proyecto-final

November 23, 2024

#*INTRODUCCIÓN*

Con este estudio se pretende analizar la relación estadísticas de ventas entre la marca de NIKE y las plataformas de Amazon para comprender el comportamiento de mercado, identificar tendencias y tomar decisiones estratégicas.

Amazon y Nike son dos líderes de la industria en sus respectivas áreas: Amazon en el comercio electrónico global y Nike en el sector deportivo. Entre 2017 y 2019, ambas empresas experimentaron transformaciones significativas impulsadas por innovaciones tecnológicas, cambios en los hábitos de consumo y estrategias comerciales que influyeron en sus resultados financieros

#*MARCO TEÓRICO*

El periodo entre 2017 y 2019 fue clave para entender la dinámica entre marcas globales y marketplaces digitales. Nike, líder mundial en el sector deportivo, y Amazon, el gigante del comercio electrónico, mantuvieron una relación comercial estratégica que concluyó oficialmente en 2019 debido a sus constantes sospechas de plagio sobre la marca. El análisis de esta relación permite explorar cómo el acceso a una plataforma masiva como Amazon afectó las ventas, el posicionamiento de marca y las decisiones estratégicas de Nike durante este periodo. El comercio electrónico experimentó un crecimiento exponencial durante estos años. Según datos de la Conferencia de las Naciones Unidas sobre Comercio y Desarrollo (UNCTAD), las ventas en línea representaron un 13.6% de las ventas minoristas globales en 2019, frente al 10.2% en 2017. Amazon fue un actor clave en esta expansión, atrayendo a marcas tradicionales que buscaban aprovechar su alcance global. Nike en busca de un fortalecimiento en sus nuevas estrategias de venta llegado el inicio de la era digital y la expansión del comercio electrónico decidió relacionar la marca con la corporación de Amazon, sin mencionar la disminuchión del mercado gris de productos nike en la plataforma de Amazon y algunas otras; estrategia que funcionó de manera casi inmediata, dado que ya en el 2019 se reconoció un aumento de al menos el 7% sollo en sus ventas digitales. Sin embargo, en 2019 la marca decidió teminar la relación debido a la alta cantidad de falsificación de sus productos y la baja rentabilidad de los términos, fue entonces que contempló la idea de generar sus procios servicios de vetas directas (B2C) y algunos otros intermediarios minoristas. Decisión que favoreció a su mercado pues a finales de 2019 se detectó un crecimiento del 15%. Mientras que Amazon, al considerar un impulso a su plataforma al establecer una relación estrategica con una marca grande y prestigiosa como NIKE y otras marcas, incrementaría su nivel de flujo en ventas a nivel global, estrategia bien diseñada, dado que entre 2017 y 2019 según la Tasa de Crecimiento Anual Compuesto se registró un crecimiento alto de el 27.9% ($280.5 mil millones aproximadamente), siendo el 60% de las ventas totales de la marca mencionada; lo que significó una gran pérdida al finalizar su relación con NIKE a pesar de haberse posicionado como lider del mercado digital.

a) Hipótesis

Nike y Amazon:

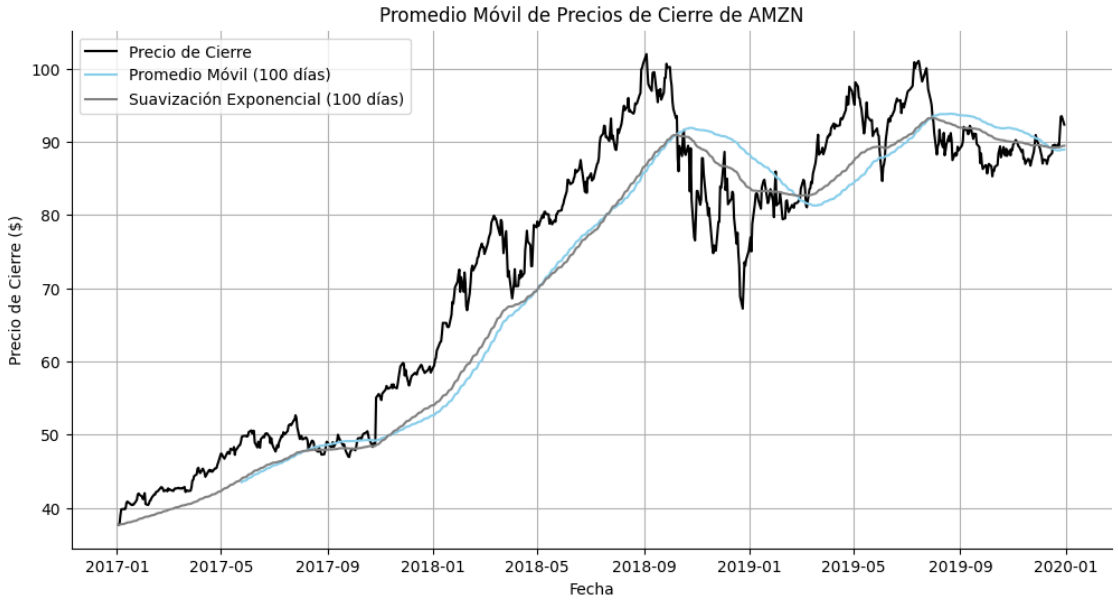
b)Análisis de cada serie de tiempo de manera independiente

[1]:
```
import yfinance as yf  #Esta línea importa la biblioteca yfinance, que permite␣
 ↪acceder a datos financieros de Yahoo Finance. Se le asigna el alias yf para␣
 ↪facilitar su uso posterior en el código.
import pandas as pd    #Aquí se importa la biblioteca pandas, que es␣
 ↪fundamental para el manejo y análisis de datos en Python. Se le asigna el␣
 ↪alias pd, lo que es una convención común para simplificar su uso.
import matplotlib.pyplot as plt   #Esta

from statsmodels.tsa.stattools import adfuller
from numpy.polynomial.polynomial import Polynomial
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from scipy.stats import pearsonr

import warnings
# Suprimir todos los warnings
warnings.filterwarnings('ignore')

df = yf.download( tickers= ["NKE","AMZN"],   # Esta línea inicia la descarga de␣
 ↪datos financieros y asigna el resultado a la variable df, que típicamente se␣
 ↪utiliza para almacenar un DataFrame de pandas. Este parámetro especifica el␣
 ↪símbolo del ticker que se va a descargar. En este caso, "MSFT" representa a␣
 ↪Microsoft.
                  start= "2017-01-01",   #YYYY= año en 4 digitos,  mm= mes con␣
 ↪2 digitos,  dd= dia en 2 digitos  (YYYY-mm-dd).  Define la fecha de inicio␣
 ↪para la descarga de datos. En este caso, los datos comenzarán desde el 1 de␣
 ↪enero de 2015.
                  end= "2019-12-31",     #Establece la fecha de finalización␣
 ↪para los datos descargados. Aquí, se están obteniendo datos hasta el 27 de␣
 ↪octubre de 2024.
                  interval= "1d",      # Se va a trabajar con un dia.  Este␣
 ↪parámetro establece el intervalo de tiempo entre las entradas de datos. "1d"␣
 ↪significa que se obtendrán datos diarios.
                  group_by= None,      #Este parámetro especifica cómo se␣
 ↪agruparán los datos. Al establecerlo en None, los datos no se agruparán por␣
 ↪tickers.
                  auto_adjust= False,  #Define si los datos deben ajustarse␣
 ↪automáticamente para dividendos y splits. Con False, se obtienen los datos␣
 ↪sin ajustar.
```

```
                    actions= False       #Este parámetro indica si se deben␣
 ↪incluir acciones corporativas (como dividendos y splits) en los datos. Con␣
 ↪False, no se incluirán.
                    )

# Paso 1: Mover 'Date' del índice a columna regular
df = df.reset_index()      #para restablecer el índice de un DataFrame contexto

# 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]  #se utiliza para modificar los nombres de las columnas de␣
 ↪un DataFrame en pandas, especialmente cuando se tienen columnas con␣
 ↪múltiples niveles

#convertir la columna "date" a tipo datetime si no lo es ya
df['Date'] = pd.to_datetime(df['Date'])  #se utiliza para convertir una columna␣
 ↪de fechas en un DataFrame de pandas, que inicialmente está en formato de␣
 ↪texto, a un tipo de dato

df['Date'] = df['Date'].dt.date  #se eliminará la información de tiempo (hora,␣
 ↪minutos, segundos) y se mantendrá solo la parte de la fecha (año, mes, día)

####################
df.set_index("Date",inplace=True)

# Promedio Móvil Simple
ventana_sma = 100      #media se calculará tomando en cuenta los últimos 100␣
 ↪valores de la columna
df['SMA'] = df['AMZN_Close'].rolling(window=ventana_sma).mean()  #la serie␣
 ↪temporal de precios, ayudando a identificar tendencias y patrones en los␣
 ↪datos al eliminar la volatilidad a corto plazo.

# Promedio Móvil Exponencial
ventana_ses = 100  # un parámetro flexible para definir un rango de␣
 ↪observaciones en diferentes cálculos o análisis en el contexto de series␣
 ↪temporales.
df['SES'] = df['AMZN_Close'].ewm(span=ventana_ses, adjust=False).mean()  #se␣
 ↪utiliza para calcular la media móvil exponencial (SES, por sus siglas en␣
 ↪inglés) de la columna

# Graficar
plt.figure(figsize=(12, 6))  #se utiliza para crear una figura con dimensiones␣
 ↪específicas, mejorando la presentación y la claridad de los gráficos
plt.plot(df['AMZN_Close'], label='Precio de Cierre', color='black')
```

```
plt.plot(df['SMA'], label=f'Promedio Móvil ({ventana_sma} días)',␣
 ↪color='skyblue')
plt.plot(df['SES'], label=f'Suavización Exponencial ({ventana_ses} días)',␣
 ↪color='gray')
plt.title('Promedio Móvil de Precios de Cierre de AMZN')
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()
```

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



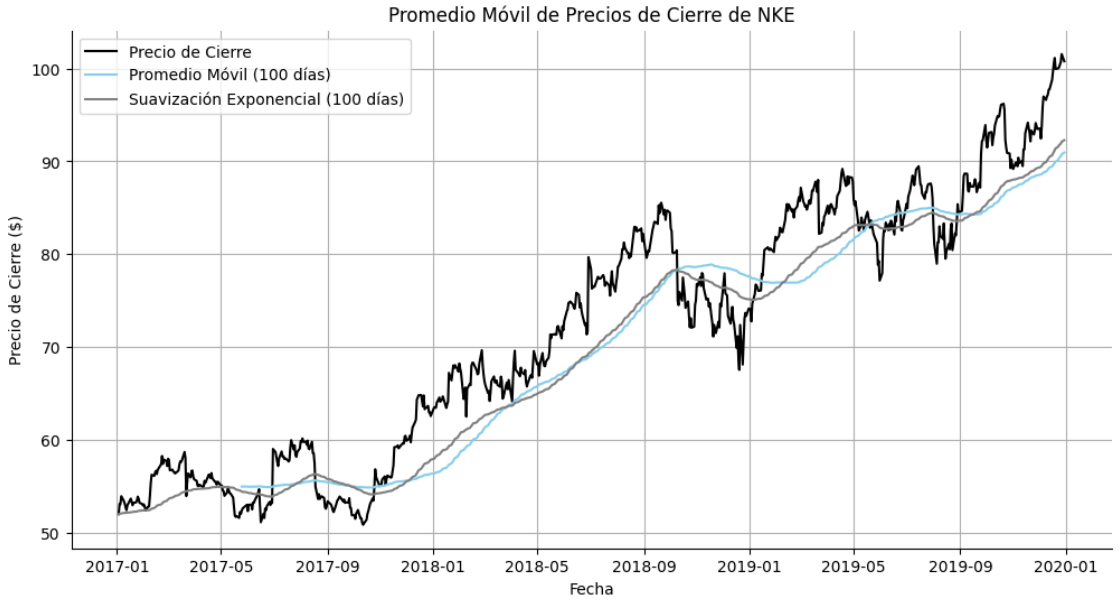
Interpretación:

El gráfico se percibe una tendencia alcista general en el precio de las acciones de Amazon durante el período analizado.Esto indica que, a largo plazo, la compañía ha experimentado un crecimiento significativo en su valor de mercado. Sin embargo, es importante destacar que el precio de las acciones ha estado sujeto a fluctuaciones considerables a corto plazo, lo cual es común en los mercados financieros.

