.6569(Y_{t-1} - Y_{t-2}) - 0.6934(Y_{t-2} - Y_{t-3}) - 1.8116\epsilon_t$$

In [ ]: