tp-final

April 24, 2025

#

TRABAJO FINAL - ANÁLISIS DE SERIES TEMPORALES - MIA

Nota:

La funcionalidad de visualización de jupyter notebooks en github es solamente un preview.

Para mejor visualización se sugiere utilizar el visualizador recomndado por la comunidad: nbviewer

Puedes a acceder al sigiente enlace para ver este notebook en dicha página: Trabajo final

---

Instalaciones:

Este notebook utiliza Poetry para la gestión de dependencias. Para instalar las dependencias necesarias, instala Poetry y ejecuta el siguiente comando en la raíz del proyecto:

```
poetry install
# Activa el entorno virtual de Poetry
eval $(poetry env activate)
```

Importaciones:

[1]:
```
# Sistema y manejo de archivos
import os
import datetime

# Manejo de datos
import pandas as pd
import numpy as np

# Visualización
import matplotlib.pyplot as plt
import mplfinance as mpf
from tqdm.notebook import tqdm
from wand.image import Image as WImage

# Modelado y análisis de series temporales
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.seasonal import seasonal_decompose
import statsmodels.tsa.stattools as sts
```

```
import statsmodels.graphics.tsaplots as sgt
from pmdarima import auto_arima
from prophet import Prophet

# Machine Learning
from sklearn.metrics import mean_absolute_error, mean_absolute_percentage_error
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import TimeSeriesSplit
from xgboost import XGBRegressor

# Deep Learning
# NOTA: Backend keras = TensorFlow
from keras.models import Sequential
from keras.layers import LSTM, Dense, Input

# API y configuración
from dotenv import load_dotenv
from binance.client import Client

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

```
Importing plotly failed. Interactive plots will not work.
```

Configuraciones:

[2]:
```
load_dotenv()

BINANCE_API_KEY = os.getenv("BINANCE_API_KEY")
BINANCE_API_SECRET = os.getenv("BINANCE_API_SECRET")


BASE_FOLDER_PATH = os.getcwd() # Trabajo Final
DATASET_FOLDER_PATH = os.path.join(BASE_FOLDER_PATH, "datasets") # Trabajo␣
 ↪Final/dataset
DATASET_CSV_FILE_PATH = os.path.join(DATASET_FOLDER_PATH, "BTCUSDT_1D.csv") #␣
 ↪Trabajo Final/dataset/BTCUSDT_1D.csv
DATASET_PICKLE_FILE_PATH = os.path.join(DATASET_FOLDER_PATH, "BTCUSDT_1D.pkl")␣
 ↪# Trabajo Final/dataset/BTCUSDT_1D.pkl

LABEL_FONTSIZE = 10
TITLE_FONTSIZE = 16
```

Figura 1: Imagen generada con Microsoft Image Creator

Datos del proyecto:

Subtitulo	Trabajo final - Análisis de series temporales 1 - FIUBA

Descrpción Bitcoin ↔ USDT
Integrantes Bruno Masoller (brunomaso1@gmail.com)- Juan Cruz Ferreyra (ferreyra.juancruz95@gmail.com)- Francisco Rassi (franciscorassi@gmail.com)

Tabla de contenido

0.1 Consigna

[3]:
```
WImage(filename='TP_final_AST_01MA2025.pdf')
```

[3]:

Universidad de Buenos Aires

Laboratorio de Sistemas Embebidos

Especialización en Inteligencia Artificial

Análisis de Series de Tiempo

Docente: Camilo Argoty

Nombre: _____ Código: _____

Fecha: _____

Trabajo final

El grupo debe trabajar con alguna fuente de datos la cual debe buscar por su cuenta. una o más series de tiempo para analizar. Cada estudiante debe presentar:

1. (50 % de la calificación) Código en Python comentado con el procesamiento de los datos. El código debe ser reproducible (ejecutable) e incluir las siguientes secciones:
   a) Limpieza y preparación de los datos.
   b) Creación de modelos de análisis de series de tiempo. Estudiar al menos 3 tipos posibles de modelos seleccionados por el estudiante.
   c) Generación de pronósticos por cada modelo, evaluación y comparación.
2. (50 % de la calificación) Informe de análisis explicando los procesos realizados y las conclusiones obtenidas de los análisis. El informe debe incluir:
   a) Planteamiento de pregunta de investigación: Plantee una pregunta que se pueda responder con su trabajo con los datos.
   b) Descripción de los datos: Describa el origen y el tipo de dato de cada uno de los atributos de la tabla.
   c) Descripción de los modelos: Indique las características de los modelos construidos, incluyendo las gráficas de las series de tiempo originales y de los datos simulados usando los modelos.
   d) Pruebas sobre los modelos: Describa los resultados de las pruebas, evaluaciones y validaciones realizadas sobre los modelos analizados.
   e) Conclusiones: Saque conclusiones finales, tanto sobre el fenómeno estudiado como de los modelos utilizados. El objetivo es responder la pregunta de investigación. También incluya lecciones aprendidas, situaciones difíciles con los datos y cómo se superaron y otros comentarios que se consideren relevantes.

El plazo para la entrega es el próximo jueves 24 de abril a las las 7:00 pm, horario de la clase de Análisis de Series de Tiempo. Es importante que los documentos y archivos entregados contengan los códigos y nombres de todos los miembros del grupo.

0.2 Resolución

0.2.1 Contexto

Binance es una plataforma de intercambio de criptomonedas que permite a los usuarios comprar, vender e intercambiar una variedad de criptomonedas. Fundada en 2017, Binance se ha convertido en uno de los intercambios más grandes y populares del mundo, ofreciendo una amplia gama de servicios relacionados con criptomonedas, incluyendo comercio al contado, comercio de futuros, préstamos y ahorros en criptomonedas, y más.

Como parte de su oferta, Binance proporciona una API (Interfaz de Programación de Aplicaciones) que permite a los desarrolladores acceder a datos de mercado, realizar operaciones y gestionar cuentas programáticamente. Esta API es ampliamente utilizada por traders algorítmicos, desarrolladores de aplicaciones y otros interesados en el ecosistema de criptomonedas.

El objetivo general de este trabajo es analizar el comportamiento de los precios de las criptomonedas a través de la API de Binance. Se utilizará datos históricos de precios para realizar un análisis exploratorio y aplicar técnicas de análisis de series temporales. A lo largo del trabajo, abordaremos los siguientes puntos: - Descripción de la API de Binance y su funcionalidad. - Planteo de objetivos específicos desafiantes. - Obtención de datos históricos de precios de criptomonedas a través de la API. - Análisis exploratorio de los datos obtenidos. - Aplicación de técnicas de análisis de series temporales para modelar y predecir el comportamiento de los precios. - Discusión de los resultados y conclusiones. - Recomendaciones para futuros trabajos y mejoras en el análisis.

Conocimientos básicos

API Binance

La API pública de Binance ofrece un conjunto completo de datos de mercado históricos para todos los símbolos soportados, organizados en archivos diarios y mensuales, cubriendo tanto Spot como los dos tipos de Futuros (USD -margined y COIN-margined). Estos datos incluyen operaciones agregadas (`aggTrades`), operaciones individuales (`trades`) y velas OHLCV (`klines`).

Se puede acceder a la API de Binance a través de su documentación oficial, que proporciona información detallada sobre cómo utilizarla y qué datos están disponibles: https://developers.binance.com/docs/binance-spot-api-docs/rest-api/general-api-information

Sin embargo, para comunicarse con la API de Binance, es necesario tener una cuenta en Binance y generar una clave API (key y secret).

Por otro lado, Binance ya compila los datos históricos de precios en archivos CSV, lo que facilita el acceso a los datos sin necesidad de utilizar la API. Estos archivos CSV están disponibles en el siguiente enlace: https://data.binance.vision/

Este recompliado de datos incluye la misma información que la API, pero no es necesario tener una API key para acceder a ellos. Los archivos CSV están organizados por símbolo y tipo de mercado (Spot, USD -margined y COIN-margined), lo que facilita la búsqueda y descarga de los datos deseados. Se puede obtener más información en el repositorio oficial de Binance en GitHub: https://github.com/binance/binance-public-data?tab=readme-ov-file

Por ejemplo, si se quiere obtener los datos por día (agregados mensualmente) de BTCUSDT en el mercado Spot, se puede acceder a la siguiente URL: https://data.binance.vision/?prefix=data/spot/monthly/klines/BTCUSDT/1d/. Esto pro-

porcionará una lista de archivos CSV que contienen los datos de precios diarios de BTCUSDT en el mercado Spot, organizados por mes.

> Nota: En el repositorio se puede obtener el mismo recompilado que se descarga con la API en la carpeta *datasets*.

Simbolos

Los símbolos son pares de criptomonedas que se pueden intercambiar entre sí. Por ejemplo, el símbolo `BTCUSDT` representa el intercambio de Bitcoin (BTC) por Tether (USDT). Binance soporta una amplia variedad de símbolos, lo que permite a los usuarios intercambiar diferentes criptomonedas entre sí.

> Nota: Se puede consultar la lista completa de símbolos disponibles en Binance a través de su API o en su sitio web. La API proporciona un endpoint específico para obtener información sobre los símbolos y sus características, como el precio mínimo, el precio máximo, el tamaño mínimo de la orden, entre otros. La lista de símbolos se puede obtener a través de la siguiente URL: https://api.binance.com/api/v3/exchangeInfo. Esta API devuelve un objeto JSON que contiene información sobre todos los símbolos disponibles en Binance, incluyendo su estado, límites de precios y tamaños de órdenes. Esto no requiere una API key.

> Tip: Utilizando el cliente, se pueden filtrar los símbolos activos:

```
client = Client(api_key, api_secret)
exchange_info = client.get_exchange_info()
for s in exchange_info['symbols']:
    print(s['symbol'])
```

Mercados disponibles

SPOT

El mercado Spot es el mercado donde se compran y venden activos financieros para entrega inmediata. En el contexto de criptomonedas, esto significa que los usuarios pueden comprar o vender criptomonedas al precio actual del mercado y recibir la criptomoneda de inmediato. En Binance, el mercado Spot permite a los usuarios intercambiar una amplia variedad de criptomonedas entre sí.

FUTURES

Binance Futures son contratos derivados que especulan sobre el precio de un activo en una fecha futura. Hay dos tipos principales: - USD -margined Futures: denominados y liquidados en stablecoins (USDT o USDC), con ganancias y pérdidas calculadas en esas monedas - COIN-margined Futures: denominados y liquidados en la criptomoneda subyacente, ideales para HODLers y uso de la propia criptomoneda como margen

Tipos de datos

Dentro de cada mercado (Spot, USD -margined y COIN-margined), Binance expone tres flujos/datasets principales:

- *aggTrades*: datos de operaciones agregadas por precio y lado (taker/maker) en intervalos muy cortos (100 ms), que reducen redundancias y facilitan el análisis de volúmenes

- *trades*: operaciones individuales recientes —hasta 1.000 por solicitud en Spot y hasta 500 en Futures— con detalles de cada fill: precio, cantidad, timestamp, y flags como isBuyerMaker
- *klines*: Velas OHLCV (Open, High, Low, Close, Volume) en intervalos configurables (1s, 1m, 3m, 5m, 15m, 30m, 1h, 2h, 4h, 6h, 8h, 12h, 1d, 3d, 1w, 1mo)

0.2.2 Objetivo específicos

Como objetivos específicos, que comprenden la consigna del trabajo, se plantean los siguientes desafíos:

1. Uno de los principales desafíos en en mundo critpo es obtener predicciones fiables y precisas sobre el comportamiento de los precios de las criptomonedas. BTCUSDT es el par más negociado en Binance, por lo que un análisis interesante sería responder la pregunta de como se ha comportado el dolar frente al Bitcoin y si es posible predecir su comportamiento a futuro.
2. Un segundo desafío interesante es analizar los modelos de series de tiempo trabajados durante el cursado para evaluar su potencial predictivo en una serie extremadamente volátil como lo es el precio de las criptomonedas.

0.2.3 Carga de datos

> Advertencia: Si estás ejecutando este cuaderno, asegúrate de cargar las variables de entorno necesarias para acceder a la API de Binance, esto implica tener un archivo `.env` con las variables `BINANCE_API_KEY` y `BINANCE_API_SECRET` definidas.
>
> Nota: Si no tienes una API key, no es necesario para este cuaderno. Una vez descargado el dataset, se lo guarda en un formato que se puede recuperar con todas sus características. Simplemente puedes saltear las siguientes celdas hasta *Análisis de datos* en la ejecución.

[4]:
```
# Creamos el cliente de Binance
binance_client = Client(api_key=BINANCE_API_KEY, api_secret=BINANCE_API_SECRET)

# Cargamos los datos de la API de Binance
symbol = "BTCUSDT"
# format (year, month, day, hour, minute, second)
start_time = datetime.datetime(2024, 3, 1, 0, 0, 0)
end_time = datetime.datetime(2025, 3, 31, 0, 0, 0)
klines = binance_client.get_historical_klines(
    symbol=symbol,
    interval=Client.KLINE_INTERVAL_1DAY,  # 1 day interval
    start_str=str(start_time),
    end_str=str(end_time),
)

# Convertimos los datos a un DataFrame de pandas
df = pd.DataFrame(
    klines,
    columns=[
```

```
        "Open Time",
        "Open",
        "High",
        "Low",
        "Close",
        "Volume",
        "Close Time",
        "Quote Asset Volume",
        "Number of Trades",
        "Taker Buy Base Asset Volume",
        "Taker Buy Quote Asset Volume",
        "Ignore",
    ],
)

print(f"Cantidad de filas: {len(df)}")
df.head()  # Mostramos las primeras filas del DataFrame
```

```
Cantidad de filas: 396
```

[4]:

	Open Time	Open	High	Low	Close	Volume	Close Time	Quote Asset Volume	Number of Trades	Taker Buy Base Asset Volume	Taker Buy Quote Asset Volume	Ignore
0	1709251200000	61130.99000000	63114.23000000	60777.00000000	62387.90000000	47737.93473000	1709337599999	2956537377.96602980	1947444	24195.70252000	1498771144.09995600	0
1	1709337600000	62387.90000000	62433.19000000	61561.12000000	61987.28000000	25534.73659000	1709423999999	1582566973.83409310	1641808	12691.37721000	786583052.08627260	0
2	1709424000000	61987.28000000	63231.88000000	61320.00000000	63113.97000000	28994.90903000	1709510399999	1804536272.94434580	1992011	14905.18600000	927868983.73084310	0
3	1709510400000	63113.97000000	68499.00000000	62300.00000000	68245.71000000	84835.16005000	1709596799999	5568877537.08125230	3887853	45319.08640000	2974396037.68572030	0
4	1709596800000	68245.71000000	69000.00000000	59005.00000000	63724.01000000	132696.78130000	1709683199999	8674526591.28714170	5310706	65991.84526000	4318205559.64365010	0

[5]:
```
# Guardamos el DataFrame en un archivo csv
df.to_csv(DATASET_CSV_FILE_PATH, index=False)
```

0.2.4 Análisis de datos

[6]:
```
# Cargamos el conjunto de datos csv
df = pd.read_csv(DATASET_CSV_FILE_PATH)
```

Tipos de datos De la documentación de la API de Binance, se sabe que las columnas del conjunto de datos son las siguientes: - `Open time` → Timestamp de apertura de la vela (en milisegundos) - `Open` → Precio de apertura de la vela - `High` → Precio máximo de la vela - `Low` → Precio mínimo de la vela - `Close` → Precio de cierre de la vela - `Volume` → Volumen de operaciones durante la vela - `Close time` → Timestamp de cierre de la vela (en milisegundos) - `Quote asset volume` → Volumen de operaciones en la moneda base (en USDT) - `Number of trades` → Número de operaciones realizadas durante la vela - `Taker buy base asset volume` → Volumen de operaciones en la moneda base (en USDT) compradas por el lado del comprador - `Taker buy quote asset volume` → Volumen de operaciones en la moneda base (en USDT) compradas por el lado del vendedor - `Ignore` → Si ignora el campo, se puede dejar vacío

Se aplica la siguiente conversión de tipos de datos:

[7]:
```
# Definimos los tipos de datos para las columnas
columns_to_convert = {
    "Open Time": "datetime64[ms]",
    "Close Time": "datetime64[ms]",
    "Open": float,
    "High": float,
    "Low": float,
    "Close": float,
    "Volume": float,
    "Quote Asset Volume": float,
    "Taker Buy Base Asset Volume": float,
    "Taker Buy Quote Asset Volume": float,
    "Number of Trades": int,
}

# Apply type conversions
for column, dtype in columns_to_convert.items():
    df[column] = df[column].astype(dtype)

# Establecer como índice la columna "Open Time"
df.set_index("Open Time", inplace=True)

df.head()
```

[7]:
```
                Open      High       Low     Close        Volume  \
Open Time
2024-03-01  61130.99  63114.23  60777.00  62387.90   47737.93473
```

```
2024-03-02  62387.90  62433.19  61561.12  61987.28   25534.73659
2024-03-03  61987.28  63231.88  61320.00  63113.97   28994.90903
2024-03-04  63113.97  68499.00  62300.00  68245.71   84835.16005
2024-03-05  68245.71  69000.00  59005.00  63724.01  132696.78130

                          Close Time  Quote Asset Volume  Number of Trades  \
Open Time
2024-03-01 2024-03-01 23:59:59.999        2.956537e+09           1947444
2024-03-02 2024-03-02 23:59:59.999        1.582567e+09           1641808
2024-03-03 2024-03-03 23:59:59.999        1.804536e+09           1992011
2024-03-04 2024-03-04 23:59:59.999        5.568878e+09           3887853
2024-03-05 2024-03-05 23:59:59.999        8.674527e+09           5310706

            Taker Buy Base Asset Volume  Taker Buy Quote Asset Volume  Ignore
Open Time
2024-03-01                  24195.70252                 1.498771e+09       0
2024-03-02                  12691.37721                 7.865831e+08       0
2024-03-03                  14905.18600                 9.278690e+08       0
2024-03-04                  45319.08640                 2.974396e+09       0
2024-03-05                  65991.84526                 4.318206e+09       0
```

Información básica Utilizamos la función `info()` para obtener información básica sobre el conjunto de datos, como el número de filas y columnas, los tipos de datos y la cantidad de valores no nulos en cada columna. Esto nos ayuda a comprender la estructura del conjunto de datos y a identificar cualquier problema potencial con los datos.

[8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 396 entries, 2024-03-01 to 2025-03-31
Data columns (total 11 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Open                          396 non-null    float64
 1   High                          396 non-null    float64
 2   Low                           396 non-null    float64
 3   Close                         396 non-null    float64
 4   Volume                        396 non-null    float64
 5   Close Time                    396 non-null    datetime64[ms]
 6   Quote Asset Volume            396 non-null    float64
 7   Number of Trades              396 non-null    int32
 8   Taker Buy Base Asset Volume   396 non-null    float64
 9   Taker Buy Quote Asset Volume  396 non-null    float64
 10  Ignore                        396 non-null    int64
dtypes: datetime64[ms](1), float64(8), int32(1), int64(1)
memory usage: 35.6 KB
```

Podemos observar que no hay elementos nulos. También se puede utilizar la función

`isnull().sum()` para verificar si hay valores nulos en el conjunto de datos y `df.isna().sum()` para verificar si hay valores `NaN`.

[9]:
```
cant_nulos = df.isnull().sum()
print(f"Cantidad de datos nulos:\n{cant_nulos}")
```

```
Cantidad de datos nulos:
Open                            0
High                            0
Low                             0
Close                           0
Volume                          0
Close Time                      0
Quote Asset Volume              0
Number of Trades                0
Taker Buy Base Asset Volume     0
Taker Buy Quote Asset Volume    0
Ignore                          0
dtype: int64
```

[10]:
```
nan_values = df.isna().sum()
print(f"Cantidad de datos NaN:\n{nan_values}")
```

```
Cantidad de datos NaN:
Open                            0
High                            0
Low                             0
Close                           0
Volume                          0
Close Time                      0
Quote Asset Volume              0
Number of Trades                0
Taker Buy Base Asset Volume     0
Taker Buy Quote Asset Volume    0
Ignore                          0
dtype: int64
```

Finalmente, mostramos las estadísticas descriptivas del conjunto de datos utilizando la función `describe()`. Esto nos proporciona información sobre la distribución de los datos, como la media, la desviación estándar, los valores mínimo y máximo, y los percentiles. Esto es útil para comprender mejor el comportamiento de los precios y el volumen de operaciones en el conjunto de datos.