Por lo tanto, Amazon ha demostrado ser una inversión atractiva durante el período analizado, reflejando el crecimiento y la solidez de la compañía en el mercado.

Nike

[2]:
```
# Promedio Móvil Simple
ventana_sma = 100      #media se calculará tomando en cuenta los últimos 100␣
 ↪valores de la columna
df['SMA'] = df['NKE_Close'].rolling(window=ventana_sma).mean()  #la serie␣
 ↪temporal de precios, ayudando a identificar tendencias y patrones en los␣
 ↪datos al eliminar la volatilidad a corto plazo.

# Promedio Móvil Exponencial
ventana_ses = 100  # un parámetro flexible para definir un rango de␣
 ↪observaciones en diferentes cálculos o análisis en el contexto de series␣
 ↪temporales.
df['SES'] = df['NKE_Close'].ewm(span=ventana_ses, adjust=False).mean()  #se␣
 ↪utiliza para calcular la media móvil exponencial (SES, por sus siglas en␣
 ↪inglés) de la columna

# Graficar
plt.figure(figsize=(12, 6))  #se utiliza para crear una figura con dimensiones␣
 ↪específicas, mejorando la presentación y la claridad de los gráficos
plt.plot(df['NKE_Close'], label='Precio de Cierre', color='black')
plt.plot(df['SMA'], label=f'Promedio Móvil ({ventana_sma} días)',␣
 ↪color='skyblue')
plt.plot(df['SES'], label=f'Suavización Exponencial ({ventana_ses} días)',␣
 ↪color='gray')
plt.title('Promedio Móvil de Precios de Cierre de NKE')
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()
```



Interpretacion

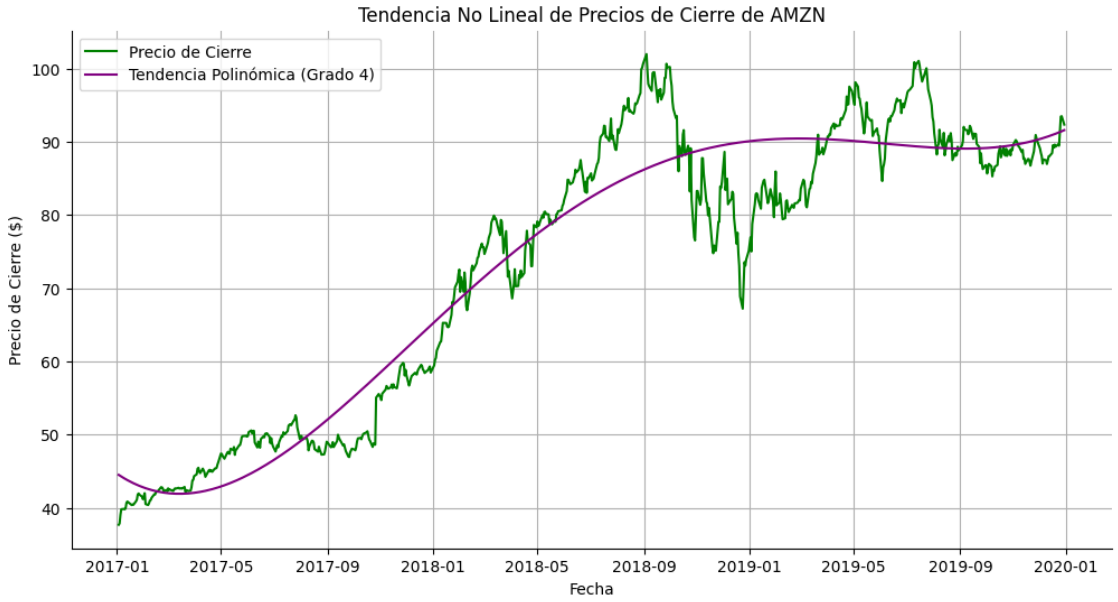
En esta grafica se muestra el promedio movil de los precios de cierre de nike (NKE), en esta se puede observar que la tendencia de la grafica va a la alza, a demas del promedio movil de 100 dias que de igual manera tambien va a la alza, debido a esto se podria decir que este promedio movil me puede ayudar a identificar esta tendendcia y estar mas seguro de la misma.

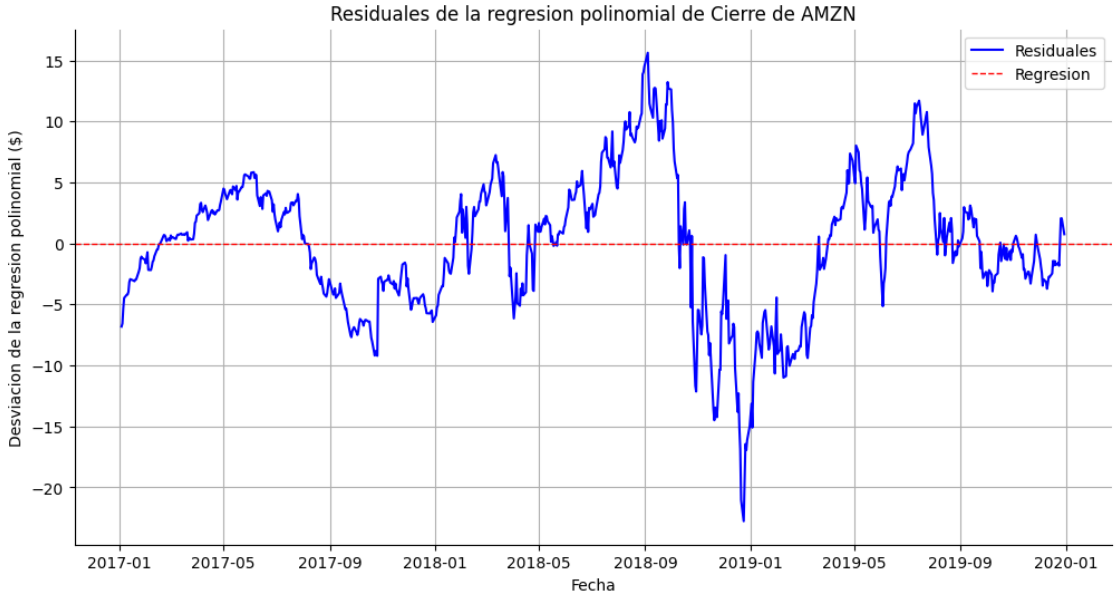
Regresión polinomial y residuales

Amazon

[3]:
```
# Paso 1: Convertir fechas a números
# Creamos una nueva columna llamada 'Date' la cual contiene las fechas en
# segundos desde el 1 de de enero de 1970 (tiempo UNIX)
df['Date'] = pd.to_datetime(df.index).map(pd.Timestamp.timestamp)

# Paso 2: Definir las variables independientes (X) y la variable dependiente (y)
X = df['Date']
y = df['AMZN_Close']

# Paso 3: Ajustar un modelo polinómico
grado = 4
modelo = Polynomial.fit(X, y, deg = grado)
df['Poly_Trend'] = modelo(X)
df['Poly_Resid'] = y - df['Poly_Trend']

# Graficar polinomial
# abre una nueva ventana gráfica
plt.figure(
```

```
    figsize=(12, 6) # tamaño (horizontal, vertical)
    )

# qué se va a graficar:
plt.plot(
    y,
    label='Precio de Cierre', # etiqueta que aparecerá en la leyenda
    color='green'
    )

plt.plot(df['Poly_Trend'], label=f'Tendencia Polinómica (Grado {grado})',␣
 ↪color='purple')
plt.title('Tendencia No Lineal de Precios de Cierre de AMZN') # Título
plt.xlabel('Fecha')                                          # nombre del eje x
plt.ylabel('Precio de Cierre ($)')                           # nombre del eje y
plt.legend()                                                 # agrega la␣
 ↪leyenda
plt.grid()                                                   # agrega una␣
 ↪cuadrícula
plt.gca().spines['top'].set_visible(False)                   # elimina los␣
 ↪bordes superiores
plt.gca().spines['right'].set_visible(False)                 # elimina los␣
 ↪bordes derechos
plt.show()                                                   # muestra el␣
 ↪gráfico (innecesario en colab)



# Graficar los residuales del polinomio
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='--', linewidth=1)  #␣
 ↪Línea horizontal en el valor cero
plt.title('Residuales de la regresion polinomial de Cierre de AMZN')
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()
```





[4]:
```
# Aquí no le muevan a nada
# Básicamente, lo que se hace este bloque es crear la función
# Transformada_de_Fourier, puesto a que no existe para lo que la quiero usar.

def Transformada_de_Fourier(serie, terminos, un_grafico):
  '''La función acepta los argumentos "serie", la cual debe de ser un array
  en numpy y en pandas de la forma df["my_variable"].values. Por otra parte,
```

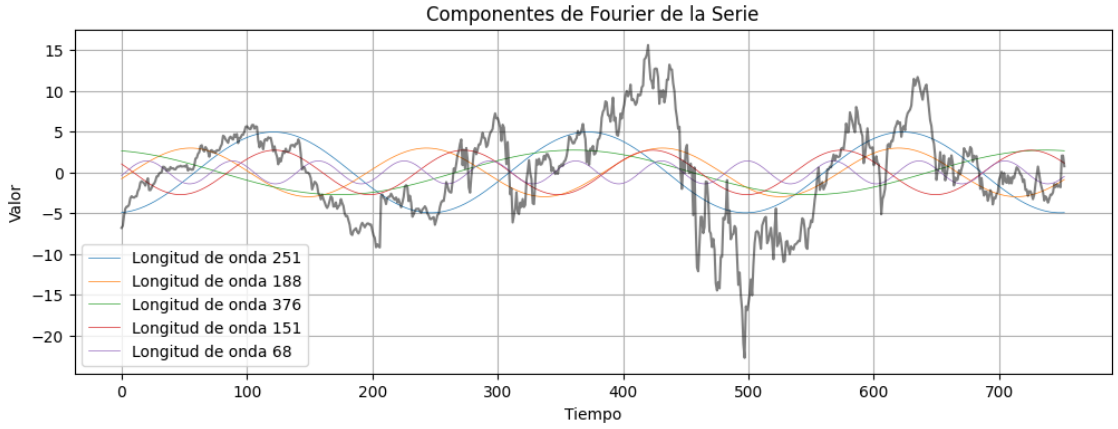
```
el parámetro "terminos" es un número natural que indica la cantidad de
términos que desarrolla la serie. Entre mayor sea el número de términos,
la serie será más precisa, pero más difícil de interpretar.

La función tiene como salida una lista con los componentes sinosoidales de la serie.
El último elemento de la lista, es la suma de todos los componentes'''

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.fftpack import fft, ifft

# Supongamos que tienes una serie en df['AAPL_Close']
# Reemplaza esta línea con tu DataFrame y serie específicos
n = len(serie)

# Calcular la transformada de Fourier
transformada_fourier = fft(serie)

# Obtener las frecuencias
frecuencias = np.fft.fftfreq(n)

# Crear un DataFrame para almacenar la frecuencia y su magnitud
componentes = pd.DataFrame({
    'frecuencia': frecuencias,
    'magnitud': np.abs(transformada_fourier),
    'longitud_de_onda': 1 / frecuencias
})

# Ordenar el dataframe de mayor a menor en términos de magnitud

componentes = componentes.sort_values(by='magnitud', ascending=False)

# Seleccionar las frecuencias con mayor magnitud (excluyendo la frecuencia cero)
top_frecuencias = componentes.loc[componentes['frecuencia'] > 0].nlargest(terminos, 'magnitud')
top_frecuencias.reset_index(drop=True, inplace=True)
print("Frecuencias principales:\n", top_frecuencias)

# Crear el índice de tiempo para la serie
t = np.arange(n)

# Graficar cada componente de frecuencia junto con la serie original
plt.figure(figsize=(12, 4))
```

