proyecto-u3

November 23, 2024

INTRODUCCIÓN

En el presente trabajo estaremos trabajando dos series de tiempo para estudiar dos marcas que esten relacionadas se llevará a cabo mediante Python ya que es una manera de facilitar al momento de ejecutar los códigos, el proyecto esta enfocado para en los conocimientos que hemos etado desarrollando durante las ultimas dos unidades al igual que dos nuevos conocimientos de los cuales se mencionara su definición que son la causalidad de granger y prophet.

Este documento describe todos los pasos que se llevaron a cabo para ejecutar el proceso incluyendo el diseño y funciones para las graficas. Para la selección de las dos marcas que se solicictaron se realizó la busqueda en Yahoo finance para tomar datos reales a través del tiempo para las fechas solicitadas y a partir de eso se puedan trabajar los datos como calculando la serie detiempo de ambas marcas por separado, calculando los precios de cierre, la regresión polinomial la descomposición de las series, etc. Así mismo aplicando las causalidad de granger y prophet en ambas series.

#MARCO TEÓRICO

Principalmente las marcas con las que se estará trabajando son dos tipos de marcas que se relacionan por vender del mismo tipo de producto.

La primera marca es "ELF" la cual es una marca de maquillaje y productos de cuidado de la piel que ha ganado gran popularidad en todo el mundo. Fundada en 2004 en los Estados Unidos, la marca se distingue por ofrecer productos de belleza de alta calidad a precios asequibles. La segunda marca es "Ulta Beauty" es una de las principales cadenas de tiendas de belleza en Estados Unidos, conocida por ofrecer una gran variedad de productos de maquillaje, cuidado de la piel, fragancias y herramientas de belleza.

La elección de estas dos marcas es debido a su relación ya que ambas marcas se dedican a la venta de maquillaje y productos del cuidado de la piel, las marcas son muy populares tanto entre consumidores como entre profesionales de la belleza por su amplia selección y su enfoque en la accesibilidad.

CAUSALIDAD DE GRANGER

Sirve para determinar si una serie temporal puede predecir a otra. No se trata de una causalidad en el sentido tradicional (es decir, causa y efecto), sino de una relación temporal en la que los valores pasados de una serie ayudan a predecir los valores futuros de otra.

MODELO PROPHET

Prophet es una herramienta poderosa y fácil de usar para modelar y predecir series temporales, especialmente en contextos donde hay estacionalidades complejas, datos faltantes o eventos especiales. Su simplicidad y flexibilidad la hacen ideal para usuarios que buscan una solución práctica sin necesidad de una comprensión profunda de los métodos estadísticos subyacentes, aunque también ofrece opciones de personalización para usuarios más avanzados.

Limitaciones: No es tan potente para datos altamente complejos: Prophet no es ideal para todas las situaciones. En casos de series temporales extremadamente volátiles o con patrones muy irregulares (por ejemplo, en datos financieros o de alta frecuencia), otras técnicas más avanzadas, como los modelos ARIMA o redes neuronales, podrían ser más adecuadas. Suavizado de la tendencia: Aunque Prophet permite controlar la flexibilidad de la tendencia, el modelo puede ser menos eficaz en ciertos contextos donde las tendencias cambian de manera abrupta o tienen estructuras muy complejas.

Alcances de Prophet: 1. Predicción de series temporales con estacionalidad: Prophet es especialmente eficaz para modelar series temporales con estacionalidades complejas. Es capaz de capturar patrones estacionales diarios, semanales y anuales, lo que lo hace ideal para datos como: Ventas de productos que varían con las estaciones del año. Demanda de servicios que fluctúan durante la semana (por ejemplo, turismo, tráfico web). Datos meteorológicos que siguen un patrón estacional.

2. Manejo de días festivos o eventos especiales: Uno de los puntos fuertes de Prophet es la incorporación de efectos de días festivos o eventos especiales que alteran el comportamiento normal de una serie temporal. Prophet permite especificar estos días de manera personalizada, de modo que el modelo pueda capturar los picos o caídas asociadas a estas fechas.

3. Manejo de datos faltantes y atípicos: Prophet es robusto frente a datos faltantes (missing values) y valores atípicos (outliers). No es necesario imputar los valores faltantes, ni tampoco preocuparse por los datos atípicos, ya que Prophet ajusta el modelo sin que estos afecten en gran medida el rendimiento. Esto lo hace ideal para escenarios donde los datos son incompletos o contienen errores o registros extremos que de otra forma podrían distorsionar otros modelos estadísticos más sensibles.

4. Modelado de tendencias no lineales:Prophet puede capturar tanto tendencias lineales (crecimiento o decrecimiento constante) como tendencias no lineales (crecimiento o decrecimiento acelerado o desacelerado). En el caso de una tendencia no lineal, el modelo ajusta dinámicamente la forma en que la serie temporal crece o decrece a lo largo del tiempo, lo que lo hace más flexible para situaciones en las que la tendencia no sigue un patrón fijo.

5. Descomposición de la serie temporal: Prophet descompone la serie temporal en tres componentes: Tendencia: El comportamiento a largo plazo. Estacionalidad: Los ciclos periódicos. Días festivos o eventos especiales. Esta descomposición facilita la interpretación del modelo y la comprensión de cómo cada factor contribuye a las predicciones.

6. Escalabilidad y uso en grandes volúmenes de datos: También puede manejar grandes volúmenes de datos y series temporales de largo plazo.

7. Facilidad de implementación: Prophet está diseñado para ser fácil de usar, lo que permite que incluso los usuarios sin experiencia en programación o estadísticas puedan crear modelos de predicción eficaces. Se proporciona en dos lenguajes populares: Python y R.

[152]:
```
import yfinance as yf
import pandas as pd
import warnings
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.tsa.seasonal import seasonal_decompose
from scipy.stats import ttest_rel
from statsmodels.graphics.tsaplots import plot_acf
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from statsmodels.tsa.stattools import adfuller
from numpy.polynomial.polynomial import Polynomial
from statsmodels.tsa.stattools import adfuller
```

a) Selecciona dos marcas

- ELF
- ULTA

b) Realiza un análisis de cada serie de tiempo de manera independiente

MARCA ELF

[153]:
```
df=yf.download(
    tickers = 'ELF',
    start = '2018-06-08',
    end = '2022-12-07',
    interval = '1d',
    group_by = None,
    auto_adjust= False,
)
df
```

```
[**********************100%**********************]  1 of 1 completed
```

[153]:

Ticker	ELF				\
Price	Open	High	Low	Close	
Date					
2018-06-08 00:00:00+00:00	19.150000	19.559999	19.150000	19.330000	
2018-06-11 00:00:00+00:00	19.400000	19.820000	19.391001	19.620001	
2018-06-12 00:00:00+00:00	19.480000	20.190001	19.250000	19.900000	
2018-06-13 00:00:00+00:00	20.000000	20.000000	19.170000	19.389999	
2018-06-14 00:00:00+00:00	19.410000	19.549999	19.120001	19.500000	
…	…	…	…	…	
2022-11-30 00:00:00+00:00	53.669998	55.275002	53.380001	54.959999	
2022-12-01 00:00:00+00:00	54.880001	55.903000	54.505001	55.400002	
2022-12-02 00:00:00+00:00	55.049999	55.570000	54.700001	55.259998	
2022-12-05 00:00:00+00:00	54.570000	55.049999	53.680000	53.930000	

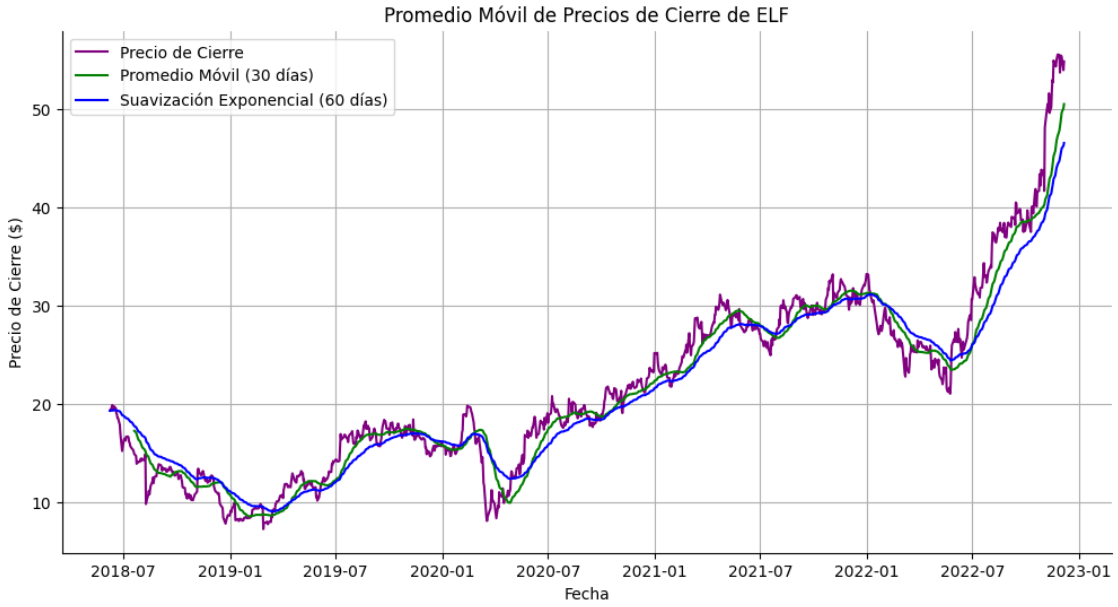
```
2022-12-06 00:00:00+00:00  54.029999  55.660000  53.980000  54.810001

Ticker
Price                      Adj Close  Volume
Date
2018-06-08 00:00:00+00:00  19.330000  458600
2018-06-11 00:00:00+00:00  19.620001  311600
2018-06-12 00:00:00+00:00  19.900000  544900
2018-06-13 00:00:00+00:00  19.389999  377500
2018-06-14 00:00:00+00:00  19.500000  233700
…                                …     …
2022-11-30 00:00:00+00:00  54.959999  840700
2022-12-01 00:00:00+00:00  55.400002  657100
2022-12-02 00:00:00+00:00  55.259998  734700
2022-12-05 00:00:00+00:00  53.930000  760200
2022-12-06 00:00:00+00:00  54.810001  820300

[1133 rows x 6 columns]
```

[154]:
```
# Paso 1: Mover 'Date' del índice a columna regular
df = df.reset_index()

# Paso 2: Aplanar el MultiIndex de las columnas, manteniendo 'Open', 'High',␣
 ↪etc.
df.columns = ['_'.join(col).strip() if col[1] != '' else col[0] for col in df.
 ↪columns.values]
df['Date'] = pd.to_datetime(df['Date'])
df['Date'] = df['Date'].dt.date
df = df.set_index('Date')
df
```

[154]:
```
             ELF_Open   ELF_High    ELF_Low  ELF_Close  ELF_Adj Close  \
Date
2018-06-08  19.150000  19.559999  19.150000  19.330000      19.330000
2018-06-11  19.400000  19.820000  19.391001  19.620001      19.620001
2018-06-12  19.480000  20.190001  19.250000  19.900000      19.900000
2018-06-13  20.000000  20.000000  19.170000  19.389999      19.389999
2018-06-14  19.410000  19.549999  19.120001  19.500000      19.500000
…                 …          …          …          …              …
2022-11-30  53.669998  55.275002  53.380001  54.959999      54.959999
2022-12-01  54.880001  55.903000  54.505001  55.400002      55.400002
2022-12-02  55.049999  55.570000  54.700001  55.259998      55.259998
2022-12-05  54.570000  55.049999  53.680000  53.930000      53.930000
2022-12-06  54.029999  55.660000  53.980000  54.810001      54.810001

            ELF_Volume
Date
```

```
2018-06-08      458600
2018-06-11      311600
2018-06-12      544900
2018-06-13      377500
2018-06-14      233700
…                 …
2022-11-30      840700
2022-12-01      657100
2022-12-02      734700
2022-12-05      760200
2022-12-06      820300

[1133 rows x 6 columns]
```