[11]:
```
df.describe()
```

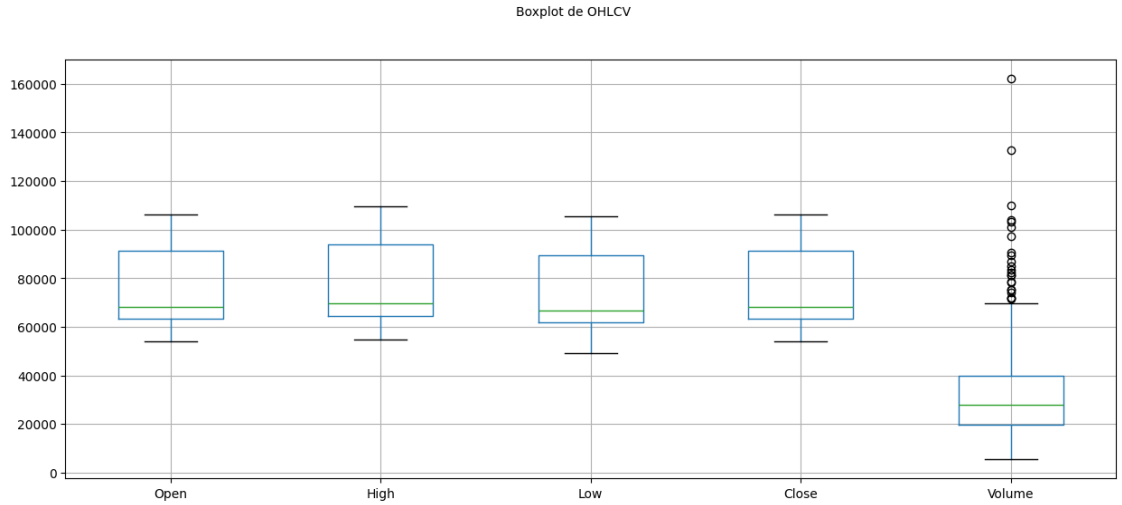
[11]:

	Open	High	Low	Close	\
count	396.000000	396.000000	396.000000	396.000000	
mean	75130.871667	76660.888384	73510.109293	75184.959520	
min	53962.970000	54850.000000	49000.000000	53962.970000	
25%	63161.887500	64464.245000	61804.452500	63192.085000	

```
50%     68247.795000   69500.000000   66837.835000   68256.935000
75%     91162.280000   94033.247500   89286.742500   91162.272500
max    106143.820000  109588.000000  105321.490000  106143.820000
std     15343.870506   15612.707655   15029.769726   15332.134304

              Volume                  Close Time  Quote Asset Volume  \
count     396.000000                         396        3.960000e+02
mean    32429.972411  2024-09-15 11:59:59.999000        2.428852e+09
min      5420.221140  2024-03-01 23:59:59.999000        4.562354e+08
25%     19830.229000  2024-06-08 17:59:59.999000        1.444602e+09
50%     27921.300355  2024-09-15 11:59:59.999000        2.016132e+09
75%     39982.410077  2024-12-23 05:59:59.999000        2.967016e+09
max    162065.591860  2025-03-31 23:59:59.999000        1.104800e+10
std     20288.615124                         NaN        1.599045e+09

       Number of Trades  Taker Buy Base Asset Volume  \
count      3.960000e+02                   396.000000
mean       3.322309e+06                 15938.662369
min        5.628320e+05                  2284.191780
25%        1.562842e+06                  9642.221350
50%        2.546511e+06                 13793.797980
75%        4.265081e+06                 19865.147165
max        1.201236e+07                 77363.263230
std        2.386543e+06                 10101.220495

       Taker Buy Quote Asset Volume  Ignore
count                  3.960000e+02   396.0
mean                   1.193337e+09     0.0
min                    1.922910e+08     0.0
25%                    6.903443e+08     0.0
50%                    9.736778e+08     0.0
75%                    1.447411e+09     0.0
max                    5.657171e+09     0.0
std                    7.982296e+08     0.0
```

Mirando rápidamente podemos ver que tenemos valores coherentes, sin embargo, también podemos hacer un chequeo de outliers:

[12]:
```python
# Chequear outliers
plt.figure(figsize=(15, 6))
plt.suptitle("Boxplot de OHLCV", fontsize=LABEL_FONTSIZE)
df.boxplot(column=["Open", "High", "Low", "Close", "Volume"], figsize=(15, 5))
plt.show()
```



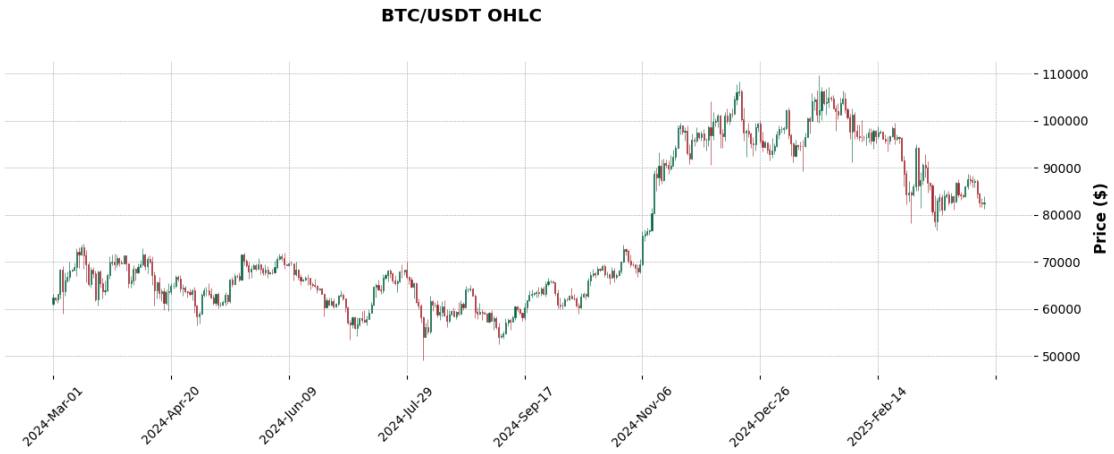
Nota: Se puede aplicar un filtro para eliminar los outliers. Esto se puede hacer utilizando el método `quantile()` para calcular los percentiles 0.01 y 0.99 y luego filtrando los datos para mantener solo aquellos que están dentro de este rango. Sin embargo, no queremos "gaps" en la serie (ya que habría que ver una política de imputación de datos), por lo que se trabajará con esta información.

Finalmente, guardamos el dataset en un archivo pickle que nos permite incorporar información extra como los tipos de las columnas para su posterior uso:

[13]:
```
df.to_pickle(DATASET_PICKLE_FILE_PATH)
```

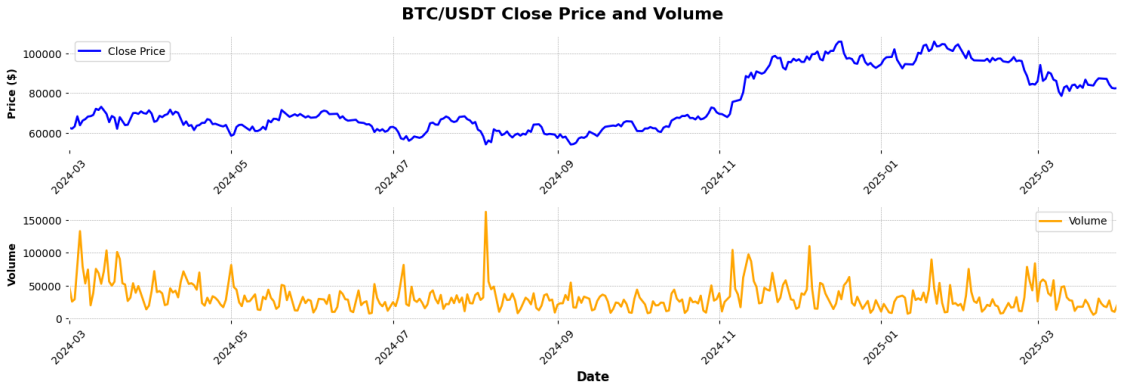
Visualización rápida Se utiliza la librería `mplfinance` (también se puede utilizar la librería `datavizcatalogue`) para graficar y obtener una rápida visión de los datos. La función `plot_candlestick` permite visualizar las velas OHLC (Open, High, Low, Close) de los precios a lo largo del tiempo:

[14]:
```
df_ohlc = df[[ "Open", "High", "Low", "Close"]].copy()

# Definimos el estilo del gráfico
style = mpf.make_mpf_style(base_mpf_style="charles", rc={"font.size": 10})

# Creamos el gráfico de velas
mpf.plot(df_ohlc, type="candle", style=style, title="BTC/USDT OHLC",␣
 ↪ylabel="Price ($)", figsize=(16, 5))
```

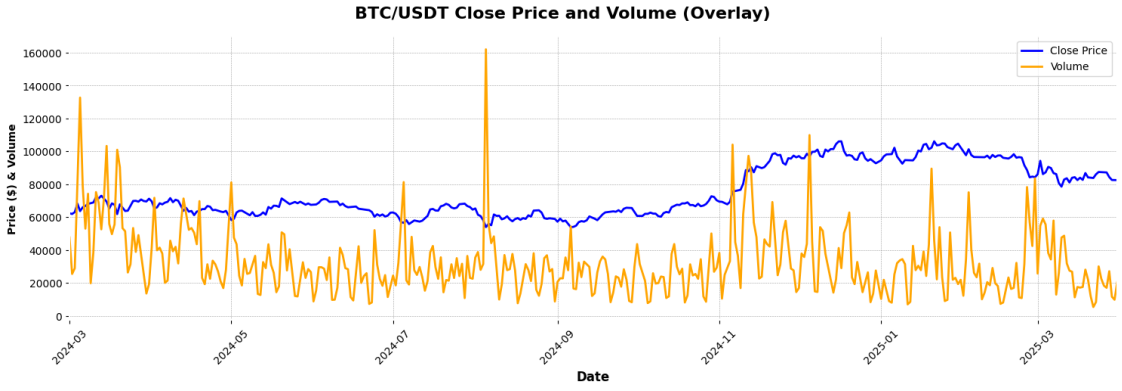


Gráficos sobre las variables Las variables de interés para resolver los objetivos específicos son el precio de cierre. Podemos realizar gráficos simples sobre esta variable para observar su comportamiento. También se muestra la relación que tiene con el volumen de operaciones.