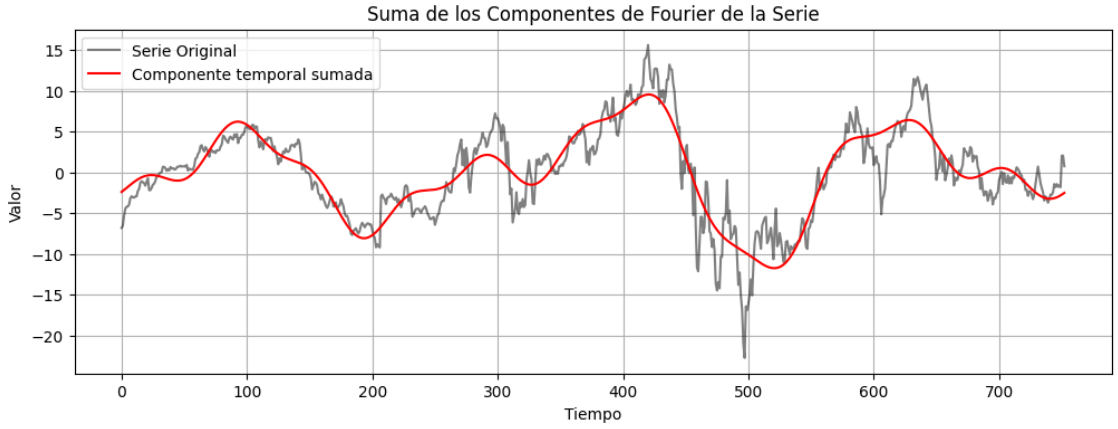
```
componente_temporal_sumado = np.zeros_like(serie)
componentes_temporales = []

n=0
for i, row in top_frecuencias.iterrows():
    n+=1
    # Copiar la transformada de Fourier y mantener solo la frecuencia actual
    fourier_component = np.zeros_like(transformada_fourier)
    idx = np.where(frecuencias == row['frecuencia'])[0][0]  # índice de la frecuencia en la FFT
    fourier_component[idx] = transformada_fourier[idx]  # mantener solo la frecuencia positiva
    fourier_component[-idx] = transformada_fourier[-idx]  # mantener la frecuencia negativa correspondiente
    if n == (terminos+1):
      break

    # Reconstruir la señal en el tiempo
    componente_temporal = ifft(fourier_component).real
    componentes_temporales.append(componente_temporal)
    componente_temporal_sumado += componente_temporal

    # Graficar la componente
    plt.plot(
        componente_temporal,
        label=f'Longitud de onda {1 / row["frecuencia"]:.0f}',
        alpha=1,
        linewidth = 0.5,
        )
    plt.title('Componentes de Fourier de la Serie')
    plt.xlabel('Tiempo')
    plt.ylabel('Valor')
    plt.legend()
    plt.grid()

componentes_temporales.append(componente_temporal_sumado)
plt.plot(serie, label='Serie Original', color='black', alpha=0.5)
if not un_grafico:
  plt.figure(figsize=(12, 4))
  plt.plot(serie, label='Serie Original', color='black', alpha=0.5)
plt.plot(componente_temporal_sumado, label='Componente temporal sumada', color='red')
plt.legend()
plt.title('Suma de los Componentes de Fourier de la Serie')
plt.xlabel('Tiempo')
plt.ylabel('Valor')
plt.grid()
```

```
  plt.show()

  return componentes_temporales
```

[5]:
```
serie = df['Poly_Resid'].values
terminos = 5
un_grafico = False

cts = Transformada_de_Fourier(serie, terminos, un_grafico)

# cts: componente temporal sumado
```

```
Frecuencias principales:
    frecuencia     magnitud  longitud_de_onda
0     0.003984  1867.619351        251.000000
1     0.005312  1123.900267        188.250000
2     0.002656  1030.919017        376.500000
3     0.006640  1027.448618        150.600000
4     0.014608   530.500223         68.454545
```





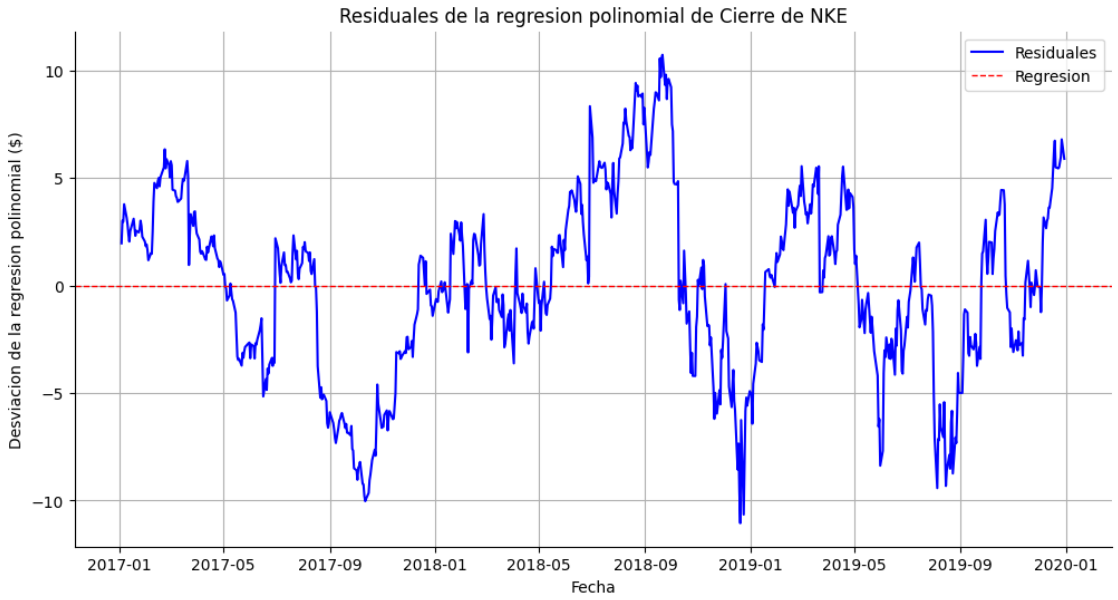
Interpretación

En las graficas se muestran la descomposición de una señal en sus componentes de frecuencia. Cada línea representa una onda sinusoidal con una frecuencia específica que contribuye a formar la señal original. Las diferentes longitudes de onda indican diferentes frecuencias presentes en la señal. Se muestra la serie original (en gris) y la suma de los componentes de Fourier (en rojo). La línea roja representa la aproximación de la serie original utilizando las componentes de Fourier calculadas.

Nike

[6]:
```
# Paso 1: Convertir fechas a números
# Creamos una nueva columna llamada 'Date' la cual contiene las fechas en
# segundos desde el 1 de de enero de 1970 (tiempo UNIX)
df['Date'] = pd.to_datetime(df.index).map(pd.Timestamp.timestamp)

# Paso 2: Definir las variables independientes (X) y la variable dependiente (y)
X = df['Date']
y = df['NKE_Close']

# Paso 3: Ajustar un modelo polinómico
grado = 2
modelo = Polynomial.fit(X, y, deg = grado)
df['Poly_Trend'] = modelo(X)
df['Poly_Resid'] = y - df['Poly_Trend']

# Graficar polinomial
# abre una nueva ventana gráfica
plt.figure(
    figsize=(12, 6) # tamaño (horizontal, vertical)
    )

# qué se va a graficar:
plt.plot(
    y,
    label='Precio de Cierre', # etiqueta que aparecerá en la leyenda
    color='green'
    )

plt.plot(df['Poly_Trend'], label=f'Tendencia Polinómica (Grado {grado})',␣
 ↪color='purple')
plt.title('Tendencia No Lineal de Precios de Cierre de NKE') # Título
plt.xlabel('Fecha')                                          # nombre del eje x
plt.ylabel('Precio de Cierre ($)')                           # nombre del eje y
plt.legend()                                                 # agrega la␣
 ↪leyenda
```

```
plt.grid()                                                    # agrega una␣
 ↪cuadrícula
plt.gca().spines['top'].set_visible(False)                    # elimina los␣
 ↪bordes superiores
plt.gca().spines['right'].set_visible(False)                  # elimina los␣
 ↪bordes derechos
plt.show()                                                    # muestra el␣
 ↪gráfico (innecesario en colab)



# Graficar los residuales del polinomio
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='--', linewidth=1)  #␣
 ↪Línea horizontal en el valor cero
plt.title('Residuales de la regresion polinomial de Cierre de NKE')
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()
```



Interpretacion

Para la grafica de tendencia no lineal para los precios de cierre de Nive se puede observar que la tendencia polinómica sigue de manera general la dirección ascendente de los precios, aunque no ajusta perfectamente todos los movimientos de corto plazo.

En el grafico de residuales para la regresion polinomial se observa que los residuales fluctúan alrededor de cero, pero no parecen distribuirse aleatoriamente, se puede observar que hay patrones evidentes de agrupación, lo que podría indicar autocorrelación.

[7]:
```
# Aquí no le muevan a nada
# Básicamente, lo que se hace este bloque es crear la función
# Transformada_de_Fourier, puesto a que no existe para lo que la quiero usar.

def Transformada_de_Fourier(serie, terminos, un_grafico):
  '''La función acepta los argumentos "serie", la cual debe de ser un array
  en numpy y en pandas de la forma df["my_variable"].values. Por otra parte,
  el parámetro "terminos" es un número natural que indica la cantidad de
  términos que desarrolla la serie. Entre mayor sea el número de términos,
  la serie será más precisa, pero más difícil de interpretar.

  La función tiene como salida una lista con los componentes sinosoidales de␣
 ↪serie.
  El último elemento de la lista, es la suma de todos los componentes'''

  import numpy as np
  import matplotlib.pyplot as plt
  import pandas as pd
```