[155]:
```python
# Promedio Móvil Simple
ventana_sma = 30
df['SMA'] = df['ELF_Close'].rolling(window=ventana_sma).mean()

# Promedio Móvil Exponencial
ventana_ses = 60
df['SES'] = df['ELF_Close'].ewm(span=ventana_ses, adjust=False).mean()

# Graficar
plt.figure(figsize=(12, 6))
plt.plot(df['ELF_Close'], label='Precio de Cierre', color='purple')
plt.plot(df['SMA'], label=f'Promedio Móvil ({ventana_sma} días)', color='green')
plt.plot(df['SES'], label=f'Suavización Exponencial ({ventana_ses} días)',␣
 ↪color='blue')
plt.title('Promedio Móvil de Precios de Cierre de ELF')
plt.xlabel('Fecha')
plt.ylabel('Precio de Cierre ($)')
plt.grid()
plt.legend()
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```

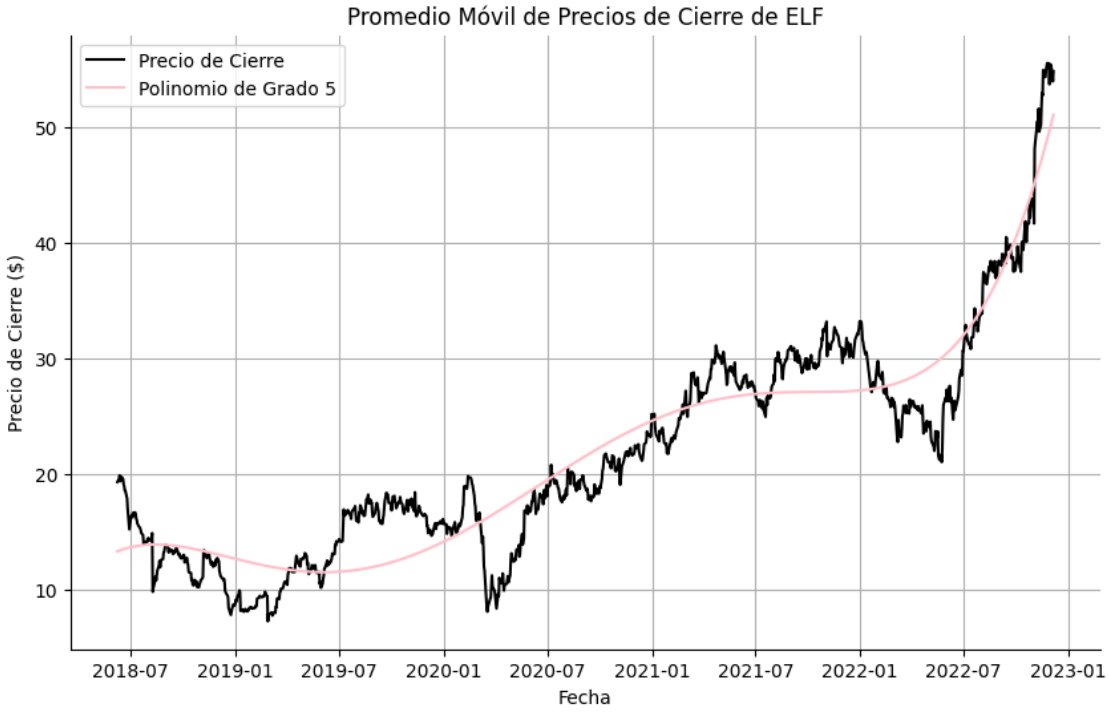


[156]:
```
#Prueba de estacionariedad con ADF
nivel_de_significancia= 0.05
adf_test= adfuller(df['ELF_Close'].dropna()) #TIRAR VALORES QUE SE VAN␣
 ↪ELIMINANDO
print('Estadistico ADF:', adf_test[0])
print('p-Value:', adf_test[1])
if adf_test[1] <= nivel_de_significancia:
  print('La serie es estacionaria (rechazamos la hipotesis nula)')
else:
  print('La serie no es estacionaria (aceptamos la hipotesis nula)')
```

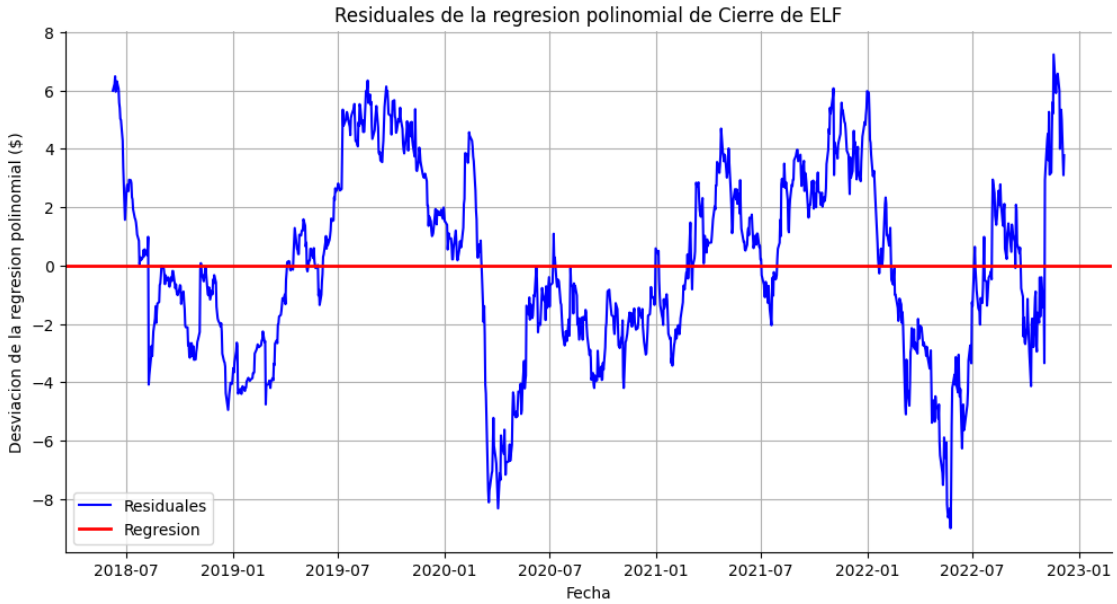
```
Estadistico ADF: 1.7903009374258128
p-Value: 0.9983295852745216
La serie no es estacionaria (aceptamos la hipotesis nula)
```

[157]:
```
#Convertir fechas numeros
df['Date'] = pd.to_datetime(df.index).map(pd.Timestamp.timestamp)
x= df['Date']
y= df ['ELF_Close']
#Ajustar un modelo polinómico
#Regresión polinomial
grado= 5
modelo= Polynomial.fit(df['Date'], df['ELF_Close'], deg = grado)
df['Poly_trend']= modelo(df['Date'])
df['Poly_resid']= df['ELF_Close']- df['Poly_trend']
#Graficar polinomial
plt.figure(figsize=(10, 6))
```

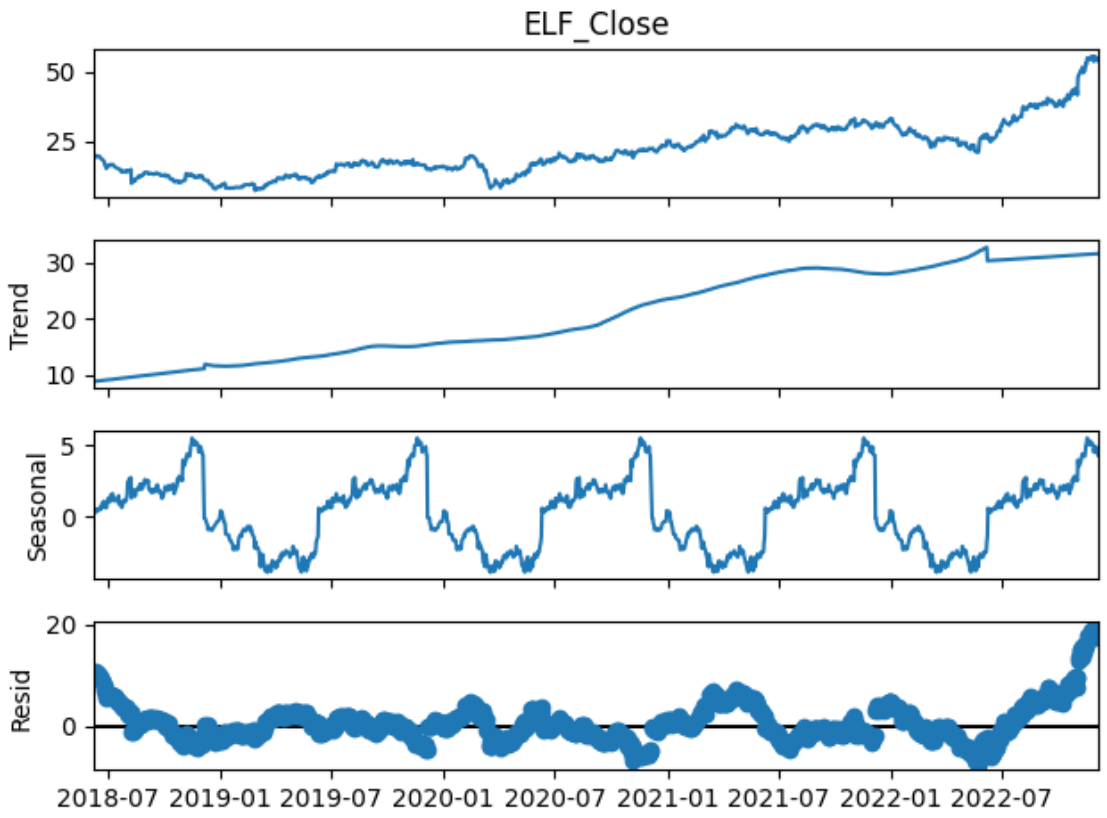
```
plt.plot(df['ELF_Close'], label='Precio de Cierre', color='black')
plt.plot(df['Poly_trend'], label=f'Polinomio de Grado {grado}', color='pink')
plt.title('Promedio Móvil de Precios de Cierre de ELF')
plt.xlabel('Fecha')
plt.ylabel('Precio de Cierre ($)')
plt.grid()
plt.legend()
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```



[158]:
```
#Graficar residuales
plt.figure(figsize=(12, 6)) # abre una nueva ventana gráfica
plt.plot(df['Poly_resid'], label='Residuales', color='blue')
plt.axhline(0, color='red', label='Regresion',linestyle='solid', linewidth=2) ␣
  ↪# Línea horizontal en el valor cero
plt.title('Residuales de la regresion polinomial de Cierre de ELF')
plt.xlabel('Fecha')
plt.ylabel('Desviacion de la regresion polinomial ($)')
plt.legend()
plt.grid()
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```



[159]:
```
descomposicion= sm.tsa.seasonal_decompose(
df['ELF_Close'],
model= 'additive',
period= 252,
extrapolate_trend='freq' # Reemplazar estrapolate_trend=0 por␣
 ↪extrapolate_trend='freq'
) #252 dias de negociacion al año
#Graficar
fig= descomposicion.plot()
plt.show()
```



[160]:
```
#Prueba de estacionariedad: Prueba de Dickey-Fuller aumentada (ADF)
nivel_de_significancia = 0.05

# Realizar la prueba de Dickey-Fuller en la tendencia
adf_test = adfuller(descomposicion.resid.dropna())

print("Estadístico ADF:", adf_test[0])
print("P-valor:", adf_test[1])

if adf_test[1] <= nivel_de_significancia:
    print("La tendencia es estacionaria (rechazamos la hipótesis nula)")
else:
    print("La tendencia no es estacionaria (aceptamos la hipótesis nula)")
```

```
Estadístico ADF: -1.3664924239607183
P-valor: 0.5982305316572688
La tendencia no es estacionaria (aceptamos la hipótesis nula)
```

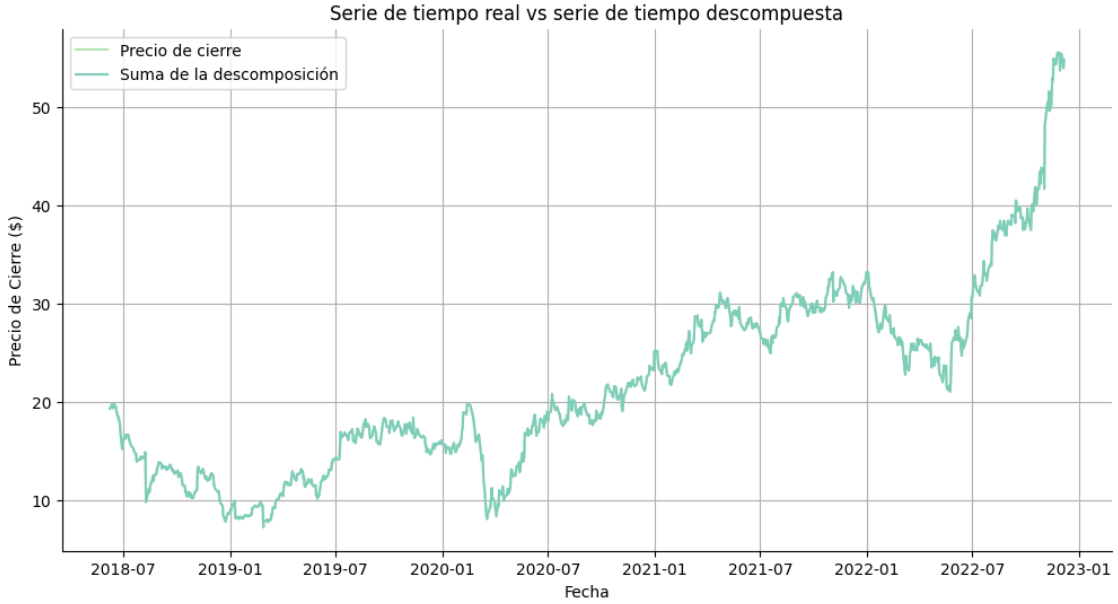
[161]:
```
from scipy.stats import ttest_rel