[15]:
```
# Graficamos ambos precios y volumen en un solo gráfico
plt.figure(figsize=(15, 5))
plt.suptitle("BTC/USDT Close Price and Volume", fontsize=TITLE_FONTSIZE)

plt.subplot(2, 1, 1)
plt.plot(df["Close"], label="Close Price", color="blue")
plt.ylabel("Price ($)", fontsize=LABEL_FONTSIZE)
plt.autoscale(axis="x", tight=True)
plt.xticks(rotation=45)
plt.legend(fontsize=LABEL_FONTSIZE)

plt.subplot(2, 1, 2)
plt.plot(df["Volume"], label="Volume", color="orange")
plt.ylabel("Volume", fontsize=LABEL_FONTSIZE)
plt.autoscale(axis="x", tight=True)
plt.xticks(rotation=45)
plt.legend(fontsize=LABEL_FONTSIZE)
plt.tight_layout()

plt.xlabel("Date")

plt.show()
```



[16]:
```
# Graficamos uno sobre el otro
plt.figure(figsize=(15, 5))
plt.suptitle("BTC/USDT Close Price and Volume (Overlay)",␣
 ↪fontsize=TITLE_FONTSIZE)

plt.plot(df["Close"], label="Close Price", color="blue")
plt.plot(df["Volume"], label="Volume", color="orange")
plt.ylabel("Price ($) & Volume", fontsize=LABEL_FONTSIZE)
plt.autoscale(axis="x", tight=True)
plt.xticks(rotation=45)
plt.legend(fontsize=LABEL_FONTSIZE)
plt.tight_layout()
plt.xlabel("Date")

plt.show()
```

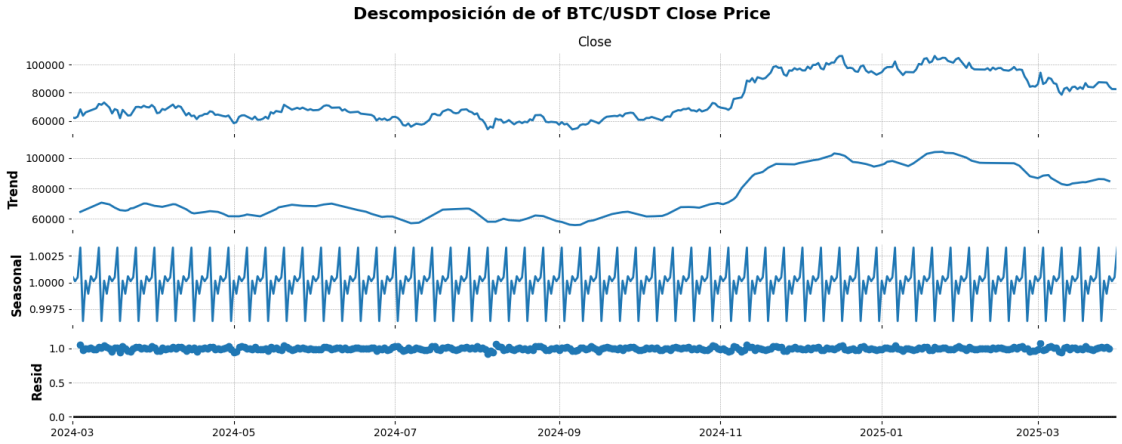


En los gráficos anteriores podemos observar que el volumen se mantiene casi constante (con un amplio ruido) a lo largo del tiempo, mientras que el precio de cierre presenta una tendencia creciente en el período estudiado. Esto sugiere que, a pesar de la estabilidad en el volumen de operaciones, el

precio de BTCUSDT ha estado aumentando. En un principio, no parece haber una relación directa entre el volumen de operaciones y el precio de cierre, ya que los picos de volumen no siempre coinciden con picos en el precio en el período analizado. Sin embargo, se puede observar que desde el 11-2024 el volumen se ha vuelto más volátil y el precio a aumentado.

Descomposición ETS La descomposición de la serie temporal es una técnica que permite la separar sus componentes principales: tendencia, estacionalidad y residuales. Podemos analizar esta descomposición para entender mejor el comportamiento del precio de cierre de la serie temporal y cómo se relacionan estas componentes entre sí. La descomposición se puede realizar utilizando la función `seasonal_decompose` de la librería `statsmodels`. Esta función permite descomponer la serie temporal en sus componentes principales y visualizar cada una de ellas por separado. Si nos enfocamos en el precio de cierre, obtenemos los siguientes resultados:

[17]:
```
result = seasonal_decompose(df["Close"], model="multiplicative")
fig = result.plot()
fig.suptitle("Descomposición de of BTC/USDT Close Price",␣
 ↪fontsize=TITLE_FONTSIZE)
fig.set_size_inches(15, 6)
fig.tight_layout()
plt.show()
```



En esta descomposición se puede observar claramente la tendencia alcista del precio de cierre luego del 11-2024. También se pueden observar uniformidad en la estacionalidad y el ruido, lo que sugiere que la serie temporal no presenta patrones estacionales claros.

Pruebas de estacionaridad Los modelos auto regresivos y de media móvil son válidos cuando la serie es estacionaria, pues su fórmula y propiedades de inferencia asumen media y varianza constantes en el tiempo. El test de Dickey-Fuller aumentado (ADF) es una prueba estadística que se utiliza para determinar si una serie temporal es estacionaria o no. La hipótesis nula de la prueba ADF es que la serie temporal tiene una raíz unitaria, lo que indica que no es estacionaria. Si el valor p de la prueba es menor que un nivel de significancia (por ejemplo, 0.05), se rechaza la hipótesis nula y se concluye que la serie temporal es estacionaria. La función `adfuller` de la librería `statsmodels`

se utiliza para realizar la prueba ADF. Esta función devuelve varios valores, incluyendo el estadístico de la prueba y el valor p. Inicialmente, se realiza una prueba de estacionaridad sobre la variable objetivo (precio de cierre):

[18]:
```
result = sts.adfuller(df["Close"])
print(f"Estadístico ADF: {result[0]}")
print(f"Valor p: {result[1]}")
print(f"Valores críticos: {result[4]}")
```

```
Estadístico ADF: -1.420856101366877
Valor p: 0.5722376131152292
Valores críticos: {'1%': -3.447014064067954, '5%': -2.8688850015516016, '10%':
-2.5706826870693797}
```

En este caso podemos observar que el valor de p es mayor que 0.05, lo que indica que no podemos rechazar la hipótesis nula y, por lo tanto, la serie temporal no es estacionaria.

Sin embargo, esto no implica que no podemos realizar un análisis de series temporales. Existen técnicas para transformar la serie temporal en una serie estacionaria, como la diferenciación o la transformación logarítmica. La diferenciación consiste en calcular la diferencia entre los valores de la serie temporal en diferentes momentos en el tiempo. Esto puede ayudar a eliminar tendencias y estacionalidades, haciendo que la serie sea más estacionaria.

Para verificar si esta técnica nos permite obtener una serie estacionaria realizamos nuevamente el test ADF sobre la serie diferenciada.

[19]:
```
result = sts.adfuller(df["Close"].diff().dropna())
print(f"Estadístico ADF: {result[0]}")
print(f"Valor p: {result[1]}")
print(f"Valores críticos: {result[4]}")
```

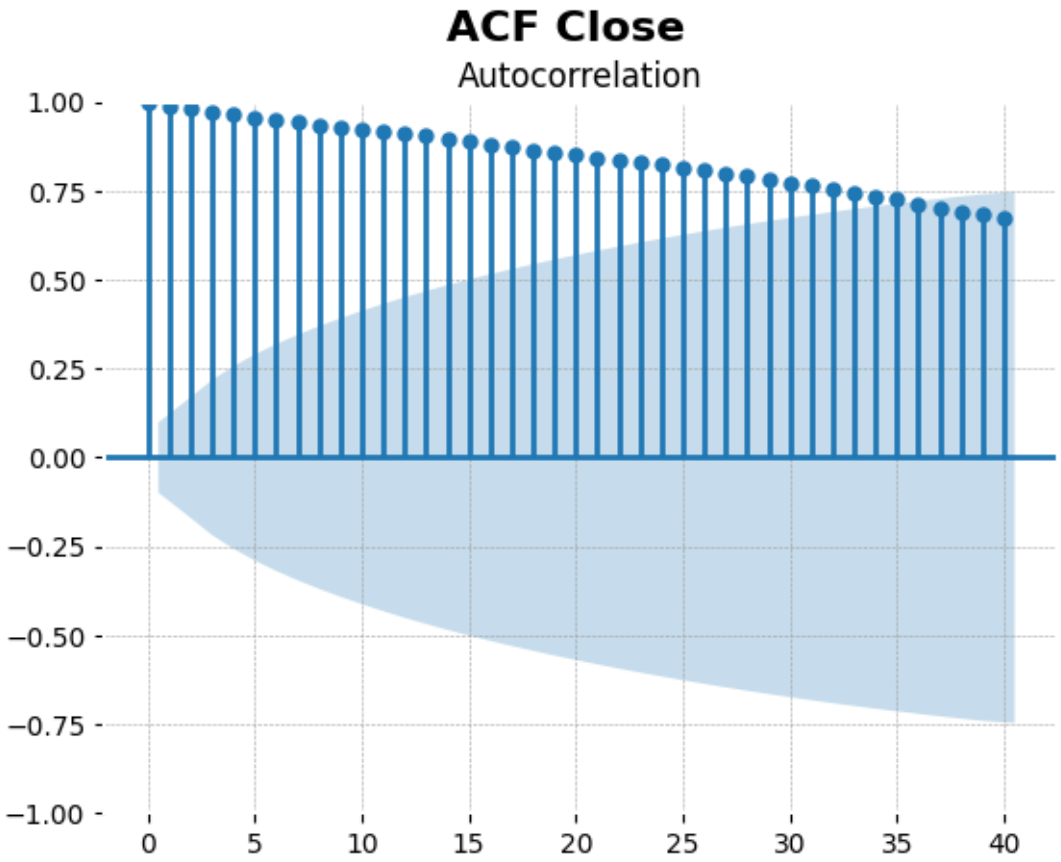
```
Estadístico ADF: -21.412563900673952
Valor p: 0.0
Valores críticos: {'1%': -3.4470566389664703, '5%': -2.8689037160476016, '10%':
-2.570692663557422}
```

Observamos que la serie de tiempo diferenciada es estacionario. Esto es importante en cuanto nos permite aplicar modelos como los autorregresivos o de media móvil que asumen estacionaridad.

Autocorrelation y autocorrelación parcial La función `plot_acf` y `plot_pacf` de la librería `statsmodels` se utilizan para graficar los correlogramas ACF y PACF, respectivamente. Estos gráficos nos permiten observar la correlación entre los valores de la serie temporal en diferentes momentos en el tiempo y determinar el orden de diferenciación necesario para hacer la serie estacionaria.