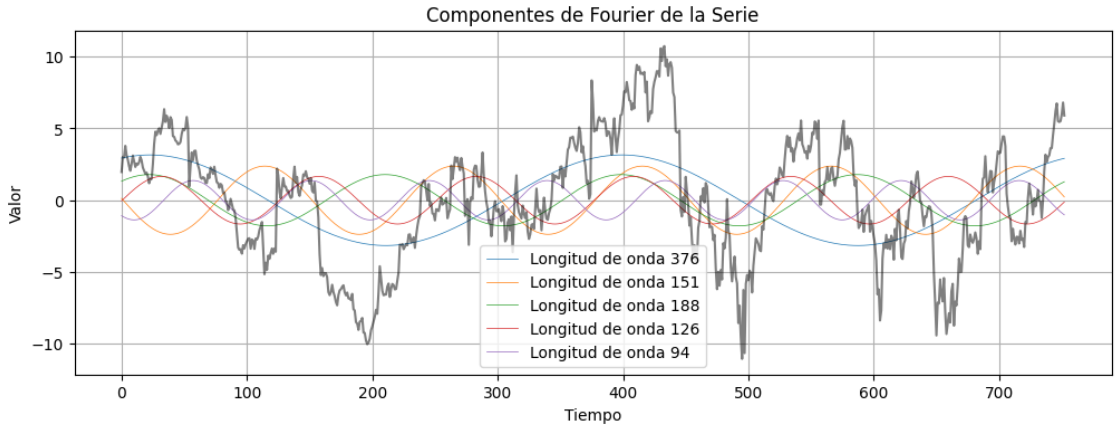
```python
from scipy.fftpack import fft, ifft

# Supongamos que tienes una serie en df['AAPL_Close']
# Reemplaza esta línea con tu DataFrame y serie específicos
n = len(serie)

# Calcular la transformada de Fourier
transformada_fourier = fft(serie)

# Obtener las frecuencias
frecuencias = np.fft.fftfreq(n)

# Crear un DataFrame para almacenar la frecuencia y su magnitud
componentes = pd.DataFrame({
    'frecuencia': frecuencias,
    'magnitud': np.abs(transformada_fourier),
    'longitud_de_onda': 1 / frecuencias
})

# Ordenar el dataframe de mayor a menor en términos de magnitud

componentes = componentes.sort_values(by='magnitud', ascending=False)

# Seleccionar las frecuencias con mayor magnitud (excluyendo la frecuencia␣
↪cero)
top_frecuencias = componentes.loc[componentes['frecuencia'] > 0].
↪nlargest(terminos, 'magnitud')
top_frecuencias.reset_index(drop=True, inplace=True)
print("Frecuencias principales:\n", top_frecuencias)

# Crear el índice de tiempo para la serie
t = np.arange(n)

# Graficar cada componente de frecuencia junto con la serie original
plt.figure(figsize=(12, 4))

componente_temporal_sumado = np.zeros_like(serie)
componentes_temporales = []

n=0
for i, row in top_frecuencias.iterrows():
    n+=1
    # Copiar la transformada de Fourier y mantener solo la frecuencia actual
    fourier_component = np.zeros_like(transformada_fourier)
    idx = np.where(frecuencias == row['frecuencia'])[0][0]  # índice de la␣
↪frecuencia en la FFT
```

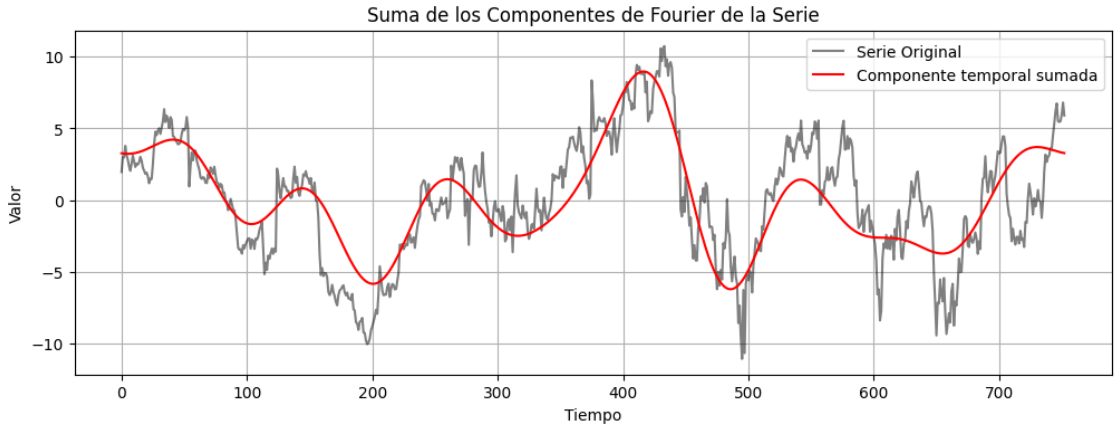
```
        fourier_component[idx] = transformada_fourier[idx]  # mantener solo la␣
↪frecuencia positiva
        fourier_component[-idx] = transformada_fourier[-idx]  # mantener la␣
↪frecuencia negativa correspondiente
        if n == (terminos+1):
          break

        # Reconstruir la señal en el tiempo
        componente_temporal = ifft(fourier_component).real
        componentes_temporales.append(componente_temporal)
        componente_temporal_sumado += componente_temporal

        # Graficar la componente
        plt.plot(
            componente_temporal,
            label=f'Longitud de onda {1 / row["frecuencia"]:.0f}',
            alpha=1,
            linewidth = 0.5,
            )
        plt.title('Componentes de Fourier de la Serie')
        plt.xlabel('Tiempo')
        plt.ylabel('Valor')
        plt.legend()
        plt.grid()

    componentes_temporales.append(componente_temporal_sumado)
    plt.plot(serie, label='Serie Original', color='black', alpha=0.5)
    if not un_grafico:
      plt.figure(figsize=(12, 4))
      plt.plot(serie, label='Serie Original', color='black', alpha=0.5)
    plt.plot(componente_temporal_sumado, label='Componente temporal sumada',␣
↪color='red')
    plt.legend()
    plt.title('Suma de los Componentes de Fourier de la Serie')
    plt.xlabel('Tiempo')
    plt.ylabel('Valor')
    plt.grid()
    plt.show()

    return componentes_temporales
```

[8]:
```
serie = df['Poly_Resid'].values
terminos = 5
un_grafico = False

cts = Transformada_de_Fourier(serie, terminos, un_grafico)
```

```
# cts: componente temporal sumado
```

```
Frecuencias principales:
    frecuencia     magnitud  longitud_de_onda
0     0.002656  1188.883299           376.500
1     0.006640   895.222034           150.600
2     0.005312   672.771457           188.250
3     0.007968   624.253606           125.500
4     0.010624   516.570434            94.125
```





Interpretacion

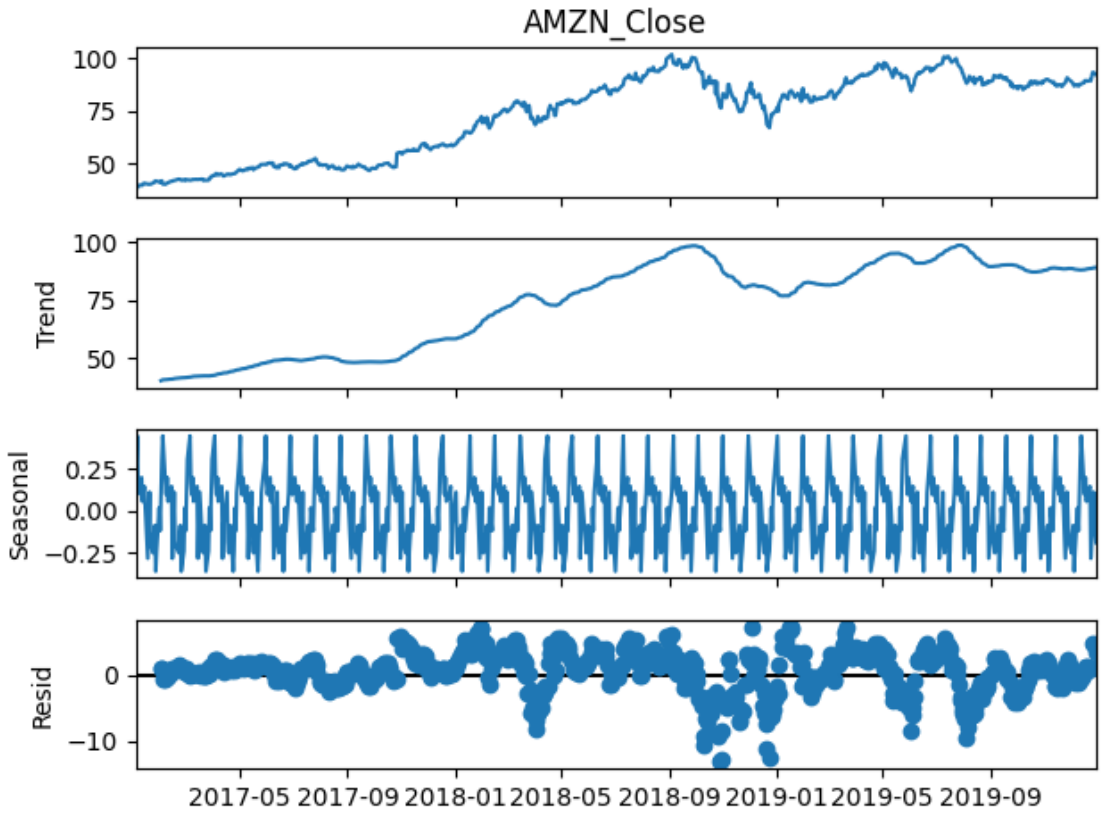
En este caso, la reconstrucción con componentes seleccionadas (línea roja) sigue bastante bien las tendencias principales de la serie, pero omite el ruido y las fluctuaciones rápidas. Esto sugiere que los componentes principales capturan la estructura esencial de la serie, siendo útiles para el análisis de tendencias o patrones de largo plazo, mientras que los componentes de alta frecuencia pueden representar ruido o variaciones menores.

Las componentes con longitudes de onda mayores (como 376) capturan tendencias de largo plazo, mientras que las componentes con longitudes de onda menores (como 94) reflejan oscilaciones más rápidas, probablemente asociadas al ruido o variaciones locales.

Descompón la serie de tiempo en sus componentes: tendencia, estacionalidad y ruido.

Amazon

[9]:
```
import statsmodels.api as sm
#Descomponer la serie temporal
descomposicion=sm.tsa.seasonal_decompose(
    df["AMZN_Close"],
    model="additive",
    period=20,
    extrapolate_trend=0,
    two_sided=False,
    )  #252 dias de negociacion al año

fig=descomposicion.plot()
plt.show()
```



Interpretación:

Serie original (AMZN_Close): Es la serie original del precio de cierre, en donde se pueden observar la fluctuación del precio.

Tendencia (Trend):Representa la tendencia subyacente de la serie, en donde las acciones de Amazon ha estado aumentando en promedio a lo largo del tiempo.

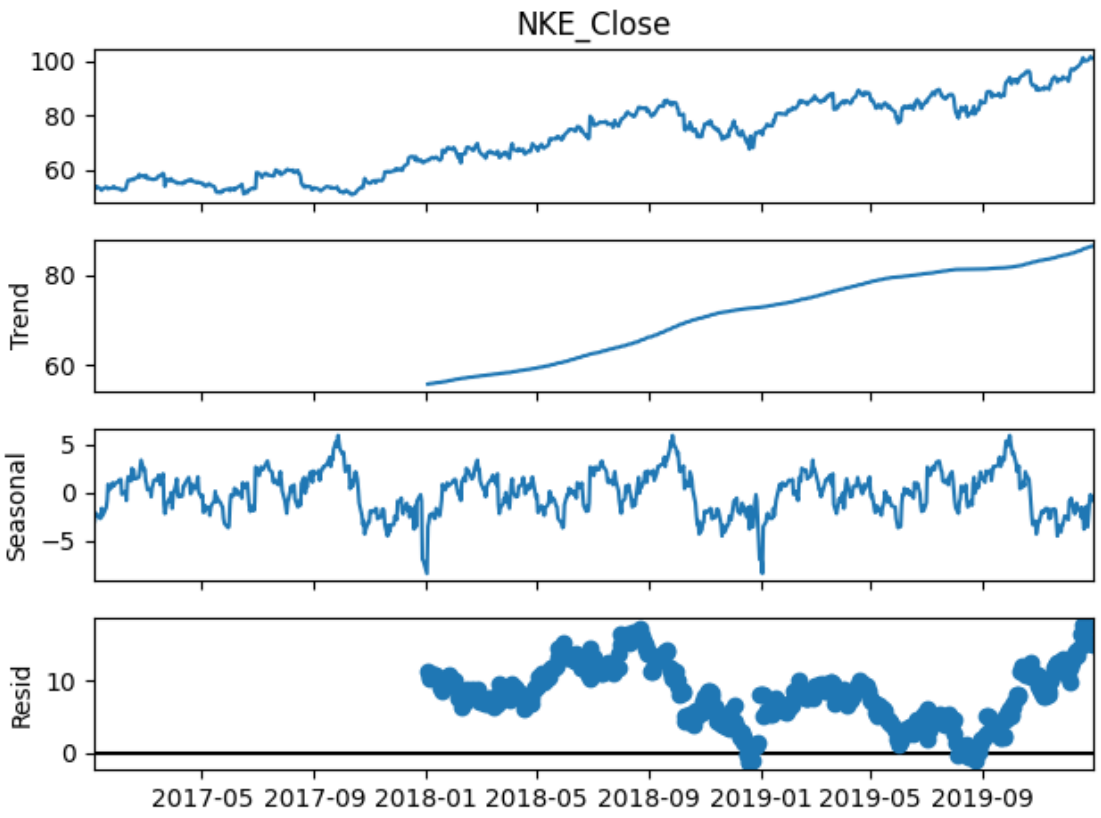
Estacionalidad (Seasonal): Se observa un patrón estacional pronunciado, lo que sugiere que el precio de las acciones de Amazon puede estar influenciado por factores estacionales, como eventos específicos que ocurren cada año.