# Prueba t pareada para el efecto significativo de la estacionalidad
```

```
#Hipótesis nula (H ): Ambas series son iguales.
#Hipótesis alternativa (H ): Ambas series son diferentes


nivel_de_significancia = 0.05
tendencia = descomposicion.trend
tendencia_estacionalidad = descomposicion.seasonal + descomposicion.trend

# Eliminar valores NaN de ambas series
tendencia.dropna(inplace=True)
tendencia_estacionalidad.dropna(inplace=True)

# Realizar la prueba t pareada
t_stat, p_valor = ttest_rel(tendencia, tendencia_estacionalidad)

print("Estadístico t:", t_stat)
print("Valor p:", p_valor)
print("\n")

# Interpretación de los resultados
if p_valor < nivel_de_significancia:
    print("El valor p es menor que 0.05, por lo tanto, rechazamos la hipótesis␣
 ↪nula.")
    print("Conclusión: La estacionalidad tiene un efecto significativo en la␣
 ↪serie de tiempo.")
else:
    print("El valor p es mayor o igual que 0.05, por lo tanto, no podemos␣
 ↪rechazar la hipótesis nula.")
    print("Conclusión: La estacionalidad no tiene un efecto significativo en la␣
 ↪serie de tiempo.")
```

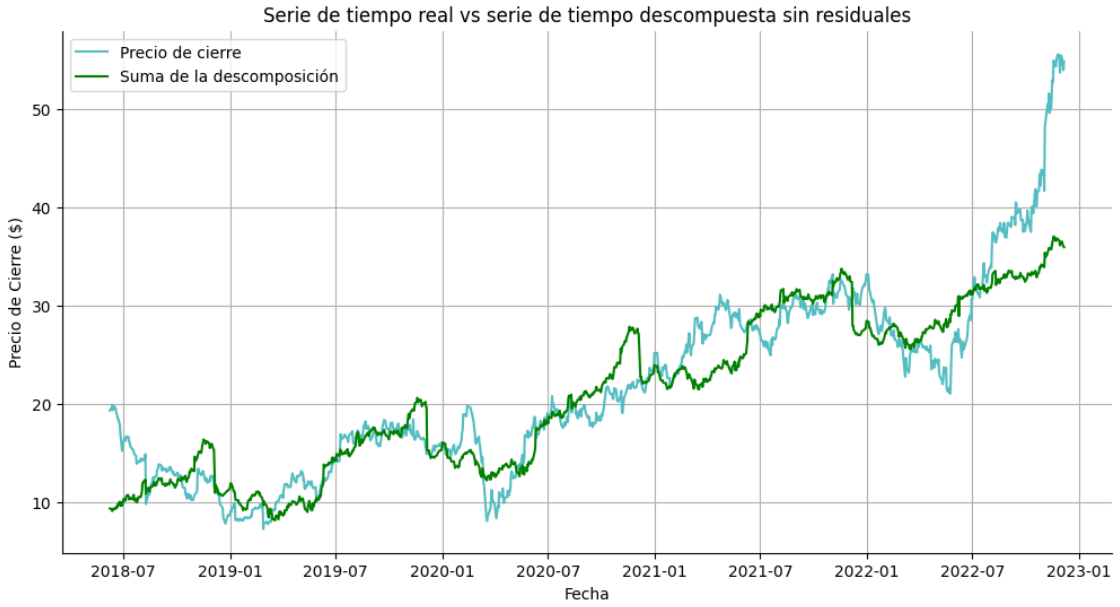
```
Estadístico t: -3.2538027561675333
Valor p: 0.0011723763618661988


El valor p es menor que 0.05, por lo tanto, rechazamos la hipótesis nula.
Conclusión: La estacionalidad tiene un efecto significativo en la serie de
tiempo.
```

[162]:
```
# Graficar descomposición
plt.figure(figsize=(12, 6)) # abre una nueva ventana gráfica
plt.plot(df['ELF_Close'], label='Precio de cierre', color='#B6E4B3')
plt.plot(descomposicion.trend + descomposicion.seasonal + descomposicion.resid,␣
 ↪label=f'Suma de la descomposición', color='#7ECDBB')
plt.title('Serie de tiempo real vs serie de tiempo descompuesta')
plt.xlabel('Fecha')
```

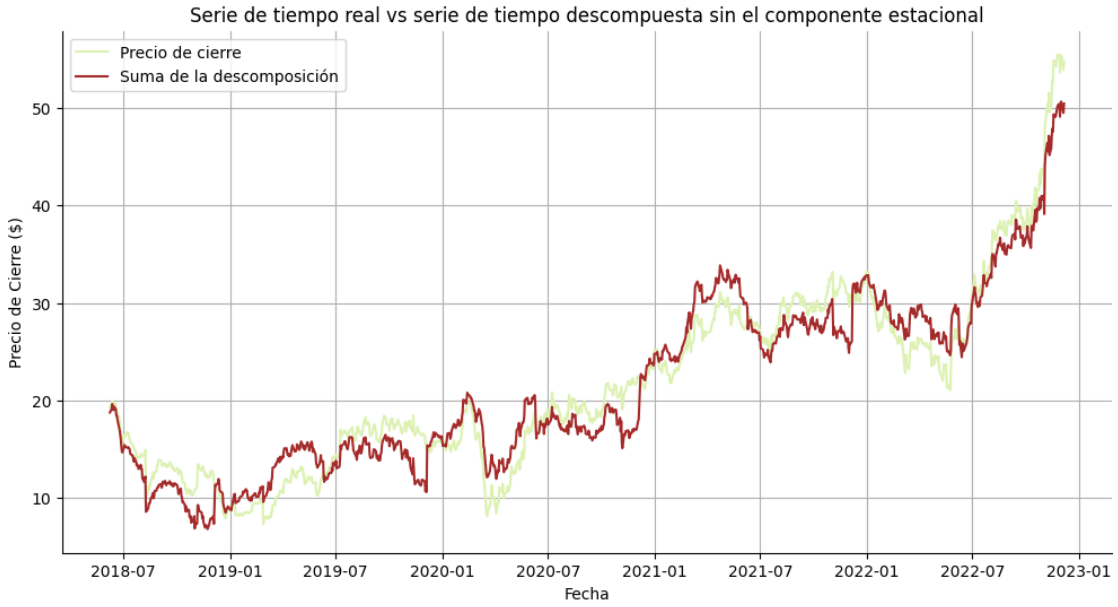
```
plt.ylabel('Precio de Cierre ($)')
plt.legend()
plt.grid()
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```



[163]:
```
# Graficar descomposición
plt.figure(figsize=(12, 6)) # abre una nueva ventana gráfica
plt.plot(df['ELF_Close'], label='Precio de cierre', color='#54BDC2')
plt.plot(descomposicion.trend + descomposicion.seasonal, label=f'Suma de la␣
 ↪descomposición', color='green')
plt.title('Serie de tiempo real vs serie de tiempo descompuesta sin residuales')
plt.xlabel('Fecha')
plt.ylabel('Precio de Cierre ($)')
plt.legend()
plt.grid()
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```



[164]:
```
# Graficar descomposición
plt.figure(figsize=(12, 6)) # abre una nueva ventana gráfica
plt.plot(df['ELF_Close'], label='Precio de cierre', color='#DDF2B2')
plt.plot(descomposicion.trend + descomposicion.resid, label=f'Suma de la␣
 ↪descomposición', color='brown')
plt.title('Serie de tiempo real vs serie de tiempo descompuesta sin el␣
 ↪componente estacional')
plt.xlabel('Fecha')
plt.ylabel('Precio de Cierre ($)')
plt.legend()
plt.grid()
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```



MARCA ULTA

[165]:
```
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.tsa.seasonal import seasonal_decompose
from scipy.stats import ttest_rel
from statsmodels.graphics.tsaplots import plot_acf
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from statsmodels.tsa.stattools import adfuller
from numpy.polynomial.polynomial import Polynomial
from statsmodels.tsa.stattools import adfuller
```

[166]:
```
df=yf.download(
    tickers = 'ULTA',
    start = '2018-06-08', # Changed 'Start' to 'start'
    end = '2022-12-07',
    interval = '1d',
    group_by = None,
    auto_adjust = False,
    actions = False
)
df
```

```
[**********************100%**********************]  1 of 1 completed
```

[166]:
```
Ticker                           ULTA                                    \
Price                            Open        High         Low       Close
Date
2018-06-08 00:00:00+00:00  255.229996  256.589996  248.130005  253.059998
2018-06-11 00:00:00+00:00  253.309998  254.240005  250.529999  251.839996
2018-06-12 00:00:00+00:00  251.279999  252.740005  247.449997  248.289993
2018-06-13 00:00:00+00:00  249.000000  251.809998  246.929993  247.520004
2018-06-14 00:00:00+00:00  247.699997  247.779999  243.679993  246.600006
…                                …           …           …         …
2022-11-30 00:00:00+00:00  449.950012  466.549988  447.059998  464.839996
2022-12-01 00:00:00+00:00  470.470001  477.079987  464.000000  472.529999
2022-12-02 00:00:00+00:00  467.049988  477.920013  461.880005  471.329987
2022-12-05 00:00:00+00:00  468.339996  473.109985  465.390015  472.519989
2022-12-06 00:00:00+00:00  472.000000  474.480011  460.230011  465.579987

Ticker
Price                         Adj Close   Volume
Date
2018-06-08 00:00:00+00:00  253.059998  1100400
2018-06-11 00:00:00+00:00  251.839996   779400
2018-06-12 00:00:00+00:00  248.289993  1220700
2018-06-13 00:00:00+00:00  247.520004  1262300
2018-06-14 00:00:00+00:00  246.600006  1454000
…                                 …        …
2022-11-30 00:00:00+00:00  464.839996  1193600
2022-12-01 00:00:00+00:00  472.529999  1499700
2022-12-02 00:00:00+00:00  471.329987  1526400
2022-12-05 00:00:00+00:00  472.519989   941300
2022-12-06 00:00:00+00:00  465.579987   897100

[1133 rows x 6 columns]
```

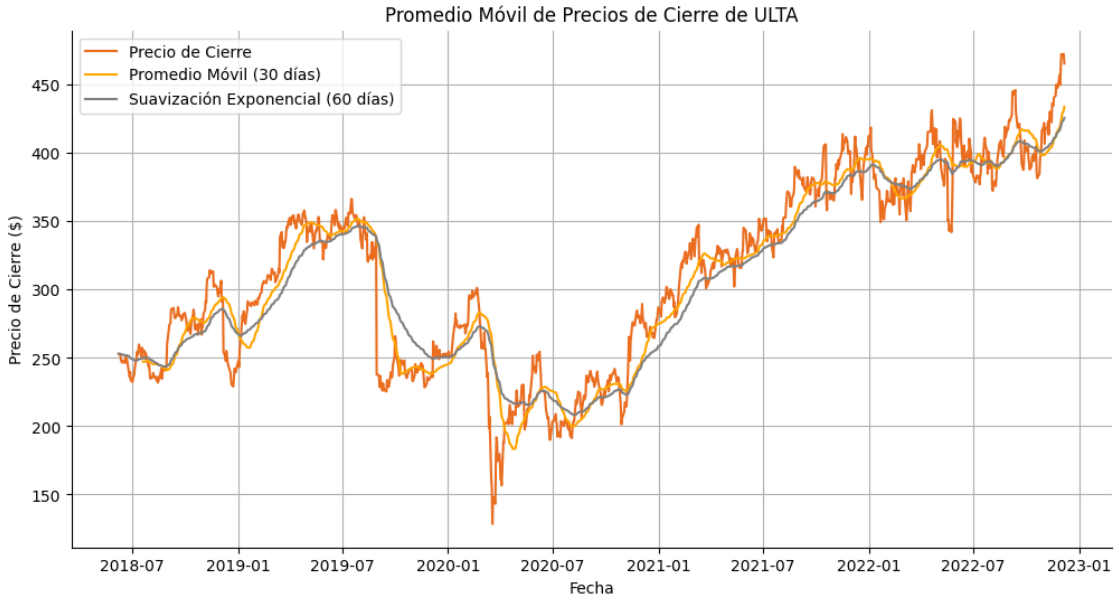
[167]:
```
# Paso 1: Mover 'Date' del índice a columna regular
df = df.reset_index()

# Paso 2: Aplanar el MultiIndex de las columnas, manteniendo 'Open', 'High',␣
 ↪etc.
df.columns = ['_'.join(col).strip() if col[1] != '' else col[0] for col in df.
 ↪columns.values]
df['Date'] = pd.to_datetime(df['Date'])
df['Date'] = df['Date'].dt.date
df = df.set_index('Date')
df
```