Autocorrelación:

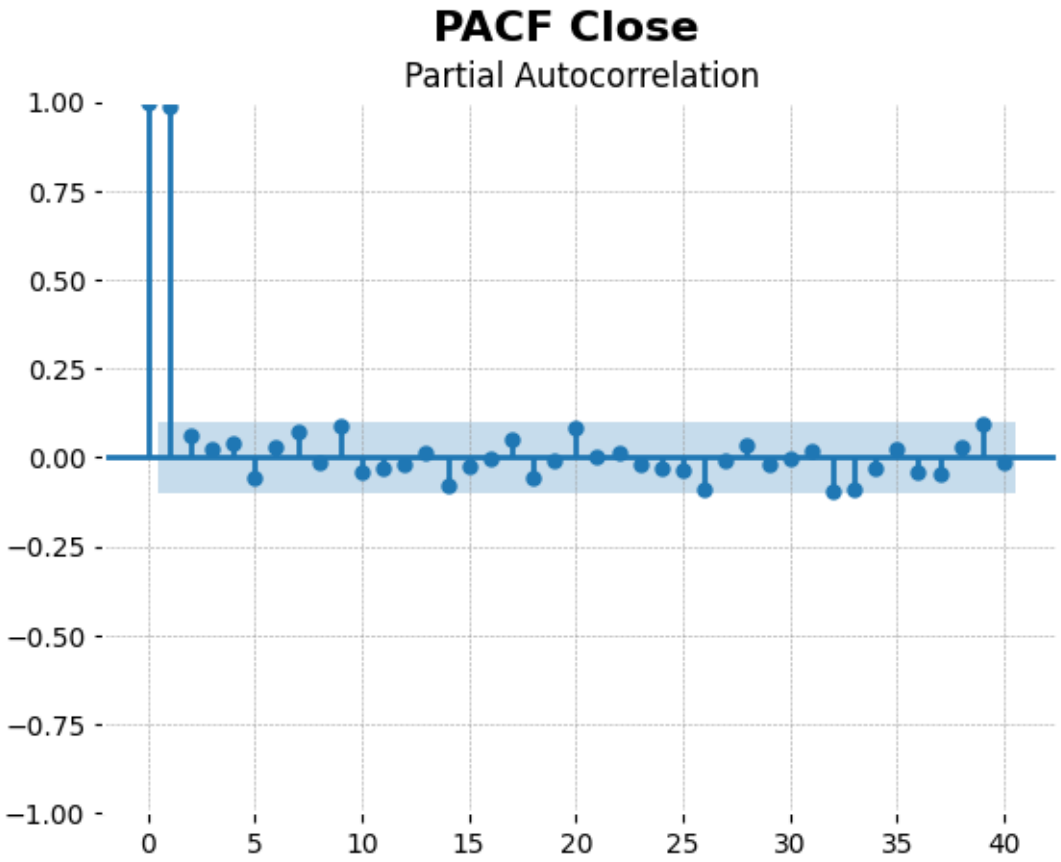
[20]:
```
result = sgt.plot_acf(df["Close"], lags=40)
plt.suptitle("ACF Close", fontsize=TITLE_FONTSIZE)
plt.show()
```



La autocorrelación permite determinar el valor de q en modelos de media movil $MA(q)$. En esta gráfica podemos observar que todos los rezagos son estadísticamente significativos, hasta el 35 aproximadamente.

Autocorrelación parcial:

[21]:
```
sgt.plot_pacf(df["Close"], lags=40)
plt.suptitle("PACF Close", fontsize=TITLE_FONTSIZE)
plt.show()
```

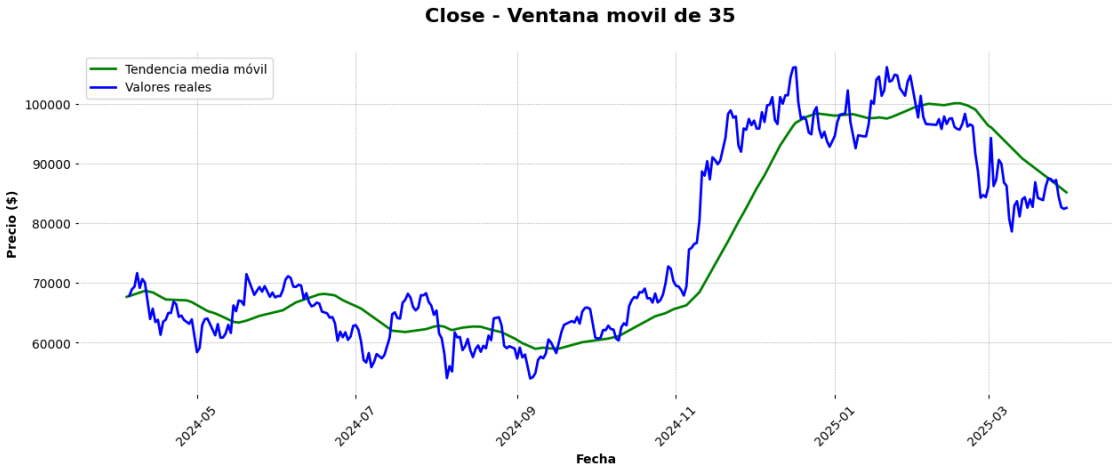


La autocorrelación parcial permite determinar el valor inicial de p que se utiliza en los modelos auto regresivos $AR(p)$. Este valor indica el rezago significante más grande luego del cual la mayoría de los rezagos se vuelven insignificantes. En el gráfico de autocorrelación parcial, podemos observar que unicamente el rezago 1 es estadísticamente significativo.

Podemos suavizar la serie original para identificar tendencias con el valor encontrado (por ejemplo, con una ventana de 35 períodos):

[22]:
```
window = 35
rolling_mean = df["Close"].rolling(window=window).mean()

plt.figure(figsize=(15, 5))
plt.suptitle("Close - Ventana movil de {}".format(window),␣
 ↪fontsize=TITLE_FONTSIZE)

plt.plot(rolling_mean, label="Tendencia media móvil", color="green")
plt.plot(df["Close"][window:], label="Valores reales", color="blue")
plt.xticks(rotation=45)
plt.xlabel("Fecha", fontsize=LABEL_FONTSIZE)
plt.ylabel("Precio ($)", fontsize=LABEL_FONTSIZE)
```

```
plt.legend(loc="best")
plt.grid(True)
plt.show()
```



Se puede observar que hay una tendencia hacia abajo para los próximos valores a corto plazo, sin embargo, dado los resultados de las pruebas y la forma de la gráfica, es muy difícil realizar una estimación a largo plazo. Se prevé que los modelos ARMA no sean efectivos para predecir el comportamiento de la serie temporal sin realizar diferenciaciones de la misma, ya que la serie no es estacionaria. En la siguiente sección se analizan varios tipos de modelos.

0.2.5 Modelos de series temporales

Creación de train y test sets:

[23]:
```
df_models = pd.DataFrame()
df_models["ds"] = df["Close Time"].dt.date.copy()
df_models["y"] = df["Close"].copy()

# Split into initial train set (for rolling forecasting) and test set
split_index = int(len(df_models) * 0.80)

initial_train = df_models.iloc[:split_index].copy()
test_set = df_models.iloc[split_index:].copy()
```

Creación de utilidades para comparar los modelos

[24]:
```
def evaluate_model(y_true, y_pred, model_name="Model"):
    mae = mean_absolute_error(y_true, y_pred)
    mape = mean_absolute_percentage_error(y_true, y_pred)
    rmse = np.sqrt(mean_absolute_error(y_true, y_pred))

    return {"model": model_name, "mae": mae, "mape": mape, "rmse": rmse}
```

[25]:
```
# List to store results
metrics_list = []
```

Baseline (Naive model)

A modo de *baseline* predecimos siempre el precio del día anterior. Esto quiere decir que los modelos de series de tiempo que entrenemos a continuación deberían mostrar una performance mayor a la de esta heurística para probar su potencial de forecasting.

[26]:
```
# Baseline: naive forecasting
naive_test_set = df_models.iloc[split_index - 1:].copy() # add last training␣
 ↪day to forecast first test day

y_pred = naive_test_set["y"][1:]
y_test = naive_test_set["y"].shift(1)[1:]

metrics = evaluate_model(y_test, y_pred, model_name="Naive")
metrics_list.append(metrics)

print(f"MAE: {metrics['mae']:.1f}")
print(f"MAPE: {metrics['mape']:.4f}")
print(f"RMSE: {metrics['rmse']:.1f}")
```

```
MAE: 1735.7
MAPE: 0.0189
RMSE: 41.7
```

ARIMA En base al análsis anterior, concluimos que la serie diferenciada una vez es estacionaria. Vamos entonces a aplicar un modelo $ARIMA$ con parámetros $(p = 1, d = 1, q = 35)$, esto es cantidad de coeficientes AR, numero de diferenciaciones iterativas, y cantidad de coeficientes MA respectivamente.

[27]:
```
# Usar la columna "Close"
data = df["Close"]

# Definir el punto de corte (80% train, 20% test)
train_size = int(len(data) * 0.8)
train, test = data[:train_size], data[train_size:]

# Entrenar ARIMA(35,1,1) en el set de entrenamiento
model = ARIMA(train, order=(1, 1, 35))
model_fit = model.fit()

# Resumen del modelo entrenado
print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Close   No. Observations:                  316
```

Model:	ARIMA(1, 1, 35)	Log Likelihood	-2809.617
Date:	Thu, 24 Apr 2025	AIC	5693.233
Time:	18:28:49	BIC	5832.078
Sample:	03-01-2024 - 01-10-2025	HQIC	5748.707
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.0997	23.587	-0.004	0.997	-46.329	46.129
ma.L1	0.0904	23.593	0.004	0.997	-46.152	46.332
ma.L2	0.0217	0.220	0.099	0.921	-0.410	0.453
ma.L3	0.0282	0.535	0.053	0.958	-1.020	1.077
ma.L4	-0.0209	0.612	-0.034	0.973	-1.220	1.178
ma.L5	0.0592	0.553	0.107	0.915	-1.025	1.143
ma.L6	-0.0429	1.453	-0.029	0.976	-2.890	2.804
ma.L7	0.0449	1.154	0.039	0.969	-2.216	2.306
ma.L8	0.0156	1.185	0.013	0.989	-2.307	2.338
ma.L9	0.0363	0.258	0.141	0.888	-0.470	0.542
ma.L10	0.0695	0.832	0.084	0.933	-1.560	1.700
ma.L11	0.0135	1.558	0.009	0.993	-3.040	3.067
ma.L12	-0.0108	0.168	-0.064	0.949	-0.340	0.318
ma.L13	0.0250	0.272	0.092	0.927	-0.507	0.557
ma.L14	-0.0441	0.621	-0.071	0.943	-1.261	1.173
ma.L15	-0.0899	1.110	-0.081	0.935	-2.265	2.085
ma.L16	0.0496	2.016	0.025	0.980	-3.902	4.002
ma.L17	0.0090	1.370	0.007	0.995	-2.677	2.695
ma.L18	-0.0918	0.085	-1.083	0.279	-0.258	0.074
ma.L19	0.0536	2.174	0.025	0.980	-4.208	4.315
ma.L20	-0.0443	1.485	-0.030	0.976	-2.955	2.867
ma.L21	-0.0555	1.200	-0.046	0.963	-2.408	2.297
ma.L22	0.0147	1.200	0.012	0.990	-2.337	2.366
ma.L23	0.0664	0.476	0.140	0.889	-0.866	0.999
ma.L24	0.0515	1.526	0.034	0.973	-2.939	3.042
ma.L25	0.0295	1.065	0.028	0.978	-2.058	2.117
ma.L26	0.0272	0.590	0.046	0.963	-1.129	1.183
ma.L27	0.0076	0.590	0.013	0.990	-1.150	1.165
ma.L28	-0.0035	0.128	-0.027	0.978	-0.255	0.248
ma.L29	-0.0304	0.106	-0.286	0.775	-0.239	0.178
ma.L30	0.0389	0.716	0.054	0.957	-1.365	1.443
ma.L31	0.0054	0.989	0.005	0.996	-1.933	1.944
ma.L32	-0.0418	0.041	-1.031	0.303	-0.121	0.038
ma.L33	-0.0198	0.987	-0.020	0.984	-1.954	1.914
ma.L34	0.0603	0.372	0.162	0.871	-0.670	0.790
ma.L35	0.0041	1.468	0.003	0.998	-2.873	2.881
sigma2	3.093e+06	2.14e+05	14.457	0.000	2.67e+06	3.51e+06