Residuales (Residual): Los residuos representan la parte aleatoria o no explicada de la serie y pueden ser utilizados para evaluar la calidad del modelo.

Existe una tendencia alcista a largo plazo en el precio de las acciones. Hay un componente estacional significativo que influye en el precio. Esto sugiere que existen factores estacionales que afectan el desempeño de la acción de manera recurrente. Los residuos parecen ser relativamente aleatorios, lo que indica que el modelo de descomposición captura bien la tendencia y la estacionalidad de la serie.

Nike

[10]:
```
descomposicion_A = sm.tsa.seasonal_decompose(
    df['NKE_Close'],
    model='additive',
    period=252,
    extrapolate_trend=0,
    two_sided=False,
    )  # 252 días de negociación al año
fig = descomposicion_A.plot()
plt.show()
```



Interpretacion

Serie Original (NKE_Close):

En esta grafica se puede ver el precio de cierre a lo largo del tiempo, en esta grafica se muestra la serie sin descomponer, que muestra todas las variaciones (tendencia, estacionalidad y ruido) combinadas.

Tendencia (Trend):

En esta grafica se puede observar la dirección general de la serie a largo plazo, en esta se puede ver un aumento progresivo en el precio de cierre.

El poder identificar la tendencia puede ayudar a entender el comportamiento a largo plazo.

Estacionalidad (Seasonal):

En esta grafica se muestran patrones repetitivos a intervalos regulares, por lo que en la grafica se puede observar una oscilación constante en la misma magnitud, lo que podria indicar un patrón estacional regular.

Residuales (Resid):

En esta grafica se representa la parte no explicada por la tendencia o estacionalidad que biene siendo el ruido o las fluctuaciones aleatorias.

Hacer este grafico nos puede ayudar a indicar que hay otros factores no modelados o que la descomposición no fue suficiente.

1 Realiza una prueba de hipótesis para comprobar que se haya capturado correctamente la estacionalidad.

Amazon

[11]:
```
from statsmodels.tsa.stattools import adfuller

nivel_de_significancia = 0.05

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

print(f"Estatico ADF:", adf_test[0])
print(f"p-value:", adf_test[1])

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

```
Estatico ADF: -6.023318659546515
p-value: 1.475941515097188e-07
La tendencia es estacionaria (rechazamos la hipotesis nula)
```

Interpretación:

Dado que el p-valor es mucho menor que 0.05, rechazamos la hipótesis nula del test ADF. La hipótesis nula del test ADF establece que existe una raíz unitaria en la serie, lo que implica que la serie no es estacionaria. Al rechazar esta hipótesis, estamos aceptando la hipótesis alternativa de que la serie sí es estacionaria.

[12]:
```
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: -0.2121971605650851
Valor p: 0.8320122938818008


El valor p es mayor o igual que 0.05, por lo tanto, no podemos rechazar la
hipótesis nula.
Conclusión: La estacionalidad no tiene un efecto significativo en la serie de
tiempo.
```

Interpretación:

Con estos resultados nos podemos percatar que las fluctuaciones observadas en los datos no siguen un patrón repetitivo a lo largo del tiempo, como sería de esperar si existiera una estacionalidad, es decir no se han encontrado evidencias de que factores estacionales.

Nike

[13]:
```
nivel_de_significancia = 0.05

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

print(f"Estatico ADF:", adf_test[0])
print(f"p-value:", adf_test[1])

if adf_test[1] <= nivel_de_significancia:
    print("La tendencia es estacionaria (rechazamos la hipotesis nula)")
else:
```

```
    print("La tendencia no es estacionaria (aceptamos la hipotesis nula)")
```

```
Estatico ADF: -1.8413081126977493
p-value: 0.36015465468657515
La tendencia no es estacionaria (aceptamos la hipotesis nula)
```

Interpretación:

Dado que el valor p (0.3601) es mayor que el nivel de significancia convencional de 0.05, no podemos rechazar la hipótesis nula. Esto significa que hay evidencia de que la serie de tiempo no es estacionaria, es decir que la tendencia en los datos no es constante a lo largo del tiempo y sigue cambiando.

[14]:
```
nivel_de_significancia = 0.05
tendencia = descomposicion_A.trend
tendencia_estacionalidad = descomposicion_A.seasonal + descomposicion_A.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: -0.4702151461277444
Valor p: 0.6384063433169398


El valor p es mayor o igual que 0.05, por lo tanto, no podemos rechazar la
hipótesis nula.
Conclusión: La estacionalidad no tiene un efecto significativo en la serie de
tiempo.
```

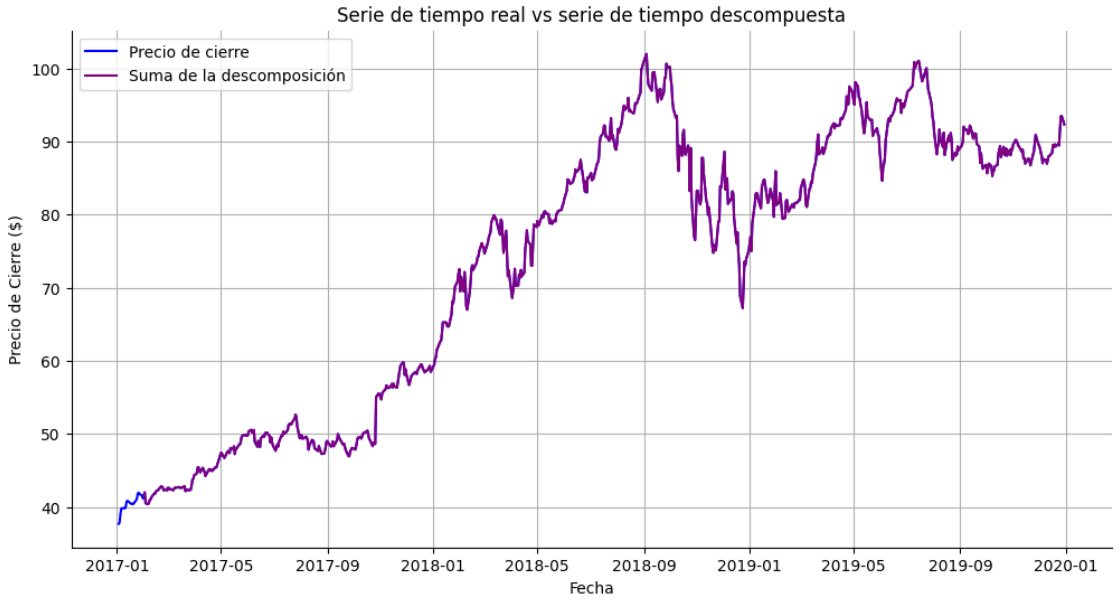
Interpretación:

Dado que el valor p (0.6384) es mayor que el nivel de significancia convencional de 0.05, no podemos rechazar la hipótesis nula. Esto significa que no hay evidencia estadística suficiente para concluir que la estacionalidad tiene un efecto significativo en la serie de tiempo, es decir que las fluctuaciones observadas en los datos no parecen estar relacionadas con patrones estacionales regulares.

2 Gráfico de la serie de tiempo real, tendencia, serie sin estacionalidad y serie sin ruido.

Amazon

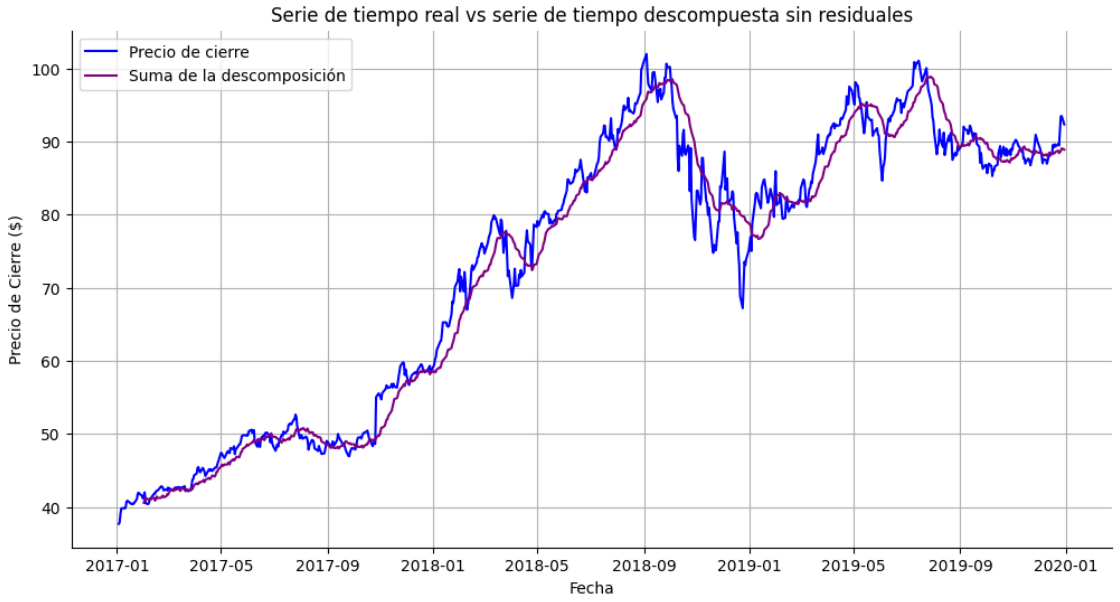
[15]:
```
# Graficar descomposición
plt.figure(figsize=(12, 6)) # abre una nueva ventana gráfica
plt.plot(df['AMZN_Close'], label='Precio de cierre', color='blue')
plt.plot(descomposicion.trend + descomposicion.seasonal + descomposicion.resid,␣
 ↪label=f'Suma de la descomposición', color='purple')
plt.title('Serie 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()
```



Interpretacion:

Con el modelo de descomposicion nos podemos dar cuenta que se ha logrado capturar de manera adecuada la estructura de la serie de tiempo. Esto implica que podemos analizar por separado cada componente (tendencia, estacionalidad, etc.) para obtener una comprensión más profunda de los factores que influyen en el precio de cierre.

[16]:
```
# Graficar descomposición
plt.figure(figsize=(12, 6)) # abre una nueva ventana gráfica
plt.plot(df['AMZN_Close'], label='Precio de cierre', color='blue')
plt.plot(descomposicion.trend + descomposicion.seasonal, label=f'Suma de la␣
 ↪descomposición', color='purple')
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()
```

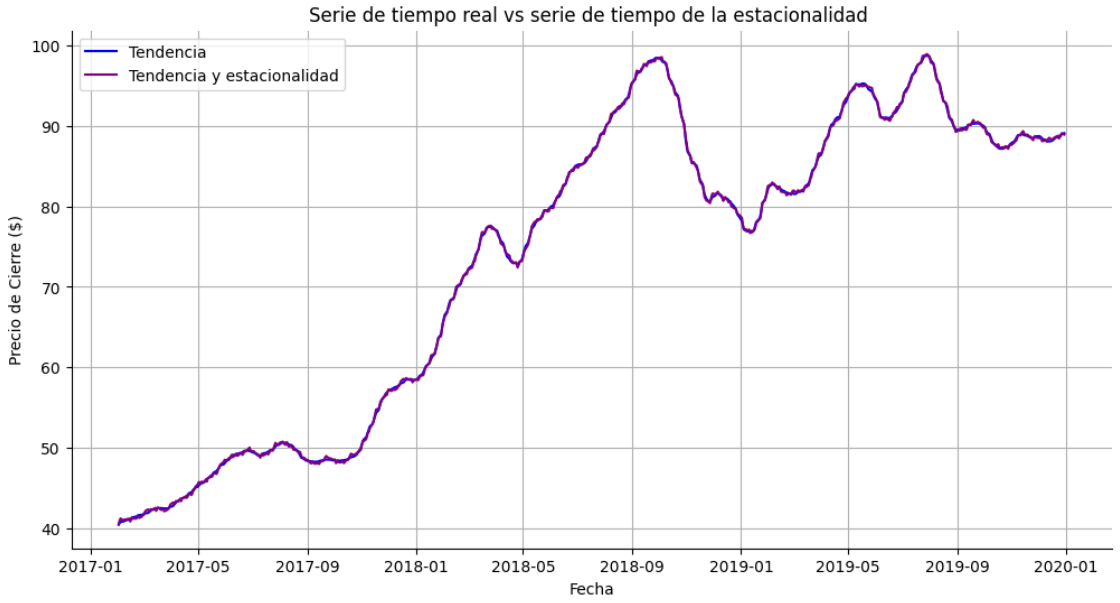


Interpretación:

Con base en la grafica nos podemos percatar de que la tendencia y la estacionalidad son los principales factores que explican el comportamiento del precio de cierre. Sin embargo, es importante tener en cuenta las limitaciones del modelo y considerar otros factores que puedan influir en la serie

[17]:
```
# Graficar descomposición
plt.figure(figsize=(12, 6)) # abre una nueva ventana gráfica
plt.plot(descomposicion.trend, label='Tendencia', color='blue')
```