[167]:
```
            ULTA_Open   ULTA_High    ULTA_Low  ULTA_Close  ULTA_Adj Close  \
Date
2018-06-08  255.229996  256.589996  248.130005  253.059998      253.059998
```

```
2018-06-11  253.309998  254.240005  250.529999  251.839996      251.839996
2018-06-12  251.279999  252.740005  247.449997  248.289993      248.289993
2018-06-13  249.000000  251.809998  246.929993  247.520004      247.520004
2018-06-14  247.699997  247.779999  243.679993  246.600006      246.600006
…                  …           …           …           …               …
2022-11-30  449.950012  466.549988  447.059998  464.839996      464.839996
2022-12-01  470.470001  477.079987  464.000000  472.529999      472.529999
2022-12-02  467.049988  477.920013  461.880005  471.329987      471.329987
2022-12-05  468.339996  473.109985  465.390015  472.519989      472.519989
2022-12-06  472.000000  474.480011  460.230011  465.579987      465.579987

            ULTA_Volume
Date
2018-06-08      1100400
2018-06-11       779400
2018-06-12      1220700
2018-06-13      1262300
2018-06-14      1454000
…                   …
2022-11-30      1193600
2022-12-01      1499700
2022-12-02      1526400
2022-12-05       941300
2022-12-06       897100

[1133 rows x 6 columns]
```

[168]:
```python
# Promedio Móvil Simple
ventana_sma = 30
df['SMA'] = df['ULTA_Close'].rolling(window=ventana_sma).mean()

# Promedio Móvil Exponencial
ventana_ses = 60
df['SES'] = df['ULTA_Close'].ewm(span=ventana_ses, adjust=False).mean()

# Graficar
plt.figure(figsize=(12, 6))
plt.plot(df['ULTA_Close'], label='Precio de Cierre', color='#EA6D20')
plt.plot(df['SMA'], label=f'Promedio Móvil ({ventana_sma} días)',␣
 ↪color='orange')
plt.plot(df['SES'], label=f'Suavización Exponencial ({ventana_ses} días)',␣
 ↪color='gray')
plt.title('Promedio Móvil de Precios de Cierre de ULTA')
plt.xlabel('Fecha')
plt.ylabel('Precio de Cierre ($)')
plt.grid()
plt.legend()
```

```
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```

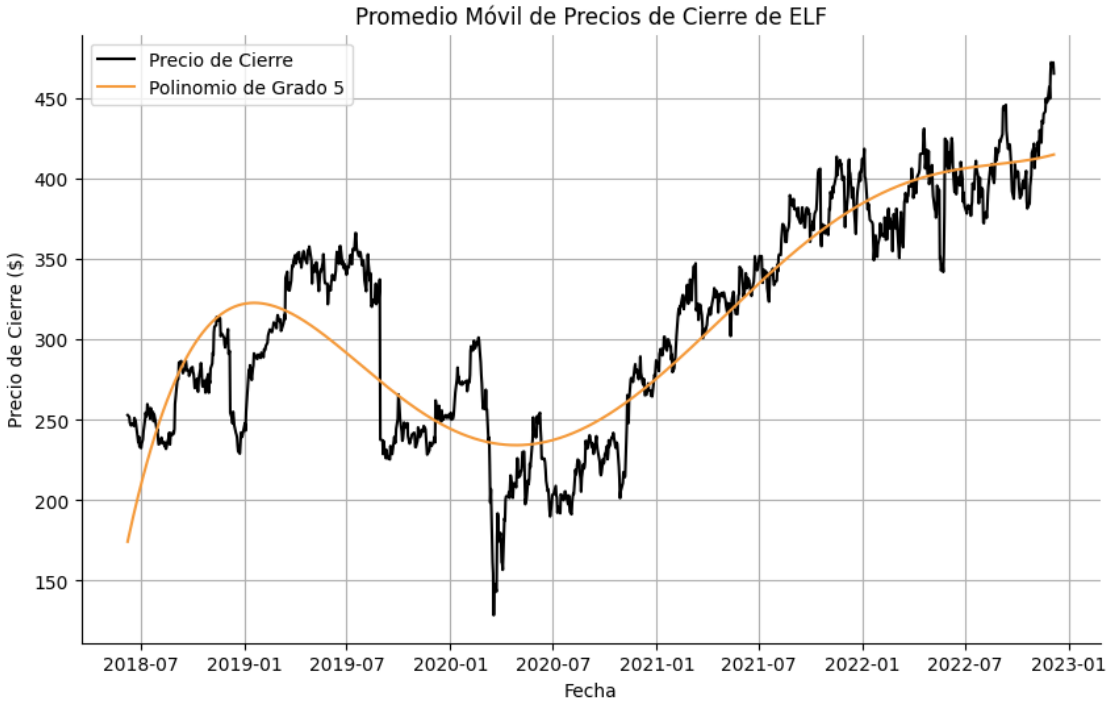


[169]:
```
#Prueba de estacionariedad con ADF
nivel_de_significancia= 0.05
adf_test= adfuller(df['ULTA_Close'].dropna()) #TIRAR VALORES QUE SE VAN␣
 ↪ELIMINANDO
print('Estadistico ADF:', adf_test[0])
print('p-Value:', adf_test[1])
if adf_test[1] <= nivel_de_significancia:
  print('La serie es estacionaria (rechazamos la hipotesis nula)')
else:
  print('La serie no es estacionaria (aceptamos la hipotesis nula)')
```

```
Estadistico ADF: -1.4036994135328107
p-Value: 0.5805012937325651
La serie no es estacionaria (aceptamos la hipotesis nula)
```

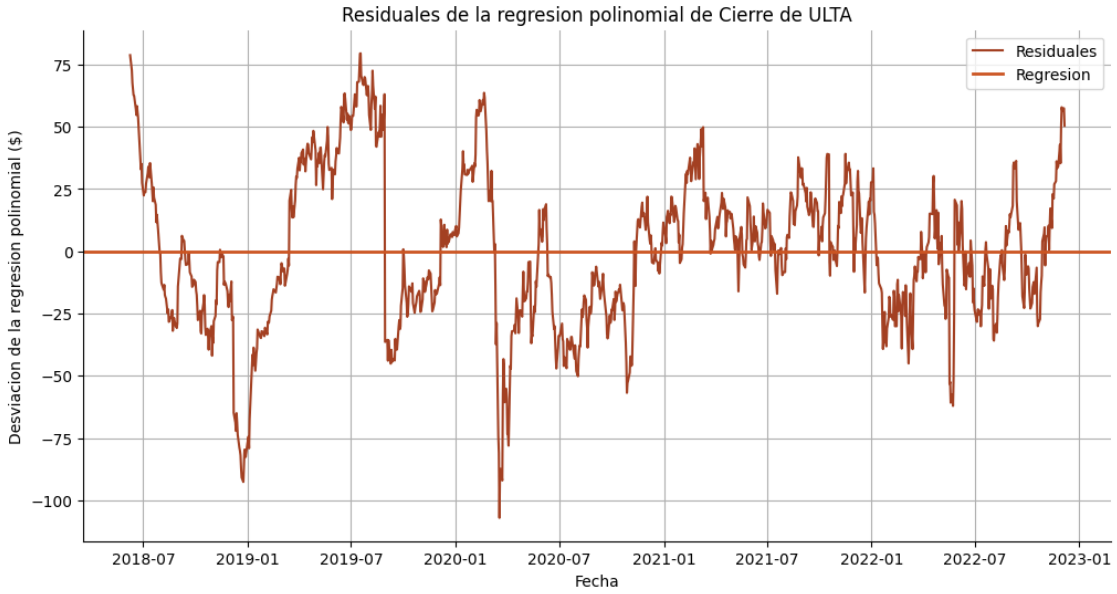
[170]:
```
#Convertir fechas numeros
df['Date'] = pd.to_datetime(df.index).map(pd.Timestamp.timestamp)
x= df['Date']
y= df ['ULTA_Close']
#Ajustar un modelo polinómico
#Regresión polinomial
grado= 5
modelo= Polynomial.fit(df['Date'], df['ULTA_Close'], deg = grado)
```

```
df['Poly_trend']= modelo(df['Date'])
df['Poly_resid']= df['ULTA_Close']- df['Poly_trend']
#Graficar polinomial
plt.figure(figsize=(10, 6))
plt.plot(df['ULTA_Close'], label='Precio de Cierre', color='black')
plt.plot(df['Poly_trend'], label=f'Polinomio de Grado {grado}', color='#F59A3A')
plt.title('Promedio Móvil de Precios de Cierre de ULTA')
plt.xlabel('Fecha')
plt.ylabel('Precio de Cierre ($)')
plt.grid()
plt.legend()
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```

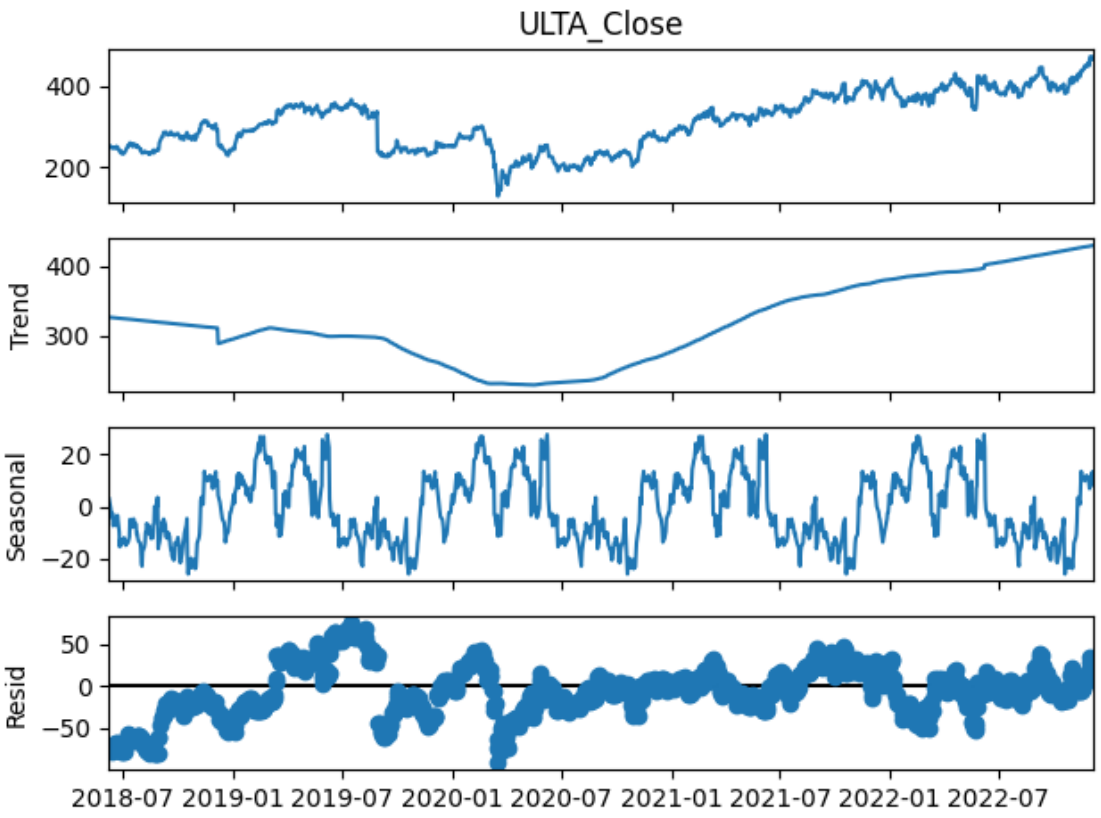


[171]:
```
#Graficar residuales
plt.figure(figsize=(12, 6)) # abre una nueva ventana gráfica
plt.plot(df['Poly_resid'], label='Residuales', color='#A34022')
plt.axhline(0, color='#CC5522', label='Regresion',linestyle='solid',␣
 ↪linewidth=2)  # Línea horizontal en el valor cero
plt.title('Residuales de la regresion polinomial de Cierre de ULTA')
plt.xlabel('Fecha')
plt.ylabel('Desviacion de la regresion polinomial ($)')
plt.legend()
```

```
plt.grid()
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```



[172]:
```
descomposicion= sm.tsa.seasonal_decompose(
df['ULTA_Close'],
model= 'additive',
period= 252,
extrapolate_trend='freq' # Reemplazar estrapolate_trend=0 por␣
 ↪extrapolate_trend='freq'
) #252 dias de negociacion al año
#Graficar
fig= descomposicion.plot()
plt.show()
```



[173]:
```
#Prueba de estacionariedad: Prueba de Dickey-Fuller aumentada (ADF)
nivel_de_significancia = 0.05