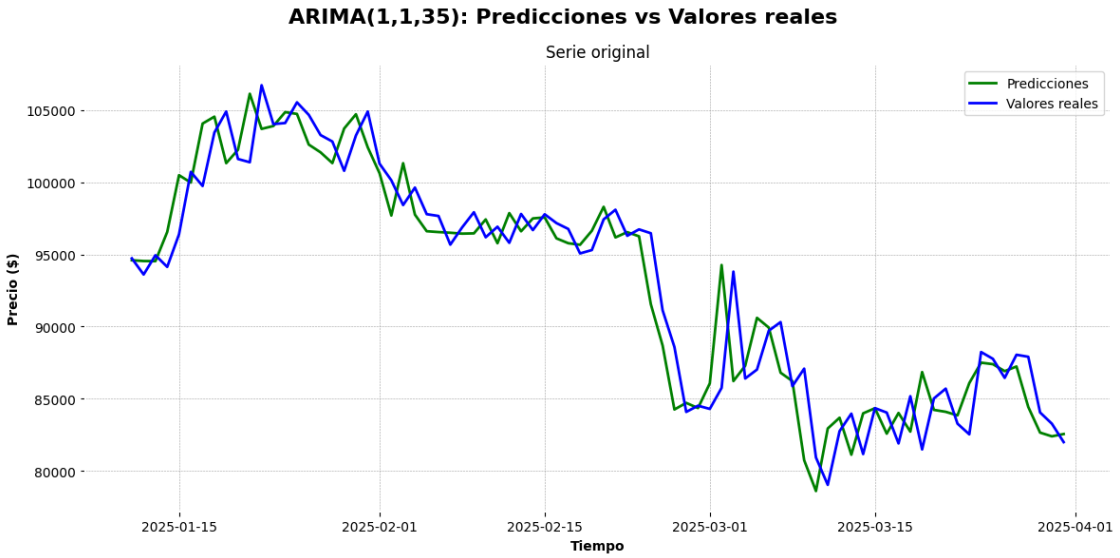
===

```
Ljung-Box (L1) (Q):                   0.28   Jarque-Bera (JB):
46.80
Prob(Q):                              0.60   Prob(JB):
0.00
Heteroskedasticity (H):               1.63   Skew:
0.41
Prob(H) (two-sided):                  0.01   Kurtosis:
4.70
===============================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
[2] Covariance matrix is singular or near-singular, with condition number
1.3e+14. Standard errors may be unstable.
```

Se observa que ninguno de los coeficientes AR y MA son significativamente diferentes de 0. Incluso, la mayor parte de ellos presentan un P-valor cercano a 1 dando cuenta de que son altamente insignificantes.

[28]:
```python
# Inicializar la historia con el set de entrenamiento
history = train.copy()
predictions = []

# Hacer predicciones paso a paso con valores reales
for t in range(len(test)):
    pred = model_fit.apply(history).forecast()[0]
    predictions.append(pred)
    history = pd.concat([history, pd.Series([test.iloc[t]], index=[test.
 ↪index[t]])])
```

[29]:
```python
# --- Gráfico de predicción vs realidad ---
plt.figure(figsize=(12,6))
plt.plot(test.index, test.values, label="Predicciones", color="green")
plt.plot(test.index, predictions, label="Valores reales", color="blue")

plt.suptitle("ARIMA(1,1,35): Predicciones vs Valores reales",␣
 ↪fontsize=TITLE_FONTSIZE)
plt.title("Serie original")
plt.xlabel("Tiempo", fontsize=LABEL_FONTSIZE)
plt.ylabel("Precio ($)", fontsize=LABEL_FONTSIZE)
plt.legend(loc="best")
plt.tight_layout()
plt.grid(True)
plt.show()
```



[30]:
```
metrics = evaluate_model(test, predictions, model_name="ARIMA(1,1,35)")
metrics_list.append(metrics)

print(f"MAE: {metrics['mae']:.1f}")
print(f"MAPE: {metrics['mape']:.4f}")
print(f"RMSE: {metrics['rmse']:.1f}")
```

```
MAE: 1891.7
MAPE: 0.0206
RMSE: 43.5
```

Se observa que el comportamiento de las predicciones del modelo sigue un patrón similar a los valores de test, pero desfasado por un día. Esto es una señal de alarma, ya que el objetivo es que el modelo ayude al usuario a tomar la decisión de si conviene vender o comprar al inicio del día en función del precio estimado para el día siguiente.

Libreria autoarima

La librería autoarima permite automatizar el proceso de selección del mejor modelo de la familia $ARIMA$ testeando diferentes combinaciones de parámetros (p, d, q) y eligiendo aquel que minimiza el criterio elegido, siendo AIC el predeterminado. Anteriormente elegimos los parámetros de nuestro modelo en función de un análisis de significancia mediante los gráficos ACF y PACF. La utilización de la librería autoarima nos permitirá validar o no nuestra elección anterior.

[31]:
```
# Get best ARIMA model structure
initial_model = auto_arima(initial_train['y'],
                           start_p=0, max_p=5,          # AR terms to try
                           start_q=0, max_q=40,         # MA terms to try
                           d = None,                    # Auto-detect d
                           seasonal=False, trace=True,
```

```
                           suppress_warnings=True, error_action='ignore')

# Extract best order
best_order = initial_model.order
print("Best ARIMA order:", best_order)
```

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=5669.513, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=5670.958, Time=0.01 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=5671.116, Time=0.01 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=5668.389, Time=0.01 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=5672.390, Time=0.08 sec

Best model:  ARIMA(0,1,0)(0,0,0)[0]
Total fit time: 0.122 seconds
Best ARIMA order: (0, 1, 0)
```

Es interesante notar que el modelo de la familia ARIMA que mejor performa es ARIMA(0,1,0)(0,0,0), es decir un modelo diferenciado de primer orden sin componentes autorregresivos ni de media movil. Nuestro modelo es entonces:

$$\Delta Y_t = Y_t - Y_{t-1} = \epsilon_t$$

Por lo que la serie diferenciada (el retorno) se modela como ruido blanco.

Al reorganizar la ecuación tenemos:

$$Y_t = Y_{t-1} + \epsilon_t$$

El valor actual de la serie de tiempo original entonces es igual al anterior más ruido blanco, es decir a una caminata aleatoria. Por lo que la predicción para el dia siguiente bajo este modelo es el valor actual (más ruido no modelable), es decir el modelo naive utilizado como baseline.

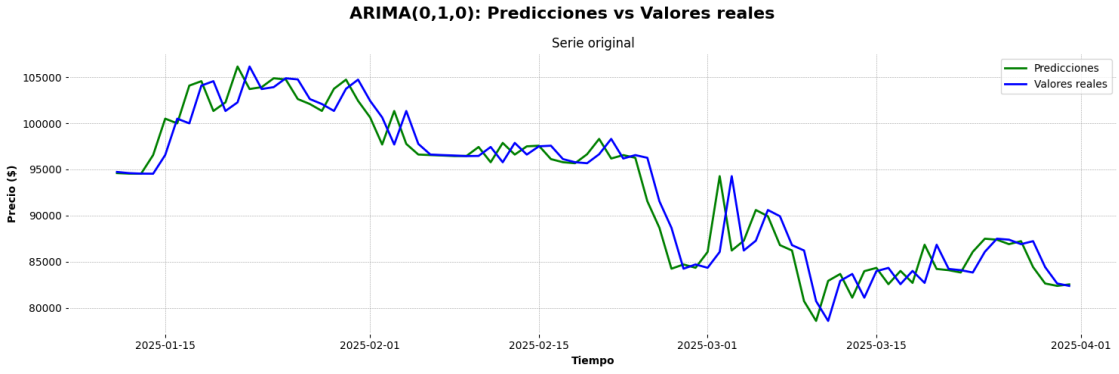
A continuación realizamos el loop de entrenamiento y predicción para comprobar que nuestras deducciones son correctas.

[32]:
```
predictions = []
actuals = []
forecast_dates = []

rolling_train = initial_train.copy()

# Loop through each day in the test set
for i in tqdm(range(len(test_set))):
    next_day = test_set.iloc[i]["ds"]

    # Train ARIMA with fixed order on updated rolling_train
    model = ARIMA(rolling_train["y"], order=best_order)
```

```
    model_fit = model.fit()

    # Predict next day value
    yhat = model_fit.forecast()[0]
    ytrue = test_set.iloc[i]["y"]

    predictions.append(yhat)
    actuals.append(ytrue)
    forecast_dates.append(next_day)

    # Append next day to training set (simulate rolling forward)
    rolling_train = pd.concat([rolling_train, test_set.iloc[[i]]])
```

```
  0%|          | 0/80 [00:00<?, ?it/s]
```

[33]:
```
results = pd.DataFrame({
    'ds': forecast_dates,
    'yhat': predictions,
    'ytrue': actuals
})

plt.figure(figsize=(15, 5))
plt.plot(results['ds'], results['ytrue'], label="Predicciones", color="green")
plt.plot(results['ds'], results['yhat'], label="Valores reales", color="blue")
plt.suptitle("ARIMA(0,1,0): Predicciones vs Valores reales",␣
 ↪fontsize=TITLE_FONTSIZE)
plt.title("Serie original")
plt.xlabel("Tiempo", fontsize=LABEL_FONTSIZE)
plt.ylabel("Precio ($)", fontsize=LABEL_FONTSIZE)
plt.legend(loc="best")
plt.tight_layout()
plt.grid(True)
plt.show()
```



[34]:
```
metrics = evaluate_model(results["ytrue"], results["yhat"],␣
  ↪model_name="ARIMA(0,1,0)")
metrics_list.append(metrics)

print(f"MAE: {metrics['mae']:.1f}")
print(f"MAPE: {metrics['mape']:.4f}")
print(f"RMSE: {metrics['rmse']:.1f}")
```