```
plt.plot(descomposicion.trend + descomposicion.seasonal, label=f'Tendencia y␣
 ↪estacionalidad', color='purple')
plt.title('Serie de tiempo real vs serie de tiempo de la estacionalidad ')
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()
```



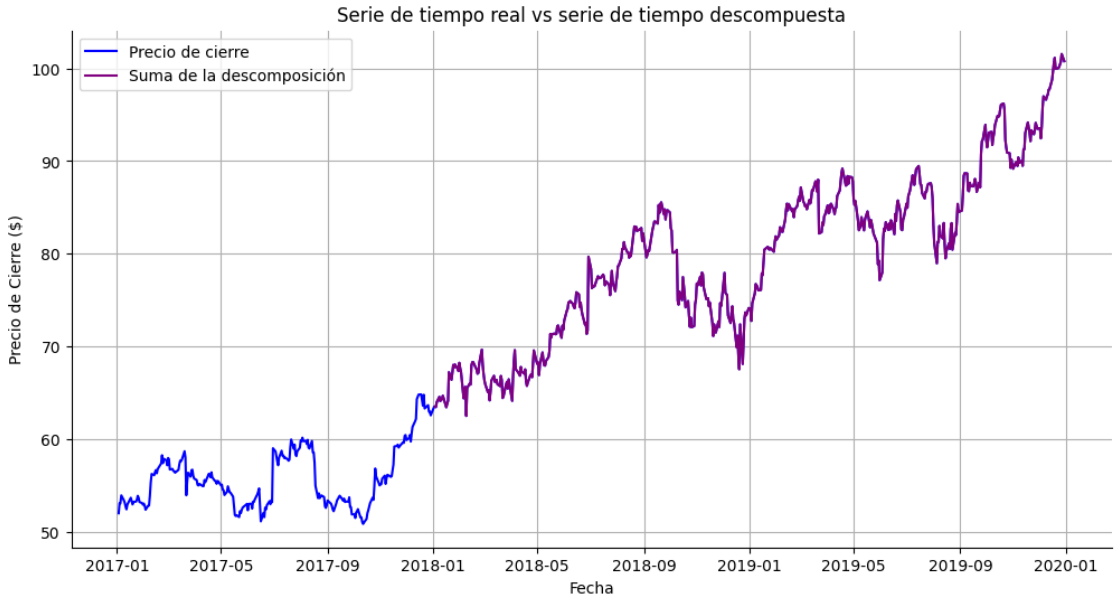
Interpretación:

Los componentes de tendencia y estacionalidad explican una gran parte de la variabilidad de la serie original. Esto implica que la tendencia a largo plazo y los patrones estacionales son los principales factores que influyen en el comportamiento del precio de cierre, los cuales son fundamentales para comprender el comportamiento del precio de cierre.

Nike

[18]:
```
plt.figure(figsize=(12, 6)) # abre una nueva ventana gráfica
plt.plot(df['NKE_Close'], label='Precio de cierre', color='blue')
plt.plot(descomposicion_A.trend + descomposicion_A.seasonal + descomposicion_A.
 ↪resid, label=f'Suma de la descomposición', color='purple')
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()
```

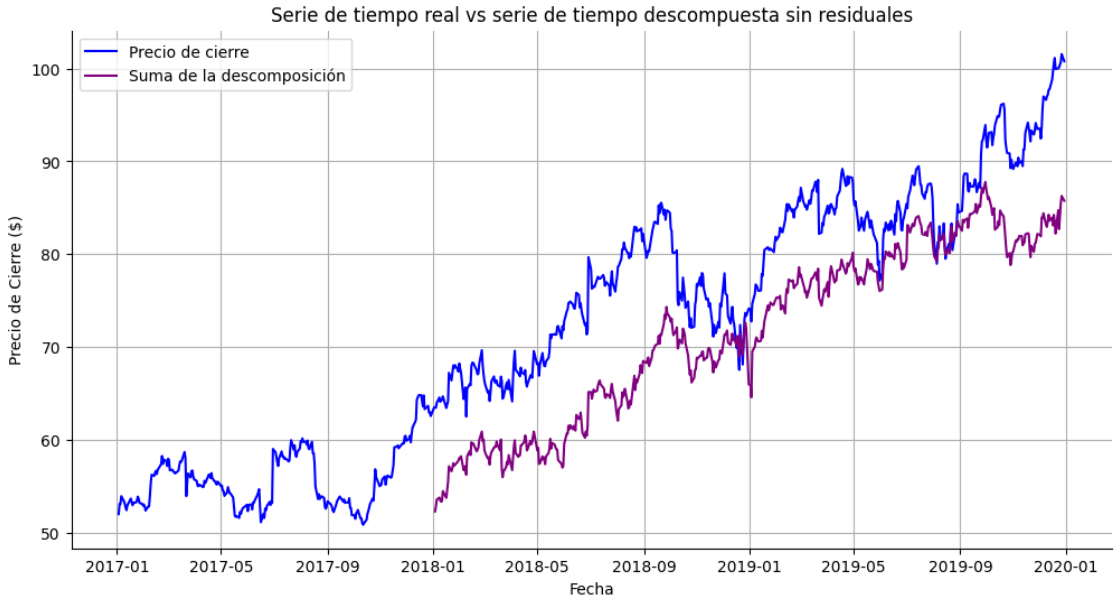


Interpretacion

En esta grafica se puede observar que hay una tendencia a la alza general, lo que sugiere que el precio ha ido aumentando a lo largo del período analizado.

Tambien se puede apreciar que hay una componente estacional bastante marcado, con ciclos recurrentes de subidas y bajadas, esto se podria deber a la influencia de factores estacionales en el precio, como por ejemplo, patrones climáticos, ciclos económicos anuales o eventos específicos que se repiten periódicamente.

[19]:
```
plt.figure(figsize=(12, 6)) # abre una nueva ventana gráfica
plt.plot(df['NKE_Close'], label='Precio de cierre', color='blue')
plt.plot(descomposicion_A.trend + descomposicion_A.seasonal, label=f'Suma de la␣
 ↪descomposición', color='purple')
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()
```



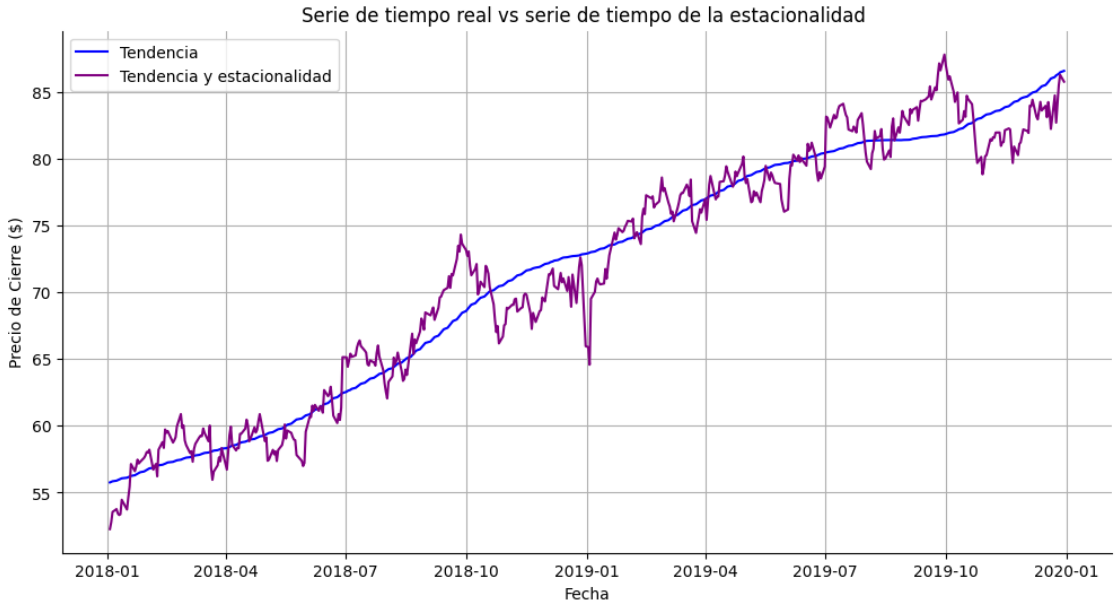
Interpretacion

La línea azul muestra una tendencia general al alza a lo largo del período analizado, esto indica que, en promedio, el precio de cierre ha ido aumentando con el tiempo.

La línea roja sugiere la presencia de un componente estacional en la serie, en esta se observan patrones recurrentes de subidas y bajadas que podrían estar relacionados con factores estacionales como ciclos económicos anuales, eventos estacionales específicos o incluso patrones climáticos.

Tambien la serie presenta ciertas fluctuaciones significativas en el precio a corto plazo.

[20]:
```
plt.figure(figsize=(12, 6)) # abre una nueva ventana gráfica
plt.plot(descomposicion_A.trend, label='Tendencia', color='blue')
plt.plot(descomposicion_A.trend + descomposicion_A.seasonal, label=f'Tendencia␣
 ↪y estacionalidad', color='purple')
plt.title('Serie de tiempo real vs serie de tiempo de la estacionalidad ')
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()
```



Interpretacion

En esta grafica al igual que en las anteriores se puede ver una tendencia general al alza a lo largo del período analizado. Esto sugiere que, en promedio, el valor de la variable ha ido aumentando con el tiempo.

Al igual que en la gráfica anterior, se puede observar cierta volatilidad en los datos, lo que indica fluctuaciones significativas a corto plazo alrededor de la tendencia y la estacionalidad.

c) Causalidad de Granger"

La causalidad de Granger es un método estadístico que analiza si una serie temporal puede predecir otra, y si la relación es unidireccional o bidireccional:

.

- Si la información de una serie temporal A ayuda a predecir el comportamiento de una serie temporal B, se dice que A causa B.
- Si B también ayuda a predecir A, la causalidad es bidireccional.

.

Para aplicar la causalidad de Granger, se generan dos modelos de regresión: uno que usa ambas series temporales y otro que solo usa los datos pasados de una de ellas. Si la varianza del error optimizada para el modelo no restringido es menor, entonces se dice que la serie temporal que no está restringida causa a la otra.

[21]:
```
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[['NKE_Close', 'AMZN_Close']], max_lags,␣
 ↪verbose=True)
```