# Realizar la prueba de Dickey-Fuller en la tendencia
adf_test = adfuller(descomposicion.resid.dropna())

print("Estadístico ADF:", adf_test[0])
print("P-valor:", adf_test[1])

if adf_test[1] <= nivel_de_significancia:
    print("La tendencia es estacionaria (rechazamos la hipótesis nula)")
else:
    print("La tendencia no es estacionaria (aceptamos la hipótesis nula)")
```

```
Estadístico ADF: -4.186577700615921
P-valor: 0.0006942320751826946
La tendencia es estacionaria (rechazamos la hipótesis nula)
```

[174]:
```
from scipy.stats import ttest_rel

# Prueba t pareada para el efecto significativo de la estacionalidad
```

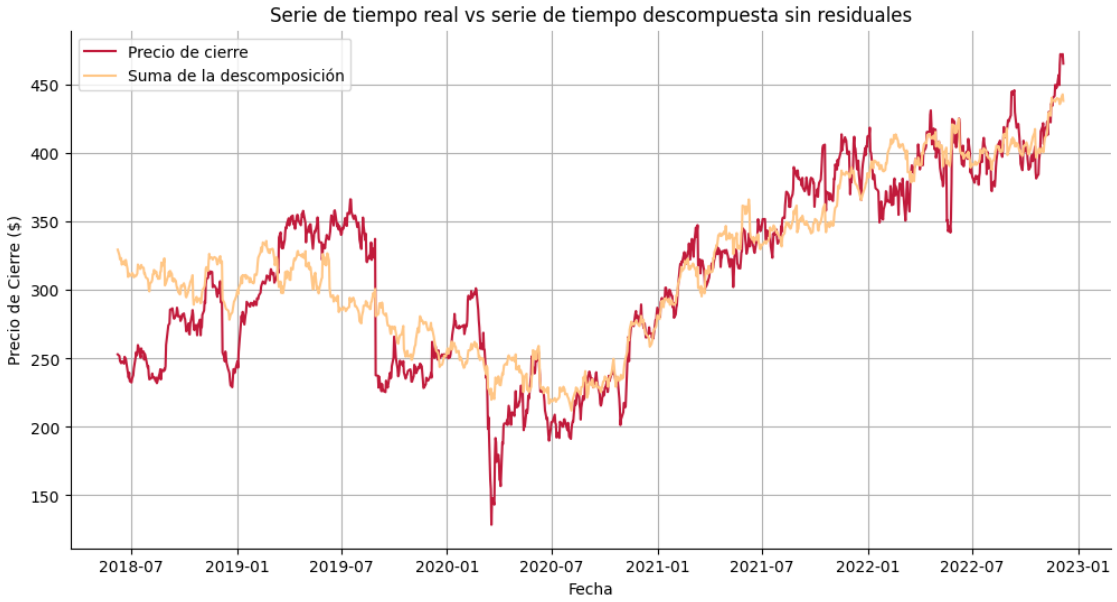
```
#Hipótesis nula (H ): Ambas series son iguales.
#Hipótesis alternativa (H ): Ambas series son diferentes


nivel_de_significancia = 0.05
tendencia = descomposicion.trend
tendencia_estacionalidad = descomposicion.seasonal + descomposicion.trend

# Eliminar valores NaN de ambas series
tendencia.dropna(inplace=True)
tendencia_estacionalidad.dropna(inplace=True)

# Realizar la prueba t pareada
t_stat, p_valor = ttest_rel(tendencia, tendencia_estacionalidad)

print("Estadístico t:", t_stat)
print("Valor p:", p_valor)
print("\n")

# Interpretación de los resultados
if p_valor < nivel_de_significancia:
    print("El valor p es menor que 0.05, por lo tanto, rechazamos la hipótesis␣
 ↪nula.")
    print("Conclusión: La estacionalidad tiene un efecto significativo en la␣
 ↪serie de tiempo.")
else:
    print("El valor p es mayor o igual que 0.05, por lo tanto, no podemos␣
 ↪rechazar la hipótesis nula.")
    print("Conclusión: La estacionalidad no tiene un efecto significativo en la␣
 ↪serie de tiempo.")
```

```
Estadístico t: 2.520250266747491
Valor p: 0.011863787849845434


El valor p es menor que 0.05, por lo tanto, rechazamos la hipótesis nula.
Conclusión: La estacionalidad tiene un efecto significativo en la serie de
tiempo.
```

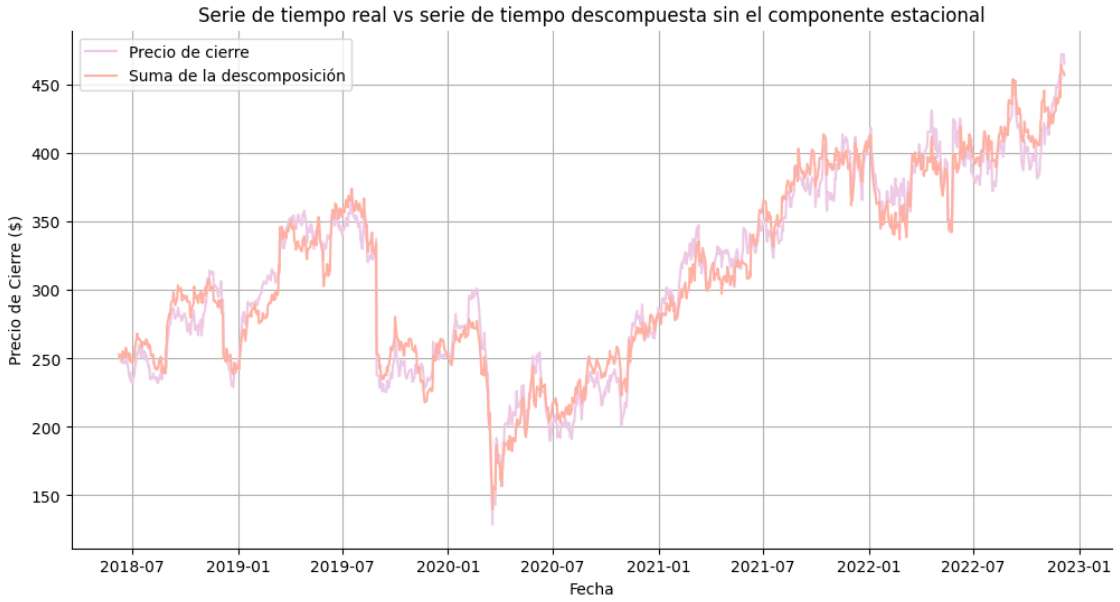
[175]:
```
# Graficar descomposición
plt.figure(figsize=(12, 6)) # abre una nueva ventana gráfica
plt.plot(df['ULTA_Close'], label='Precio de cierre', color='#F9B666')
plt.plot(descomposicion.trend + descomposicion.seasonal + descomposicion.resid,␣
 ↪label=f'Suma de la descomposición', color='#B983A7')
plt.title('Serie de tiempo real vs serie de tiempo descompuesta')
plt.xlabel('Fecha')
```

```
plt.ylabel('Precio de Cierre ($)')
plt.legend()
plt.grid()
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```

[176]:
```
# Graficar descomposición
plt.figure(figsize=(12, 6)) # abre una nueva ventana gráfica
plt.plot(df['ULTA_Close'], label='Precio de cierre', color='#C11A3B')
plt.plot(descomposicion.trend + descomposicion.seasonal, label=f'Suma de la␣
 ↪descomposición', color='#FFC685')
plt.title('Serie de tiempo real vs serie de tiempo descompuesta sin residuales')
plt.xlabel('Fecha')
plt.ylabel('Precio de Cierre ($)')
plt.legend()
plt.grid()
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```



[177]:
```
# Graficar descomposición
plt.figure(figsize=(12, 6)) # abre una nueva ventana gráfica
plt.plot(df['ULTA_Close'], label='Precio de cierre', color='#EEC9E5')
plt.plot(descomposicion.trend + descomposicion.resid, label=f'Suma de la␣
 ↪descomposición', color='#FEB1A3')
plt.title('Serie de tiempo real vs serie de tiempo descompuesta sin el␣
 ↪componente estacional')
plt.xlabel('Fecha')
plt.ylabel('Precio de Cierre ($)')
plt.legend()
plt.grid()
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.show()
```



CAUSALIDAD DE GRANGER

[178]:
```
warnings.filterwarnings('ignore')

# Obtener datos de acciones
df = yf.download(
    tickers=['ELF', 'ULTA'],            # tickers: ELF, ULTA, etc.
    start='2018-06-08',         # Fecha de inicio
    end='2023-07-12',           # Fecha de fin
    interval='1d',              # Intervalo de tiempo (1 día)
    group_by=None,              # Agrupar por ticker
    auto_adjust=False,          # ajusta automáticamente los precios de cierre,␣
 ↪apertura, máximo y mínimo para tener en cuenta los dividendos y divisiones␣
 ↪de acciones.
    actions=False,              # Si se establece en True, incluye datos sobre␣
 ↪acciones, como dividendos y divisiones.
    )
df = df.reset_index()

df.columns = ['_'.join(col).strip() if col[1] != '' else col[0] for col in df.
 ↪columns.values]
df['Date'] = pd.to_datetime(df['Date'])

df['Date'] = df['Date'].dt.date
df.set_index('Date', inplace=True)
```

```
df
```

```
[**********************100%***********************]  2 of 2 completed
```

[178]:

Date	ELF_Open	ELF_High	ELF_Low	ELF_Close	ELF_Adj Close	ELF_Volume	ULTA_Open	ULTA_High	ULTA_Low	ULTA_Close	ULTA_Adj Close	ULTA_Volume
2018-06-08	19.150000	19.559999	19.150000	19.330000	19.330000	458600	255.229996	256.589996	248.130005	253.059998	253.059998	1100400
2018-06-11	19.400000	19.820000	19.391001	19.620001	19.620001	311600	253.309998	254.240005	250.529999	251.839996	251.839996	779400
2018-06-12	19.480000	20.190001	19.250000	19.900000	19.900000	544900	251.279999	252.740005	247.449997	248.289993	248.289993	1220700
2018-06-13	20.000000	20.000000	19.170000	19.389999	19.389999	377500	249.000000	251.809998	246.929993	247.520004	247.520004	1262300
2018-06-14	19.410000	19.549999	19.120001	19.500000	19.500000	233700	247.699997	247.779999	243.679993	246.600006	246.600006	1454000
…	…	…	…	…	…	…	…	…	…	…	…	…
2023-07-05	113.639999	114.949997	112.620003	114.370003	114.370003	1336700	471.709991	480.559998	468.230011	479.829987	479.829987	789500
2023-07-06	113.540001	114.019997	110.769997	111.099998	111.099998	817200	476.200012	479.059998	470.010010	470.549988	470.549988	652800
2023-07-07	111.099998	112.660004	109.510002	110.000000	110.000000	934700	471.250000	476.000000	468.230011	471.630005	471.630005	530100
2023-07-10	110.349998	113.300003	110.349998	112.339996	112.339996	783300	473.429993	478.959991	473.130005	478.000000	478.000000	571600
2023-07-11	112.620003	114.260002	111.419998	113.489998	113.489998	753900	479.500000	484.709991	478.440002	483.239990	483.239990	506000

```
[1280 rows x 12 columns]
```

[179]:
```
from statsmodels.tsa.stattools import grangercausalitytests

# Hipótesis Nula (H ): La serie X no causa en el sentido de Granger a la serie␣
 ↪Y.

# Hipótesis Alternativa (H ): La serie X causa en el sentido de Granger a la␣
 ↪serie Y.

# Definir el número máximo de rezagos para la prueba
max_lags = 5

# Realizar la prueba de causalidad de Granger
# La función devuelve resultados para varios tests y cada rezago hasta el␣
 ↪máximo definido
resultado = grangercausalitytests(df[['ELF_Close', 'ULTA_Close']], max_lags,␣
 ↪verbose=True)
```