```
MAE: 1735.7
MAPE: 0.0189
RMSE: 41.7
```

Efectivamente, observamos que el Mean Absolute Percentage Error obtenido es el mismo que el obtenido con el modelo baseline. Este resultado permite comprobar empíricamente que un modelo $ARIMA(0, 1, 0)$ a fines prácticos predice lo mismo que un modelo naive (cada dia predice un valor igual que el observado el dia anterior).

Prophet **Prophet** es un modelo open source desarrollado por Facebook, diseñado para la predicción de series temporales. Este modelo se basa en una descomposición aditiva de la serie en tres componentes principales: **tendencia, estacionalidad y festivos o eventos especiales**.

En primer lugar, en lo que respecta a la **componente de tendencia**, Prophet es capaz de detectar cambios estructurales introduciendo automáticamente puntos de cambio (changepoints) a lo largo de la serie temporal. Durante el entrenamiento, se aplican técnicas de regularización L1 para determinar cuáles de estos puntos son realmente significativos. Esta estrategia permite al modelo modificar la pendiente de la tendencia cuando se producen cambios relevantes en el comportamiento de la serie. Además, el usuario puede incorporar manualmente puntos de cambio basados en conocimiento del dominio, como por ejemplo el inicio de una campaña de marketing o la implementación de una nueva política empresarial.

Por otro lado, Prophet ofrece soporte integrado para múltiples patrones de **estacionalidad**, como los ciclos diarios, semanales o anuales. También es posible definir patrones personalizados, lo que resulta especialmente útil en contextos donde se observan comportamientos periódicos predecibles, como ocurre frecuentemente en los datos de redes sociales o en entornos comerciales.

Finalmente, el modelo permite incluir **festivos o eventos recurrentes**, los cuales ayudan a capturar efectos temporales no estacionales que puedan generar anomalías o picos de actividad. En el caso del comercio electrónico, por ejemplo, esto puede incluir fechas clave como Black Friday o Cyber Monday. Durante la etapa de pronóstico (forecasting), el usuario puede volver a introducir estos eventos usando el mismo identificador utilizado durante el entrenamiento, lo que permite que el modelo incorpore su impacto en la predicción futura.

[35]:
```
# Wrapper function to create prophet instance.
# Includes hyperparameters options for tuning.
def create_prophet_model():
    m = Prophet(
        growth='linear',
        changepoint_range=0.8,
        changepoint_prior_scale=0.05,
```

```
        seasonality_mode='additive',
        yearly_seasonality=True,
        weekly_seasonality=True,
        daily_seasonality=False
    )

    # Add custom seasonality patter (optional)
    # m.add_seasonality(
    #     name='crypto_cycle',
    #     period=3,              # 3-day cycle (adjust accordingly)
    #     fourier_order=5        # seasonal patter complexity
    # )

    return m
```

[36]:
```
predictions = []
actuals = []
forecast_dates = []

rolling_train = initial_train.copy()

# Loop through each day in the test set
for i in tqdm(range(len(test_set))):
    next_day = test_set.iloc[i]["ds"]

    # Fit model using all data up to current day
    model = create_prophet_model()
    model.fit(rolling_train)

    # Predict next day value
    future = pd.DataFrame({'ds': [next_day]})
    forecast = model.predict(future)

    # Store forecasted and actual values
    yhat = forecast["yhat"].values[0]
    ytrue = test_set.iloc[i]["y"]

    predictions.append(yhat)
    actuals.append(ytrue)
    forecast_dates.append(next_day)

    # Append next day to training set (simulate rolling forward)
    rolling_train = pd.concat([rolling_train, test_set.iloc[[i]]])
```

```
  0%|          | 0/80 [00:00<?, ?it/s]
18:28:53 - cmdstanpy - INFO - Chain [1] start processing
18:28:53 - cmdstanpy - INFO - Chain [1] done processing
```

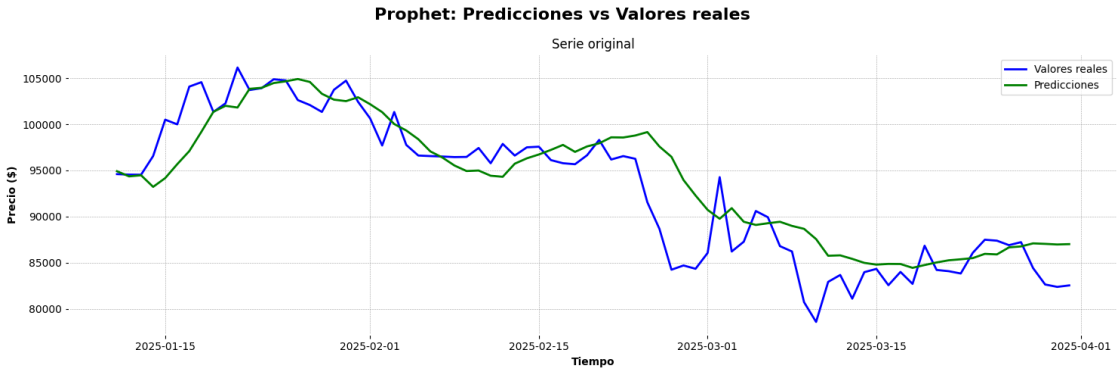
```
18:28:53 - cmdstanpy - INFO - Chain [1] start processing
18:28:53 - cmdstanpy - INFO - Chain [1] done processing
18:28:53 - cmdstanpy - INFO - Chain [1] start processing
18:28:53 - cmdstanpy - INFO - Chain [1] done processing
18:28:53 - cmdstanpy - INFO - Chain [1] start processing
18:28:53 - cmdstanpy - INFO - Chain [1] done processing
18:28:53 - cmdstanpy - INFO - Chain [1] start processing
18:28:53 - cmdstanpy - INFO - Chain [1] done processing
18:28:53 - cmdstanpy - INFO - Chain [1] start processing
18:28:53 - cmdstanpy - INFO - Chain [1] done processing
18:28:53 - cmdstanpy - INFO - Chain [1] start processing
18:28:53 - cmdstanpy - INFO - Chain [1] done processing
18:28:53 - cmdstanpy - INFO - Chain [1] start processing
18:28:53 - cmdstanpy - INFO - Chain [1] done processing
18:28:53 - cmdstanpy - INFO - Chain [1] start processing
18:28:53 - cmdstanpy - INFO - Chain [1] done processing
18:28:54 - cmdstanpy - INFO - Chain [1] start processing
18:28:54 - cmdstanpy - INFO - Chain [1] done processing
18:28:54 - cmdstanpy - INFO - Chain [1] start processing
18:28:54 - cmdstanpy - INFO - Chain [1] done processing
18:28:54 - cmdstanpy - INFO - Chain [1] start processing
18:28:54 - cmdstanpy - INFO - Chain [1] done processing
18:28:54 - cmdstanpy - INFO - Chain [1] start processing
18:28:54 - cmdstanpy - INFO - Chain [1] done processing
18:28:54 - cmdstanpy - INFO - Chain [1] start processing
18:28:54 - cmdstanpy - INFO - Chain [1] done processing
18:28:54 - cmdstanpy - INFO - Chain [1] start processing
18:28:54 - cmdstanpy - INFO - Chain [1] done processing
18:28:54 - cmdstanpy - INFO - Chain [1] start processing
18:28:54 - cmdstanpy - INFO - Chain [1] done processing
18:28:54 - cmdstanpy - INFO - Chain [1] start processing
18:28:54 - cmdstanpy - INFO - Chain [1] done processing
18:28:54 - cmdstanpy - INFO - Chain [1] start processing
18:28:54 - cmdstanpy - INFO - Chain [1] done processing
18:28:55 - cmdstanpy - INFO - Chain [1] start processing
18:28:55 - cmdstanpy - INFO - Chain [1] done processing
18:28:55 - cmdstanpy - INFO - Chain [1] start processing
18:28:55 - cmdstanpy - INFO - Chain [1] done processing
18:28:55 - cmdstanpy - INFO - Chain [1] start processing
18:28:55 - cmdstanpy - INFO - Chain [1] done processing
18:28:55 - cmdstanpy - INFO - Chain [1] start processing
18:28:55 - cmdstanpy - INFO - Chain [1] done processing
18:28:55 - cmdstanpy - INFO - Chain [1] start processing
18:28:55 - cmdstanpy - INFO - Chain [1] done processing
18:28:55 - cmdstanpy - INFO - Chain [1] start processing
18:28:55 - cmdstanpy - INFO - Chain [1] done processing
18:28:55 - cmdstanpy - INFO - Chain [1] start processing
18:28:55 - cmdstanpy - INFO - Chain [1] done processing
```

```
18:28:55 - cmdstanpy - INFO - Chain [1] start processing
18:28:55 - cmdstanpy - INFO - Chain [1] done processing
18:28:55 - cmdstanpy - INFO - Chain [1] start processing
18:28:55 - cmdstanpy - INFO - Chain [1] done processing
18:28:56 - cmdstanpy - INFO - Chain [1] start processing
18:28:56 - cmdstanpy - INFO - Chain [1] done processing
18:28:56 - cmdstanpy - INFO - Chain [1] start processing
18:28:56 - cmdstanpy - INFO - Chain [1] done processing
18:28:56 - cmdstanpy - INFO - Chain [1] start processing
18:28:56 - cmdstanpy - INFO - Chain [1] done processing
18:28:56 - cmdstanpy - INFO - Chain [1] start processing
18:28:56 - cmdstanpy - INFO - Chain [1] done processing
18:28:56 - cmdstanpy - INFO - Chain [1] start processing
18:28:56 - cmdstanpy - INFO - Chain [1] done processing
18:28:56 - cmdstanpy - INFO - Chain [1] start processing
18:28:56 - cmdstanpy - INFO - Chain [1] done processing
18:28:56 - cmdstanpy - INFO - Chain [1] start processing
18:28:56 - cmdstanpy - INFO - Chain [1] done processing
18:28:56 - cmdstanpy - INFO - Chain [1] start processing
18:28:56 - cmdstanpy - INFO - Chain [1] done processing
18:28:56 - cmdstanpy - INFO - Chain [1] start processing
18:28:57 - cmdstanpy - INFO - Chain [1] done processing
18:28:57 - cmdstanpy - INFO - Chain [1] start processing
18:28:57 - cmdstanpy - INFO - Chain [1] done processing
18:28:57 - cmdstanpy - INFO - Chain [1] start processing
18:28:57 - cmdstanpy - INFO - Chain [1] done processing
18:28:57 - cmdstanpy - INFO - Chain [1] start processing
18:28:57 - cmdstanpy - INFO - Chain [1] done processing
18:28:57 - cmdstanpy - INFO - Chain [1] start processing
18:28:57 - cmdstanpy - INFO - Chain [1] done processing
18:28:57 - cmdstanpy - INFO - Chain [1] start processing
18:28:57 - cmdstanpy - INFO - Chain [1] done processing
18:28:57 - cmdstanpy - INFO - Chain [1] start processing
18:28:57 - cmdstanpy - INFO - Chain [1] done processing
18:28:57 - cmdstanpy - INFO - Chain [1] start processing
18:28:57 - cmdstanpy - INFO - Chain [1] done processing
18:28:58 - cmdstanpy - INFO - Chain [1] start processing
18:28:58 - cmdstanpy - INFO - Chain [1] done processing
18:28:58 - cmdstanpy - INFO - Chain [1] start processing
18:28:58 - cmdstanpy - INFO - Chain [1] done processing
18:28:58 - cmdstanpy - INFO - Chain [1] start processing
18:28:58 - cmdstanpy - INFO - Chain [1] done processing
18:28:58 - cmdstanpy - INFO - Chain [1] start processing
18:28:58 - cmdstanpy - INFO - Chain [1] done processing
18:28:58 - cmdstanpy - INFO - Chain [1] start processing
18:28:58 - cmdstanpy - INFO - Chain [1] done processing
18:28:58 - cmdstanpy - INFO - Chain [1] start processing
18:28:58 - cmdstanpy - INFO - Chain [1] done processing
```