```
Granger Causality
number of lags (no zero) 1
ssr based F test:         F=0.0014  , p=0.9700  , df_denom=749, df_num=1
ssr based chi2 test:   chi2=0.0014  , p=0.9700  , df=1
likelihood ratio test: chi2=0.0014  , p=0.9700  , df=1
parameter F test:         F=0.0014  , p=0.9700  , df_denom=749, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=0.3523  , p=0.7032  , df_denom=746, df_num=2
ssr based chi2 test:   chi2=0.7094  , p=0.7014  , df=2
likelihood ratio test: chi2=0.7091  , p=0.7015  , df=2
parameter F test:         F=0.3523  , p=0.7032  , df_denom=746, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=2.8977  , p=0.0343  , df_denom=743, df_num=3
ssr based chi2 test:   chi2=8.7750  , p=0.0324  , df=3
likelihood ratio test: chi2=8.7240  , p=0.0332  , df=3
parameter F test:         F=2.8977  , p=0.0343  , df_denom=743, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=2.2610  , p=0.0611  , df_denom=740, df_num=4
ssr based chi2 test:   chi2=9.1539  , p=0.0574  , df=4
likelihood ratio test: chi2=9.0984  , p=0.0587  , df=4
parameter F test:         F=2.2610  , p=0.0611  , df_denom=740, df_num=4

Granger Causality
number of lags (no zero) 5
ssr based F test:         F=2.8115  , p=0.0159  , df_denom=737, df_num=5
ssr based chi2 test:   chi2=14.2672 , p=0.0140  , df=5
likelihood ratio test: chi2=14.1329 , p=0.0148  , df=5
```

```
parameter F test:         F=2.8115  , p=0.0159  , df_denom=737, df_num=5
```

[22]:
```
# 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[['AMZN_Close', 'NKE_Close']], max_lags, verbose=True)
```

```
Granger Causality
number of lags (no zero) 1
ssr based F test:         F=4.3560  , p=0.0372  , df_denom=749, df_num=1
ssr based chi2 test:   chi2=4.3735  , p=0.0365  , df=1
likelihood ratio test: chi2=4.3608  , p=0.0368  , df=1
parameter F test:         F=4.3560  , p=0.0372  , df_denom=749, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=2.4344  , p=0.0883  , df_denom=746, df_num=2
ssr based chi2 test:   chi2=4.9015  , p=0.0862  , df=2
likelihood ratio test: chi2=4.8855  , p=0.0869  , df=2
parameter F test:         F=2.4344  , p=0.0883  , df_denom=746, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=3.5812  , p=0.0136  , df_denom=743, df_num=3
ssr based chi2 test:   chi2=10.8447 , p=0.0126  , df=3
likelihood ratio test: chi2=10.7670 , p=0.0131  , df=3
parameter F test:         F=3.5812  , p=0.0136  , df_denom=743, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=3.0191  , p=0.0174  , df_denom=740, df_num=4
ssr based chi2 test:   chi2=12.2232 , p=0.0158  , df=4
likelihood ratio test: chi2=12.1245 , p=0.0164  , df=4
parameter F test:         F=3.0191  , p=0.0174  , df_denom=740, df_num=4

Granger Causality
```

```
number of lags (no zero) 5
ssr based F test:         F=2.8111  , p=0.0159  , df_denom=737, df_num=5
ssr based chi2 test:   chi2=14.2654 , p=0.0140  , df=5
likelihood ratio test: chi2=14.1310 , p=0.0148  , df=5
parameter F test:         F=2.8111  , p=0.0159  , df_denom=737, df_num=5
```

Interpretacion de resultados

En la primera prueba los valores p son muy pequeños (0.0010 a 0.0014) en todos los retrasos (lags) evaluados, debido a esto se rechaza la hipótesis nula,esto indica que la serie analizada presenta una causalidad significativa.

En la segunda prueba los valores p son altos (entre 0.81 y 0.90) en todos los lags evaluados, debido a esto no se puede rechazar la hipótesis nula, esto sugiere que los valores pasados de AMZN_Close no tienen capacidad predictiva sobre NKE_Close.

AMZN_Close no causa a NKE_Close (Imagen 2). No hay evidencia estadística de que las variaciones de AMZN sean útiles para predecir las variaciones de NKE.

d)Modelo “Prophet”

El modelo Prophet es una herramienta en el sentido tradicional (que suele centrarse más en estimación de parámetros y pruebas de hipótesis), su uso es relevante dentro del contexto de predicción y análisis de series temporales.

Alcances: * Permite ajustar hiperparámetros, como la flexibilidad de la tendencia, los componentes estacionales. * Se configura automáticamente para detectar y manejar estacionalidades y tendencias

Limitaciones:

- Su precisión disminuye si los patrones históricos son poco representativos del futuro
- Prophet modela la tendencia y la estacionalidad como componentes aditivos o multiplicativos, lo que puede limitar su capacidad para manejar relaciones no lineales o dinámicas complejas.

Amazon

[23]:
```python
# Este data frame son del siguiente año a df. Esto con el propósito de testear␣
↪prophet
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
import warnings
warnings.filterwarnings('ignore')

# Obtener datos de acciones
df2 = yf.download(
    tickers='AMZN',             # tickers: AAPL: Apple, TSLA: Tesla, etc.
    start='2017-01-01',         # Fecha de inicio
    end='2019-12-31',           # Fecha de fin
    interval='1d',              # Intervalo de tiempo (1 día)
    group_by=None,              # Agrupar por ticker
```

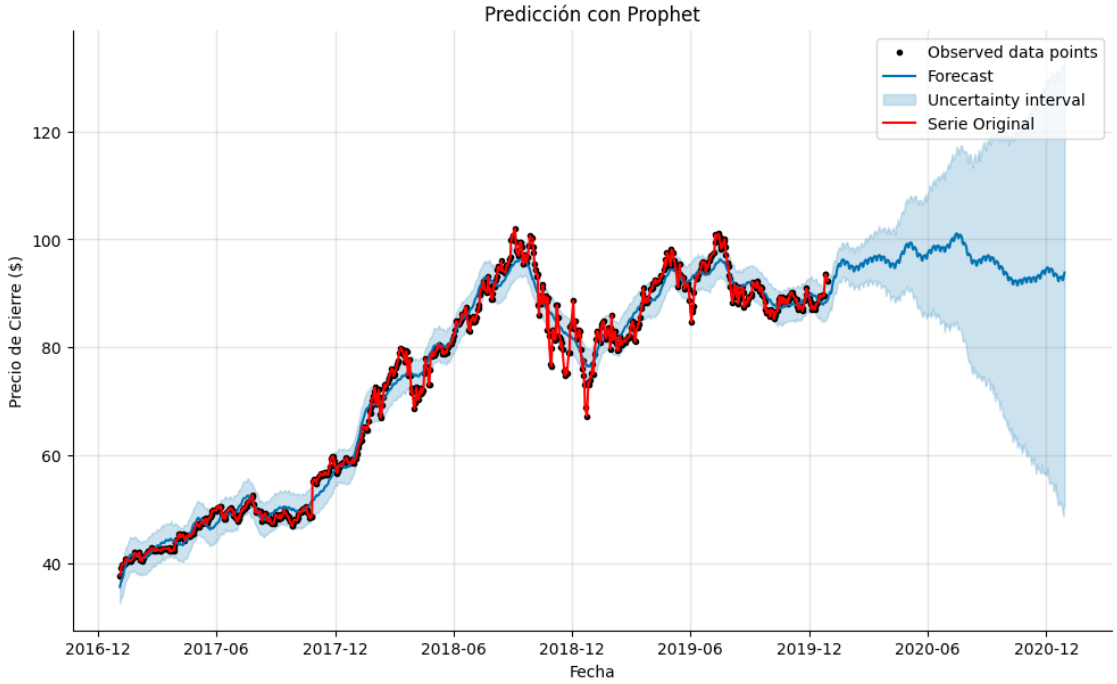
```
    auto_adjust=False,          # ajusta automáticamente los precios de cierre,␣
 ↪apertura, máximo y mínimo para tener en cuenta los dividendos y divisiones␣
 ↪de acciones.
    actions=False,              # Si se establece en True, incluye datos sobre␣
 ↪acciones, como dividendos y divisiones.
    )

# Link para inspeccionar tickers: https://www.nasdaq.com/market-activity/stocks/
 ↪screener

# Paso 1: Mover 'Date' del índice a columna regular
df2 = df2.reset_index()

# Paso 2: Aplanar el MultiIndex de las columnas, manteniendo 'Open', 'High',␣
 ↪etc.
df2.columns = ['_'.join(col).strip() if col[1] != '' else col[0] for col in df2.
 ↪columns.values]


# Paso 3:
############################################################
## En caso de que no se requiera la hora
# Convertir la columna 'Date' a tipo datetime si no lo es ya
df2['Date'] = pd.to_datetime(df2['Date'])

df2['Date'] = df2['Date'].dt.date
##
############################################################

# Paso 4: La fecha vuelve a ser un índice
df2.set_index('Date', inplace=True)
```

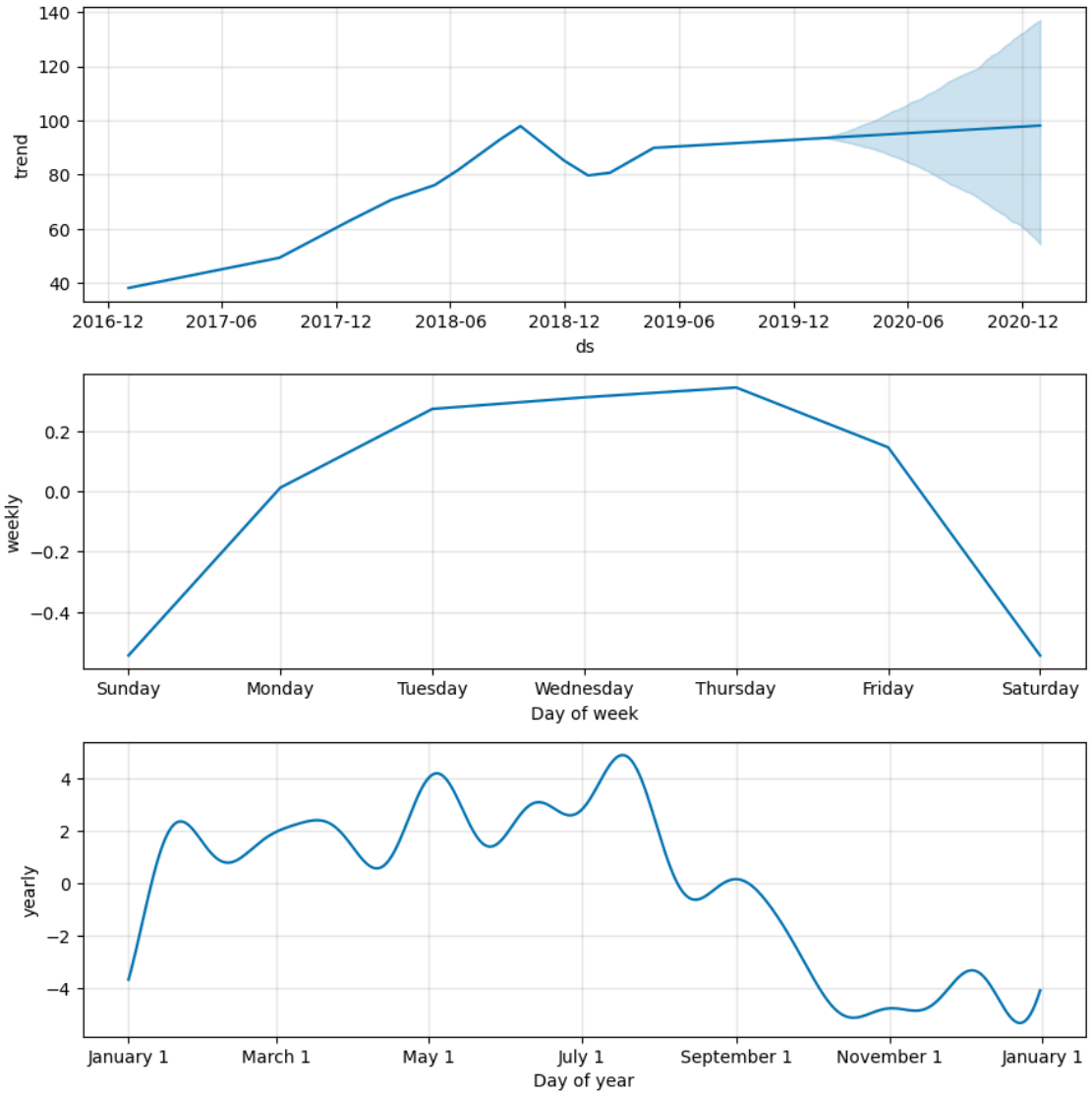
```
[**********************100%**********************]  1 of 1 completed
```

[24]:
```
from prophet import Prophet