```
Granger Causality
number of lags (no zero) 1
ssr based F test:         F=0.3149  , p=0.5748  , df_denom=1276, df_num=1
ssr based chi2 test:   chi2=0.3157  , p=0.5742  , df=1
likelihood ratio test: chi2=0.3156  , p=0.5743  , df=1
parameter F test:         F=0.3149  , p=0.5748  , df_denom=1276, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=0.1733  , p=0.8409  , df_denom=1273, df_num=2
ssr based chi2 test:   chi2=0.3480  , p=0.8403  , df=2
likelihood ratio test: chi2=0.3479  , p=0.8403  , df=2
parameter F test:         F=0.1733  , p=0.8409  , df_denom=1273, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=0.1346  , p=0.9394  , df_denom=1270, df_num=3
ssr based chi2 test:   chi2=0.4061  , p=0.9390  , df=3
likelihood ratio test: chi2=0.4060  , p=0.9390  , df=3
parameter F test:         F=0.1346  , p=0.9394  , df_denom=1270, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=0.3331  , p=0.8558  , df_denom=1267, df_num=4
ssr based chi2 test:   chi2=1.3418  , p=0.8542  , df=4
likelihood ratio test: chi2=1.3411  , p=0.8544  , df=4
parameter F test:         F=0.3331  , p=0.8558  , df_denom=1267, df_num=4

Granger Causality
```

```
number of lags (no zero) 5
ssr based F test:         F=0.2984  , p=0.9139  , df_denom=1264, df_num=5
ssr based chi2 test:   chi2=1.5049  , p=0.9125  , df=5
likelihood ratio test: chi2=1.5040  , p=0.9126  , df=5
parameter F test:         F=0.2984  , p=0.9139  , df_denom=1264, df_num=5
```

Prphet

MARCA ELF

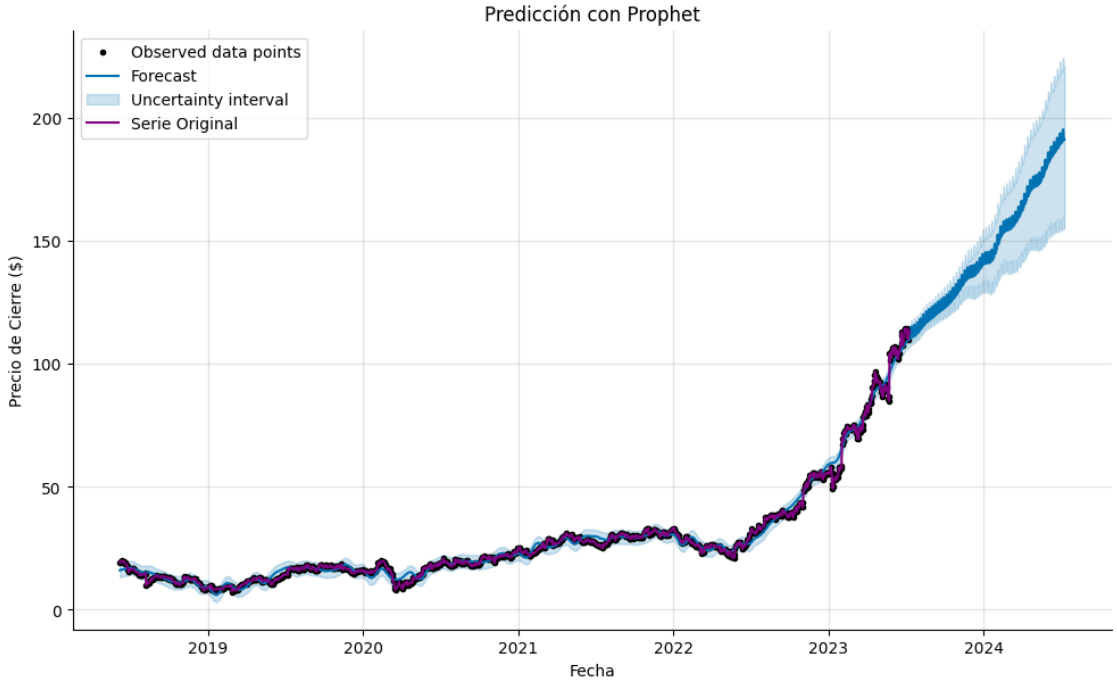
[181]:
```
!pip install prophet
```

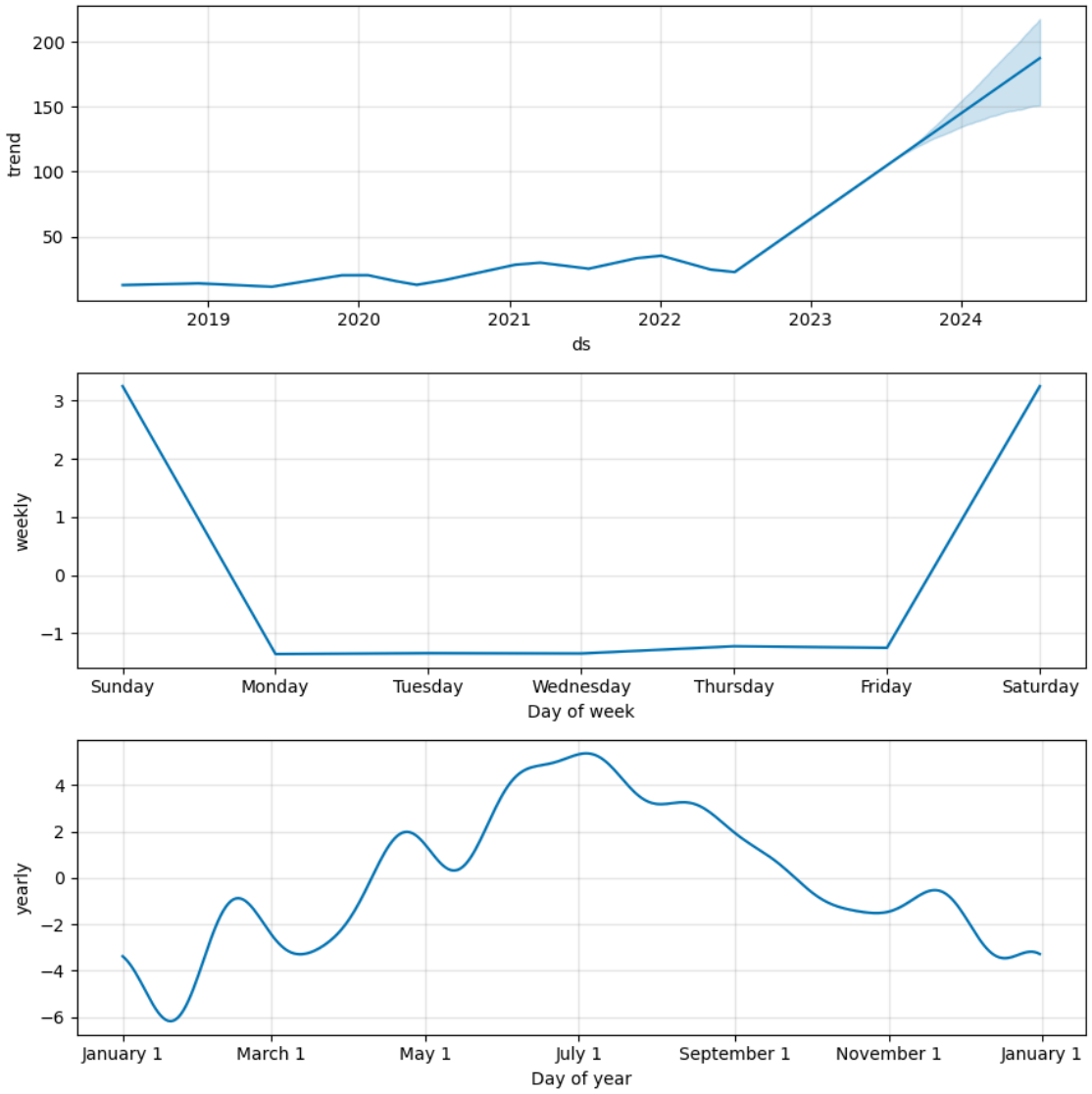
```
Requirement already satisfied: prophet in /usr/local/lib/python3.10/dist-
packages (1.1.6)
Requirement already satisfied: cmdstanpy>=1.0.4 in
/usr/local/lib/python3.10/dist-packages (from prophet) (1.2.4)
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.10/dist-
packages (from prophet) (1.26.4)
Requirement already satisfied: matplotlib>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from prophet) (3.8.0)
Requirement already satisfied: pandas>=1.0.4 in /usr/local/lib/python3.10/dist-
packages (from prophet) (2.2.2)
Requirement already satisfied: holidays<1,>=0.25 in
/usr/local/lib/python3.10/dist-packages (from prophet) (0.61)
Requirement already satisfied: tqdm>=4.36.1 in /usr/local/lib/python3.10/dist-
packages (from prophet) (4.66.6)
Requirement already satisfied: importlib-resources in
/usr/local/lib/python3.10/dist-packages (from prophet) (6.4.5)
Requirement already satisfied: stanio<2.0.0,>=0.4.0 in
/usr/local/lib/python3.10/dist-packages (from cmdstanpy>=1.0.4->prophet) (0.5.1)
Requirement already satisfied: python-dateutil in
/usr/local/lib/python3.10/dist-packages (from holidays<1,>=0.25->prophet)
(2.8.2)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet)
(1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib>=2.0.0->prophet) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet)
(4.55.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet)
(1.4.7)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet) (24.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-
packages (from matplotlib>=2.0.0->prophet) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in
```

```
/usr/local/lib/python3.10/dist-packages (from matplotlib>=2.0.0->prophet)
(3.2.0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=1.0.4->prophet) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-
packages (from pandas>=1.0.4->prophet) (2024.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil->holidays<1,>=0.25->prophet) (1.16.0)
```

[187]:
```python
from prophet import Prophet

# Crear el modelo y ajustarlo
modelo = Prophet()
modelo.fit(df['ELF_Close'].reset_index().rename(columns={'Date': 'ds',␣
 ↪'ELF_Close': 'y'}))

# Predicción para los próximos 365 días
futuro = modelo.make_future_dataframe(periods=365)
predicciones = modelo.predict(futuro)

# Visualizar las predicciones
fig = modelo.plot(predicciones)
plt.plot(df['ELF_Close'], label='Serie Original', color='purple')
plt.xlabel('Fecha')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.title('Predicción con Prophet')
plt.ylabel('Precio de Cierre ($)')
plt.legend()
fig = modelo.plot_components(predicciones)
```

```
INFO:prophet:Disabling daily seasonality. Run prophet with
daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmp9ivjos63/5ld2xsuk.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmp9ivjos63/rc_7k9dm.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-
packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=93719', 'data',
'file=/tmp/tmp9ivjos63/5ld2xsuk.json', 'init=/tmp/tmp9ivjos63/rc_7k9dm.json',
'output',
'file=/tmp/tmp9ivjos63/prophet_modeliq8in36g/prophet_model-20241122201220.csv',
'method=optimize', 'algorithm=lbfgs', 'iter=10000']
20:12:20 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
20:12:22 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```

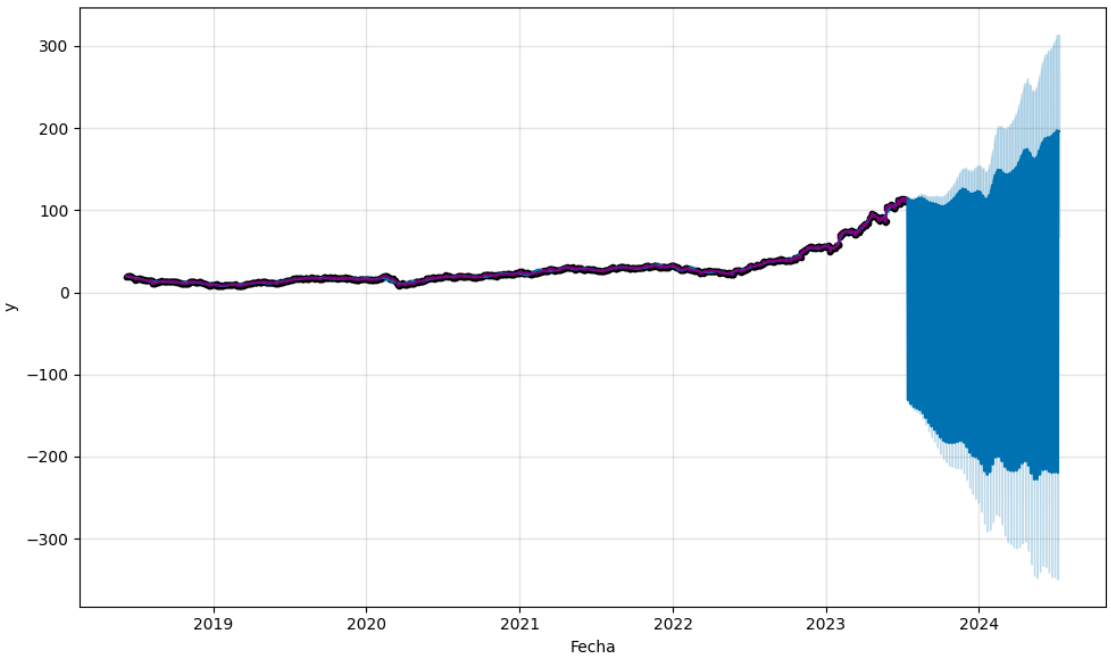

Predicción con Prophet
Observed data points
Forecast
Uncertainty interval
Serie Original
Precio de Cierre ($)
0
50
100
150
200
2019
2020
2021
2022
2023
2024
Fecha