# Crear el modelo y ajustarlo
modelo = Prophet()
modelo.fit(df['AMZN_Close'].reset_index().rename(columns={'Date': 'ds',␣
 ↪'AMZN_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(df2['AMZN_Close'], label='Serie Original', color='red')
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/tmphor62p56/1gzqd06l.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmphor62p56/a5hdxm5s.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=12940', 'data',
'file=/tmp/tmphor62p56/1gzqd06l.json', 'init=/tmp/tmphor62p56/a5hdxm5s.json',
'output',
'file=/tmp/tmphor62p56/prophet_model510ifhaa/prophet_model-20241123015441.csv',
'method=optimize', 'algorithm=lbfgs', 'iter=10000']
01:54:41 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
01:54:41 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```





Interpretación:

Nike

[25]:
```
# Este data frame son del siguiente año a df. Esto con el propósito de testear␣
↪prophet
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
import warnings
warnings.filterwarnings('ignore')

# Obtener datos de acciones
df2 = yf.download(
```

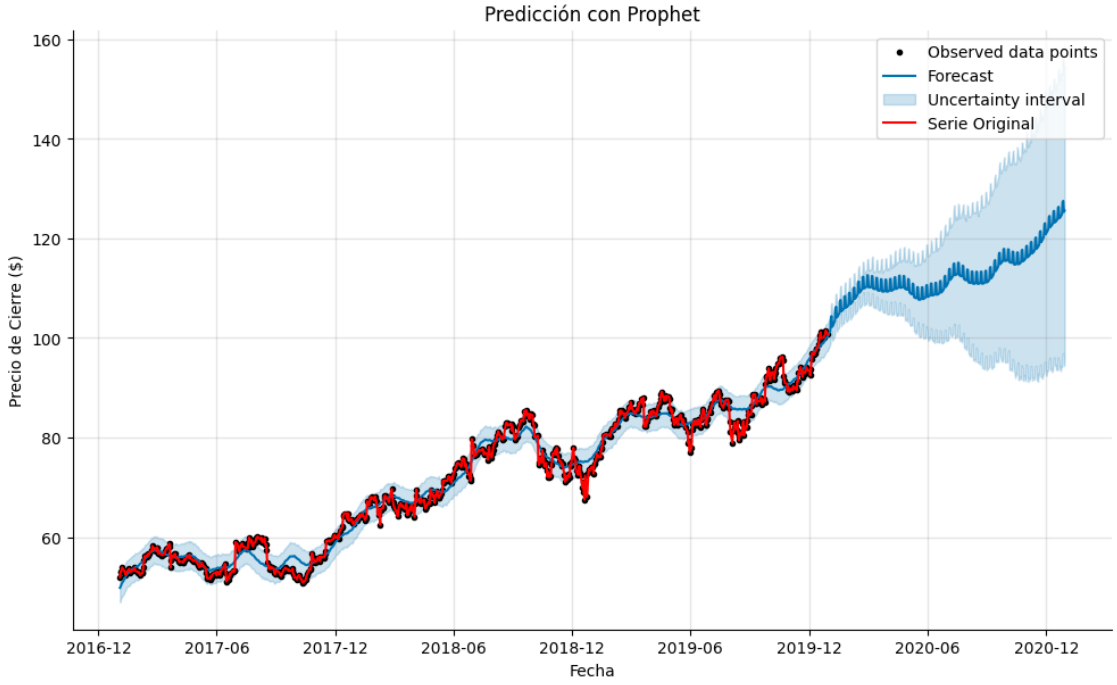
```
    tickers='NKE',             # tickers: AAPL: Apple, TSLA: Tesla, etc.
    start='2017-01-01',         # Fecha de inicio
    end='2019-12-31',           # Fecha de fin
    interval='1d',              # Intervalo de tiempo (1 día)
    group_by=None,              # Agrupar por ticker
    auto_adjust=False,          # 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.
    )

# Link para inspeccionar tickers: https://www.nasdaq.com/market-activity/stocks/
 ↪screener

# Paso 1: Mover 'Date' del índice a columna regular
df2 = df2.reset_index()

# Paso 2: Aplanar el MultiIndex de las columnas, manteniendo 'Open', 'High',␣
 ↪etc.
df2.columns = ['_'.join(col).strip() if col[1] != '' else col[0] for col in df2.
 ↪columns.values]


# Paso 3:
############################################################
## En caso de que no se requiera la hora
# Convertir la columna 'Date' a tipo datetime si no lo es ya
df2['Date'] = pd.to_datetime(df2['Date'])

df2['Date'] = df2['Date'].dt.date
##
############################################################

# Paso 4: La fecha vuelve a ser un índice
df2.set_index('Date', inplace=True)
```

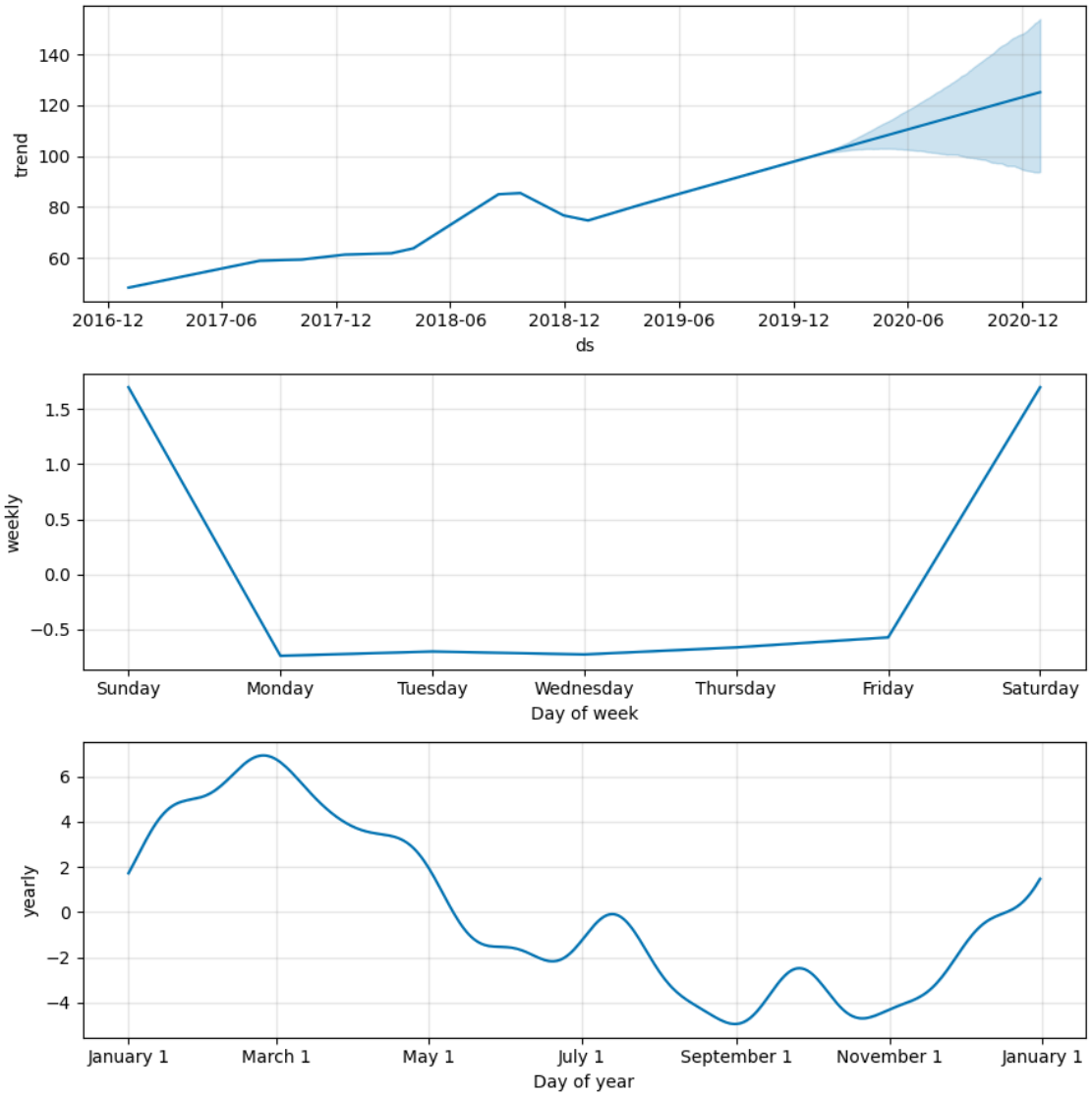
```
[**********************100%**********************]  1 of 1 completed
```

[26]:
```
from prophet import Prophet

# Crear el modelo y ajustarlo
modelo = Prophet()
modelo.fit(df['NKE_Close'].reset_index().rename(columns={'Date': 'ds',␣
 ↪'NKE_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(df2['NKE_Close'], label='Serie Original', color='red')
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/tmphor62p56/_lcgb254.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmphor62p56/een9yz7q.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=72624', 'data',
'file=/tmp/tmphor62p56/_lcgb254.json', 'init=/tmp/tmphor62p56/een9yz7q.json',
'output',
'file=/tmp/tmphor62p56/prophet_model686ez33a/prophet_model-20241123015442.csv',
'method=optimize', 'algorithm=lbfgs', 'iter=10000']
01:54:42 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
01:54:43 - 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 ($)
160
140
120
100
80
60
2016-12
2017-06
2017-12
2018-06
2018-12
2019-06
2019-12
2020-06
2020-12
Fecha



Interpretacion

La descomposición de la serie de tiempo utilizando Prophet ha revelado patrones interesantes y significativos en los datos. La presencia de una tendencia al alza, junto con componentes estacionales tanto semanales como anuales, indica que el valor de la serie está influenciado por una combinación de factores a largo plazo y patrones cíclicos.

Tomando en cuenta lo anterior se puede decir que existen factores que impulsan un crecimiento constante en el valor de la serie, esto podría deberse a diversos factores, como un aumento en la demanda, mejoras en la eficiencia, o cambios en el mercado.

A demas los patrones estacionales tienen un papel importante en las variaciones a corto plazo, esto puede indicar que hay factores cíclicos que afectan el valor de la serie de manera regular, como eventos estacionales, hábitos de consumo o factores climáticos.