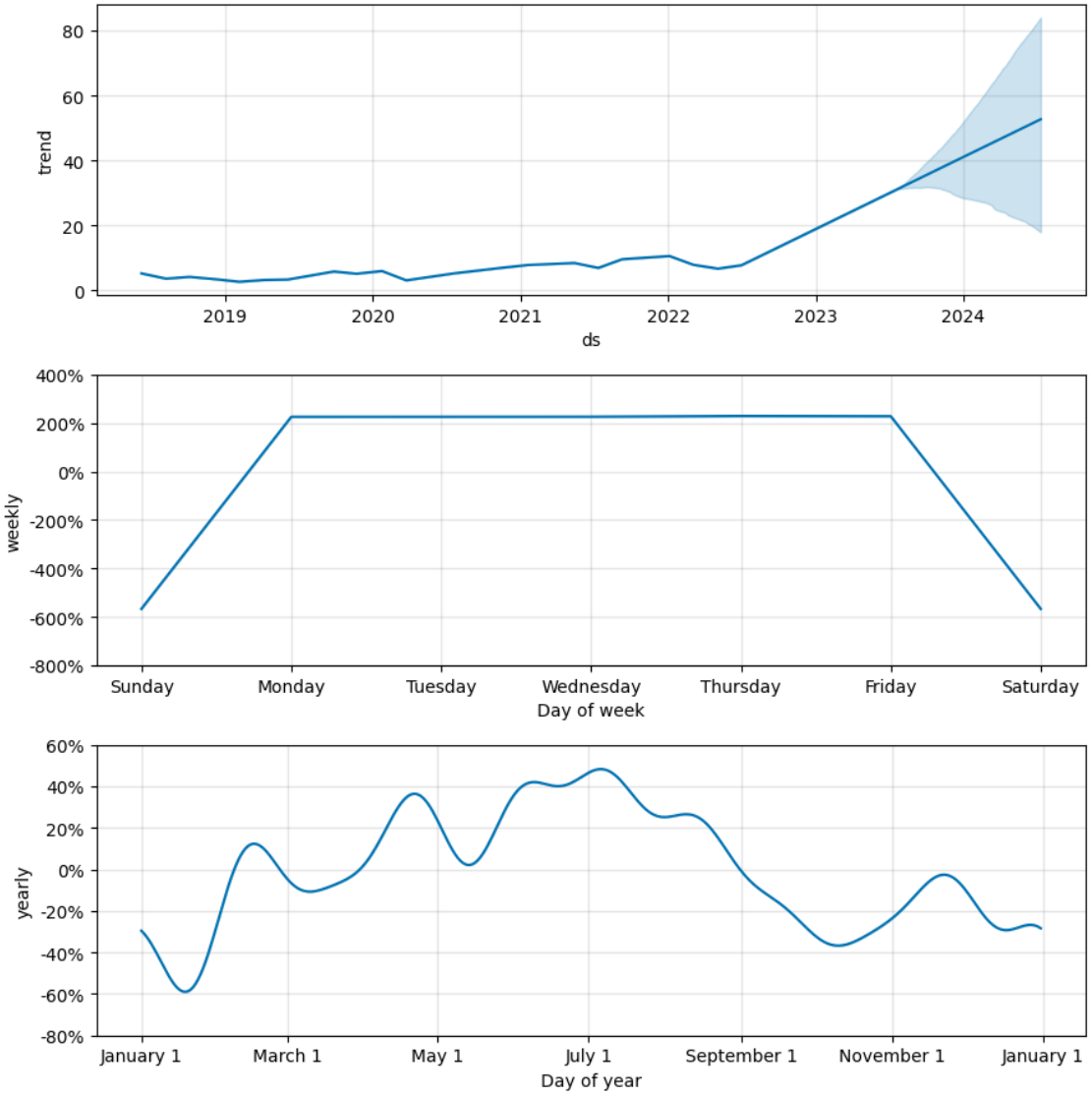


[186]:
```
from prophet import Prophet

# Crear el modelo y ajustarlo
# Personalizando el modelo lo arruinamos :(
modelo = Prophet(
    changepoint_prior_scale=0.1,
    seasonality_mode='multiplicative',
    yearly_seasonality=10,
    weekly_seasonality=True,
    interval_width=0.95
)
modelo.fit(df['ELF_Close'].reset_index().rename(columns={'Date': 'ds',␣
 ↪'ELF_Close': 'y'}))
```

```
# Predicción para los próximos 365 días
futuro = modelo.make_future_dataframe(periods=365)
predicciones = modelo.predict(futuro)

# Visualizar las predicciones
fig = modelo.plot(predicciones)
plt.plot(df['ELF_Close'], label='Serie Original', color='purple')
plt.xlabel('Fecha')
fig = modelo.plot_components(predicciones)
```

```
INFO:prophet:Disabling daily seasonality. Run prophet with
daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmp9ivjos63/7u4izzub.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmp9ivjos63/t8vmpk8o.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-
packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=83832', 'data',
'file=/tmp/tmp9ivjos63/7u4izzub.json', 'init=/tmp/tmp9ivjos63/t8vmpk8o.json',
'output',
'file=/tmp/tmp9ivjos63/prophet_model365s0tc2/prophet_model-20241122200909.csv',
'method=optimize', 'algorithm=lbfgs', 'iter=10000']
20:09:09 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
20:09:11 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```





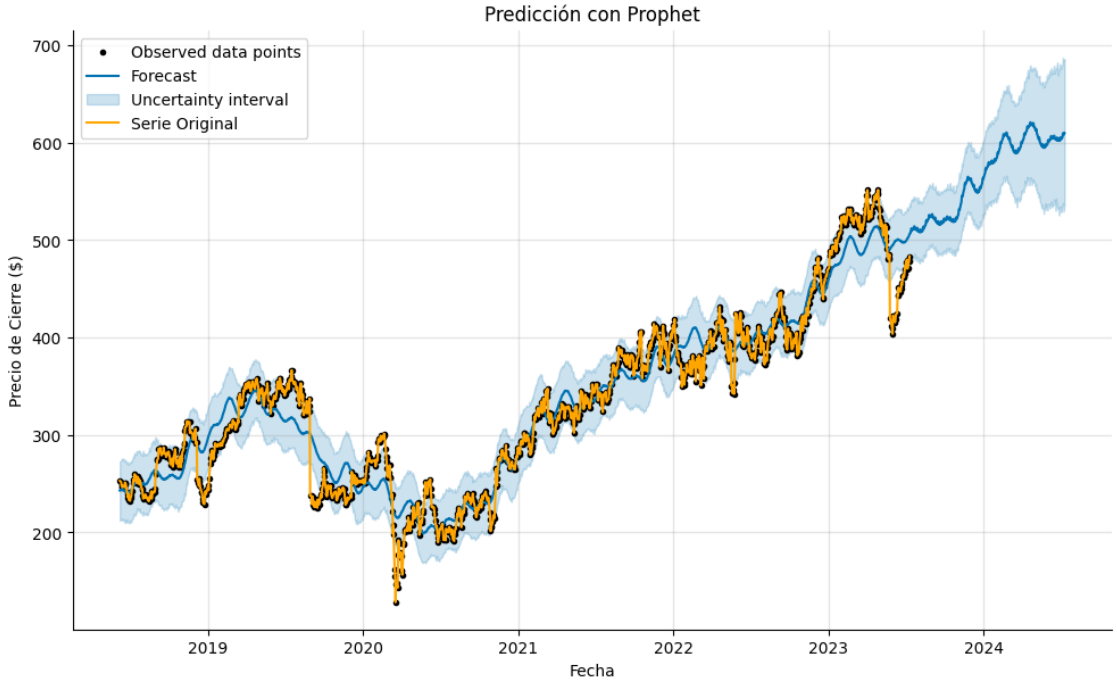
[ ]:
```
!pip install prophet
```

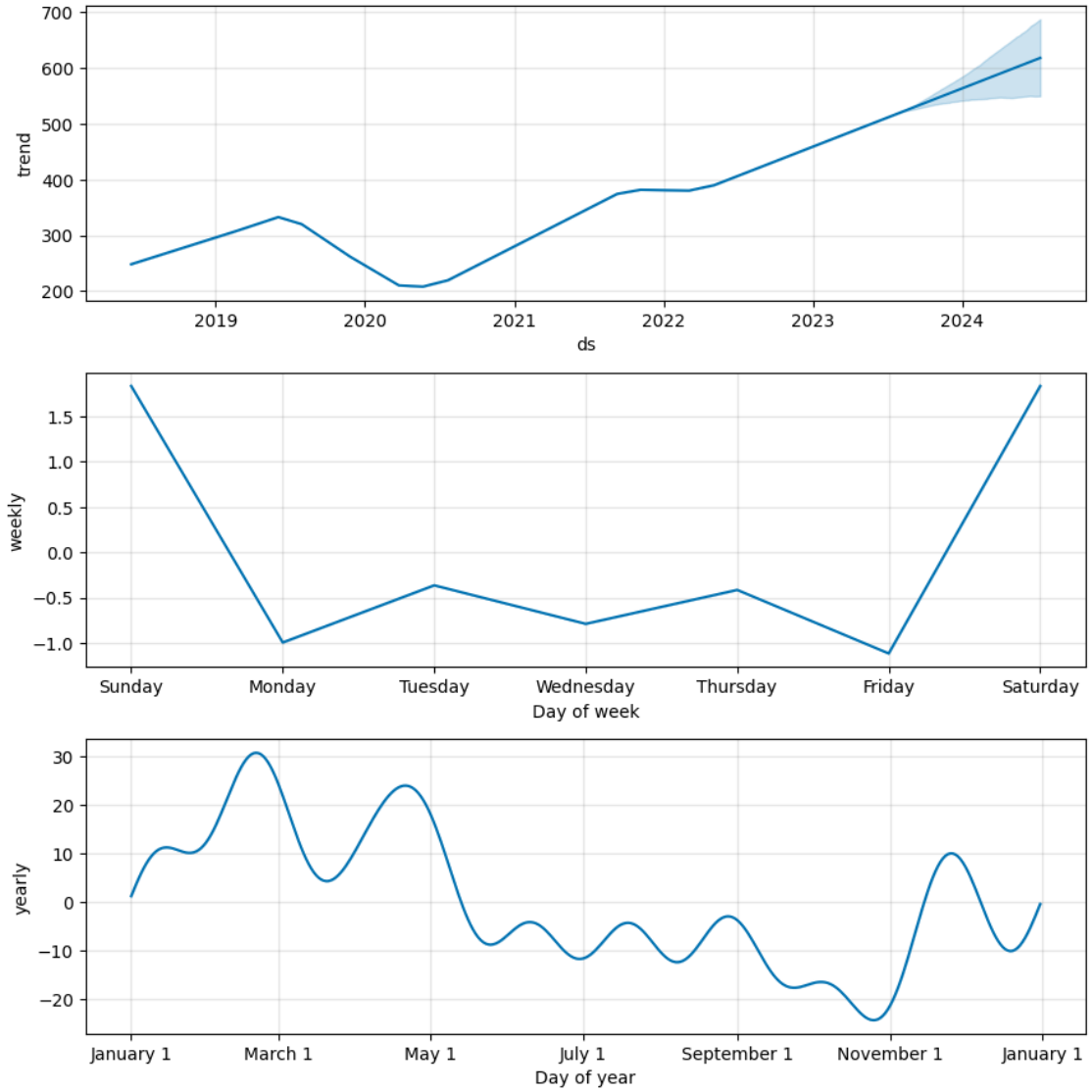
MARCA ULTA

[191]:
```
from prophet import Prophet

# Crear el modelo y ajustarlo
modelo = Prophet()
modelo.fit(df['ULTA_Close'].reset_index().rename(columns={'Date': 'ds',␣
 ↪'ULTA_Close': 'y'}))
```

```
# Predicción para los próximos 365 días
futuro = modelo.make_future_dataframe(periods=365)
predicciones = modelo.predict(futuro)

# Visualizar las predicciones
fig = modelo.plot(predicciones)
plt.plot(df['ULTA_Close'], label='Serie Original', color='orange')
plt.xlabel('Fecha')
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.title('Predicción con Prophet')
plt.ylabel('Precio de Cierre ($)')
plt.legend()
fig = modelo.plot_components(predicciones)
```

```
INFO:prophet:Disabling daily seasonality. Run prophet with
daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmp9ivjos63/y064nrzq.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmp9ivjos63/t_h4eg3e.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-
packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=27709', 'data',
'file=/tmp/tmp9ivjos63/y064nrzq.json', 'init=/tmp/tmp9ivjos63/t_h4eg3e.json',
'output',
'file=/tmp/tmp9ivjos63/prophet_modeli4i7hoz9/prophet_model-20241122203120.csv',
'method=optimize', 'algorithm=lbfgs', 'iter=10000']
20:31:20 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
20:31:20 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```

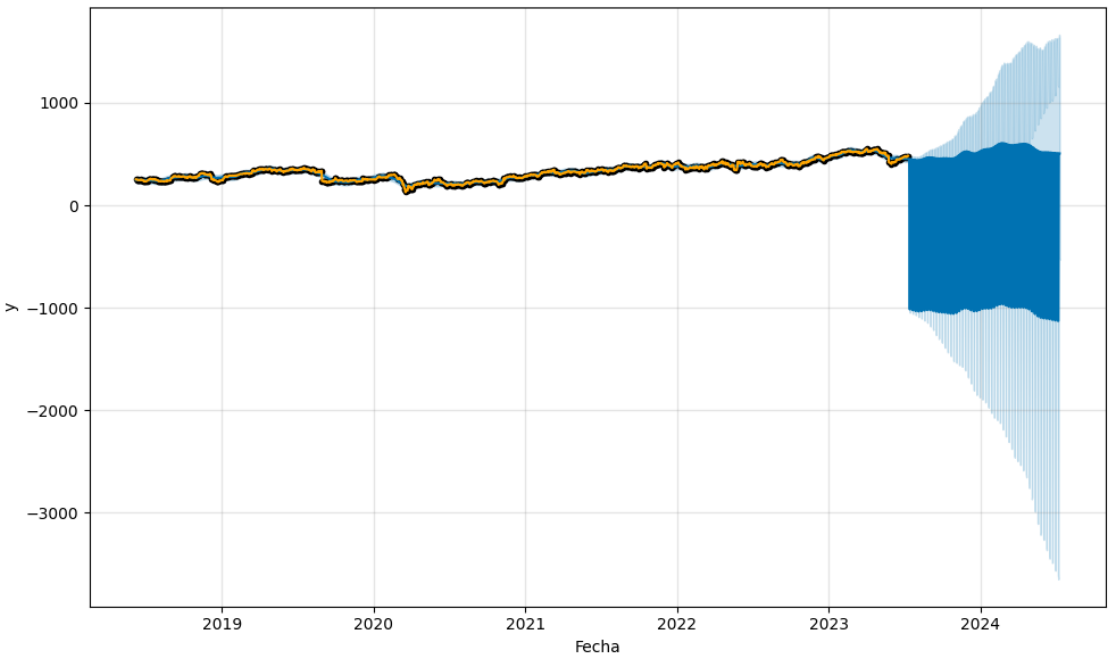

Predicción con Prophet
Observed data points
Forecast
Uncertainty interval
Serie Original
Precio de Cierre ($)
700
600
500
400
300
200
2019
2020
2021
2022
2023
2024
Fecha

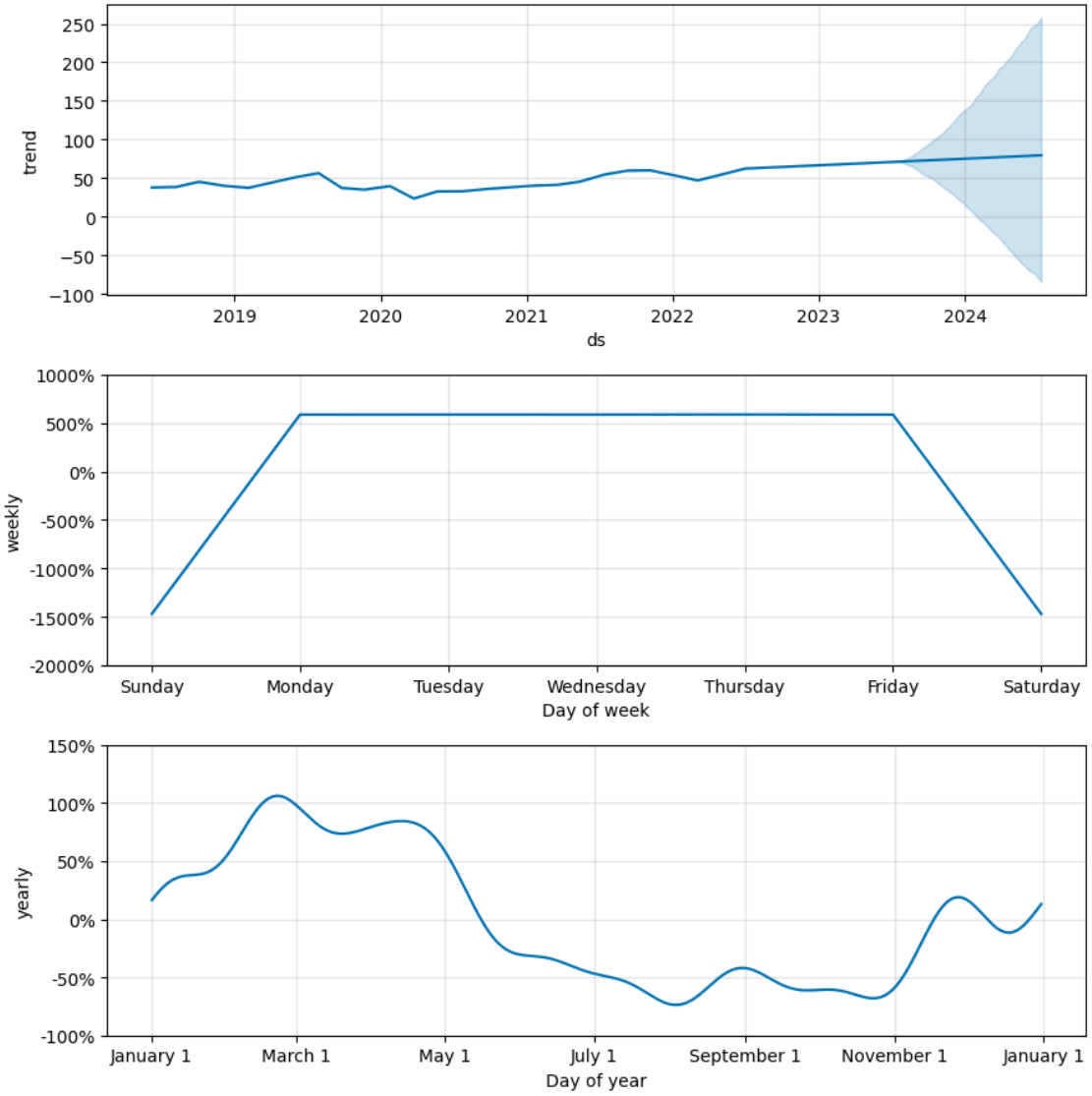


[192]:
```
from prophet import Prophet

# Crear el modelo y ajustarlo
# Personalizando el modelo lo arruinamos :(
modelo = Prophet(
    changepoint_prior_scale=0.1,
    seasonality_mode='multiplicative',
    yearly_seasonality=10,
    weekly_seasonality=True,
    interval_width=0.95
)
modelo.fit(df['ULTA_Close'].reset_index().rename(columns={'Date': 'ds',␣
 ↪'ULTA_Close': 'y'}))
```

```
# Predicción para los próximos 365 días
futuro = modelo.make_future_dataframe(periods=365)
predicciones = modelo.predict(futuro)

# Visualizar las predicciones
fig = modelo.plot(predicciones)
plt.plot(df['ULTA_Close'], label='Serie Original', color='orange')
plt.xlabel('Fecha')
fig = modelo.plot_components(predicciones)
```

```
INFO:prophet:Disabling daily seasonality. Run prophet with
daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmp9ivjos63/riecbkuh.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmp9ivjos63/u4oqhwvs.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.10/dist-
packages/prophet/stan_model/prophet_model.bin', 'random', 'seed=66047', 'data',
'file=/tmp/tmp9ivjos63/riecbkuh.json', 'init=/tmp/tmp9ivjos63/u4oqhwvs.json',
'output',
'file=/tmp/tmp9ivjos63/prophet_modelxrnvfxoe/prophet_model-20241122203146.csv',
'method=optimize', 'algorithm=lbfgs', 'iter=10000']
20:31:46 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
20:31:49 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```





1 RESULTADOS

- Principalmente para ambas marca usamos los mismos códigos para realizar la serie de tiempo, por lo que incertamos del dataframe esto con datos importados desde yahoo finance haciendoles los ajustes necesarios para poder hacer uso de la información
- Despues calculamos los promedios moviles de los precios de cierre y nos ayuda a vizualisarlos mediante una grafica para poder ver como se comportan los precios de cierre y las tendencias suavizadas.
- Luego de ello aplicamos la prueba ADF en ambas marcas la cual realiza una prueba de

estacionariedad sobre la serie temporal y aí de esta manera saber si se rechaza o no la hipotesis nula.

- Luego pasamos a calcular un modelo polinómico de grado 5 ajustando el modelo esto para que con el resultado obtenido en la grafica podamos ver los precios originales y la tendencia ajustada y de igual forma nos ayududó a calcular los residuos de ambas series temporales.
- Despues de eso calculamos los residuos de ambas series de la regresión polinomial ajustada pues los residuos representan la diferencia entre los valores reales y la tendencia predicha por el modelo polinómico y con los resultados nos ayuda a visualizar si dichos residuos estan distribuidos aleatoriamente.
- En el siguiente punto de ambas series calculamos la descomposición estacional de la serie temporal con sus componentes que son tendencia, estacionalidad y residuales esto para que con el resultado que nos arrojó podamos entender mejor la estructura de los datos.
- Volvemos a aplicar la prueba ADF pero ahora sobre los residuos de la descomposición estacional de la serie temporal, lo cual con ambos resultados podemos deducír si la serie temporal es estacionaria o no en base si rechazamos o no la hipotesis nula.
- Con base en la prueba anterior aplicamos la t pareada esto entre dos series temporales para determinar si la estacionalidad tiene un efecto significativo sobre los datos de la serie temporal original, y con los resultados obtenidos podemos ver si existe diferencias significativas.
- Al calcular esto pasamos a graficar la descomposición, con el primer código nos ayuda a mostrar una gráfica de comparación entre la serie de tiempo original de precios de cierre.
- El siguiente código nos ayudó a analizar cómo la tendencia y la estacionalidad afectan la serie de tiempo original sin la interferencia de los errores o residuos.
- Por último el código final nos ayudo a descomponer las series y a observar más claros la tendencia a largo plazo y los residuos.
- Luego pasamos a aplicar la causalidad de granger que se definio al principio del trabajo ajustando los datos, al realizar la prueba de causalidad de Granger entre dos series temporales para determinar si una serie temporal puede predecir otra serie temporal, considerando los lags en los datos.
- Y finalmente calculamos el modelo prophet que esta nos ayuda en ambas series de ambas marcas para realizar una predicción de series temporales sobre los precios de cierre de las acciones de ULTA para los próximos 365 días.
- Para el ultimo codigo se uso para ajustar el modelo de Prophet para la serie temporal de los precios de cierre de las acciones de las dos marcas con algunas personalizaciones.

2 CONCLUSIÓN

Con base en los resultados de los datos de ambas marcas al principio podemos ver que con la prueba ADF en ambas series temporales no son estacionarias por lo que se acepta la hipotesis nula en ambas series, en los promedios moviles simples podemos ver que llevan una buena continuedad ya que no hay mucho ruido en la grafica y en la regresión polinomial se puede ver en ambas graficas que siguen un patron similar. Al calcular la tendencia en la marca de elf nos da como resultado que la tendencia no es estacionaria por lo que se hacepta la hipotesis nula, más sin embargo, en la

prueba de la tendencia de la marca Ulta como resultado la tendencia si es estacionaria por lo que no se acepta la hipotesis nula.

Finalmente para concluir con la prueba de la t pareada podemos deducir con los resultados de ambas series temporales que ambas tienen efecto significativo en las series.