```
18:28:58 - cmdstanpy - INFO - Chain [1] start processing
18:28:58 - cmdstanpy - INFO - Chain [1] done processing
18:28:58 - cmdstanpy - INFO - Chain [1] start processing
18:28:58 - cmdstanpy - INFO - Chain [1] done processing
18:28:58 - cmdstanpy - INFO - Chain [1] start processing
18:28:58 - cmdstanpy - INFO - Chain [1] done processing
18:28:59 - cmdstanpy - INFO - Chain [1] start processing
18:28:59 - cmdstanpy - INFO - Chain [1] done processing
18:28:59 - cmdstanpy - INFO - Chain [1] start processing
18:28:59 - cmdstanpy - INFO - Chain [1] done processing
18:28:59 - cmdstanpy - INFO - Chain [1] start processing
18:28:59 - cmdstanpy - INFO - Chain [1] done processing
18:28:59 - cmdstanpy - INFO - Chain [1] start processing
18:28:59 - cmdstanpy - INFO - Chain [1] done processing
18:28:59 - cmdstanpy - INFO - Chain [1] start processing
18:28:59 - cmdstanpy - INFO - Chain [1] done processing
18:28:59 - cmdstanpy - INFO - Chain [1] start processing
18:28:59 - cmdstanpy - INFO - Chain [1] done processing
18:28:59 - cmdstanpy - INFO - Chain [1] start processing
18:28:59 - cmdstanpy - INFO - Chain [1] done processing
18:28:59 - cmdstanpy - INFO - Chain [1] start processing
18:28:59 - cmdstanpy - INFO - Chain [1] done processing
18:28:59 - cmdstanpy - INFO - Chain [1] start processing
18:29:00 - cmdstanpy - INFO - Chain [1] done processing
18:29:00 - cmdstanpy - INFO - Chain [1] start processing
18:29:00 - cmdstanpy - INFO - Chain [1] done processing
18:29:00 - cmdstanpy - INFO - Chain [1] start processing
18:29:00 - cmdstanpy - INFO - Chain [1] done processing
18:29:00 - cmdstanpy - INFO - Chain [1] start processing
18:29:00 - cmdstanpy - INFO - Chain [1] done processing
18:29:00 - cmdstanpy - INFO - Chain [1] start processing
18:29:00 - cmdstanpy - INFO - Chain [1] done processing
18:29:00 - cmdstanpy - INFO - Chain [1] start processing
18:29:00 - cmdstanpy - INFO - Chain [1] done processing
18:29:00 - cmdstanpy - INFO - Chain [1] start processing
18:29:00 - cmdstanpy - INFO - Chain [1] done processing
18:29:00 - cmdstanpy - INFO - Chain [1] start processing
18:29:00 - cmdstanpy - INFO - Chain [1] done processing
18:29:00 - cmdstanpy - INFO - Chain [1] start processing
18:29:00 - cmdstanpy - INFO - Chain [1] done processing
18:29:01 - cmdstanpy - INFO - Chain [1] start processing
18:29:01 - cmdstanpy - INFO - Chain [1] done processing
18:29:01 - cmdstanpy - INFO - Chain [1] start processing
18:29:01 - cmdstanpy - INFO - Chain [1] done processing
18:29:01 - cmdstanpy - INFO - Chain [1] start processing
18:29:01 - cmdstanpy - INFO - Chain [1] done processing
18:29:01 - cmdstanpy - INFO - Chain [1] start processing
18:29:01 - cmdstanpy - INFO - Chain [1] done processing
```

```
18:29:01 - cmdstanpy - INFO - Chain [1] start processing
18:29:01 - cmdstanpy - INFO - Chain [1] done processing
18:29:01 - cmdstanpy - INFO - Chain [1] start processing
18:29:01 - cmdstanpy - INFO - Chain [1] done processing
18:29:01 - cmdstanpy - INFO - Chain [1] start processing
18:29:01 - cmdstanpy - INFO - Chain [1] done processing
18:29:01 - cmdstanpy - INFO - Chain [1] start processing
18:29:01 - cmdstanpy - INFO - Chain [1] done processing
18:29:01 - cmdstanpy - INFO - Chain [1] start processing
18:29:02 - cmdstanpy - INFO - Chain [1] done processing
18:29:02 - cmdstanpy - INFO - Chain [1] start processing
18:29:02 - cmdstanpy - INFO - Chain [1] done processing
18:29:02 - cmdstanpy - INFO - Chain [1] start processing
18:29:02 - cmdstanpy - INFO - Chain [1] done processing
```

[37]:
```python
results = pd.DataFrame({
    'ds': forecast_dates,
    'yhat': predictions,
    'ytrue': actuals
})

plt.figure(figsize=(15, 5))
plt.plot(results['ds'], results['ytrue'], label="Valores reales", color="blue")
plt.plot(results['ds'], results['yhat'], label="Predicciones", color="green")
plt.suptitle("Prophet: Predicciones vs Valores reales", fontsize=TITLE_FONTSIZE)
plt.title("Serie original")
plt.xlabel("Tiempo", fontsize=LABEL_FONTSIZE)
plt.ylabel("Precio ($)", fontsize=LABEL_FONTSIZE)
plt.legend(loc="best")
plt.tight_layout()
plt.grid(True)
plt.show()
```



[38]:
```
metrics = evaluate_model(results["ytrue"], results["yhat"],␣
  ↪model_name="Prophet")
metrics_list.append(metrics)

print(f"MAE: {metrics['mae']:.1f}")
print(f"MAPE: {metrics['mape']:.4f}")
print(f"RMSE: {metrics['rmse']:.1f}")
```

```
MAE: 2582.6
MAPE: 0.0287
RMSE: 50.8
```

Se observa que las predicciones realizadas con prophet son considerablemente inferiores que el baseline de predecir siempre lo mismo que el día anterior. Si bien Prophet es un modelo que suele dar buenos resultados en series con una marcada tendencia y estacionalidad, como pueden ser ventas minoristas, tráfico, o reservas, no da buenos resultados en series tan volátiles y con falta de tendencia a corto plazo como son las criptomonedas.

El baseline, por otro lado, otorga mejores resultados en predicciónes de precios financieros ya que a corto plazo actúan similar a caminatas aleatorias. Prophet, a diferencia del modelo naive o de los modelos autorregresivos, no utiliza Y_{t-k} para predecir un nuevo valor, sino que modela la tendencia, la estacionalidad, y los festivos recurrentes de la serie. En este sentido, las tendencias de los cambios de precio en cryptos suelen ser no lineares y con cambios abruptos, comportamientos que Prophet no captura bien.

XGBoost XGBoost no es un modelo de series temporales, sino un modelo de regresión. Por lo tanto, no se puede utilizar directamente para predecir el comportamiento de la serie temporal. Sin embargo, se puede utilizar como un modelo de regresión para predecir el precio de cierre en función de otras variables, como el volumen de operaciones y los rezagos del precio de cierre.

Carga de datos:

[39]:
```
# Cargamos el conjunto de datos con el formato correcto
df = pd.read_pickle(DATASET_PICKLE_FILE_PATH)
```

Análisis de datos + ingeniería de características:

Nos quedamos solamente con el precio de cierre y creamos 35 rezagos como características, también algunas medias móviles para ver si podemos predecir el precio de cierre en función de estas características. Luego, realizamos el análisis tanto para la serie original como para la serie diferenciada.

[40]:
```
# Nos quedamos con el precio de cierre
df = df[["Close"]]

# Renombramos la columna "Close" a "y"
df.rename(columns={"Close": "y"}, inplace=True)

# Creamos la serie diferenciada
df_delta = df.copy().diff().dropna()
```

```python
# Creamos los rezagos para ambas series
for i in range(1, 35):
    df["rezago_{}".format(i)] = df.y.shift(i)
    df_delta["rezago_{}".format(i)] = df_delta.y.shift(i)

# Agregamos las medias móviles para ambas series
for window in [3, 7, 14]:
    df[f'rolling_mean_{window}'] = df['y'].rolling(window=window).mean()
    df_delta[f'rolling_mean_{window}'] = df_delta['y'].rolling(window=window).
 ↪mean()

# Eliminamos los valores NaN generados por los rezagos
df.dropna(inplace=True)
df_delta.dropna(inplace=True)

print(f"Cantidad de filas: {len(df)}")
print(f"Cantidad de filas: {len(df_delta)}")

df.head()
```

```
Cantidad de filas: 362
Cantidad de filas: 361
```

[40]:

	y	rezago_1	rezago_2	rezago_3	rezago_4	rezago_5	\
Open Time							
2024-04-04	68487.79	65963.28	65463.99	69649.80	71280.01	69582.18	
2024-04-05	67820.62	68487.79	65963.28	65463.99	69649.80	71280.01	
2024-04-06	68896.00	67820.62	68487.79	65963.28	65463.99	69649.80	
2024-04-07	69360.39	68896.00	67820.62	68487.79	65963.28	65463.99	
2024-04-08	71620.00	69360.39	68896.00	67820.62	68487.79	65963.28	

	rezago_6	rezago_7	rezago_8	rezago_9	…	rezago_28	rezago_29	\
Open Time					…			
2024-04-04	69850.54	70780.60	69469.99	69988.00	…	66823.17	66074.04	
2024-04-05	69582.18	69850.54	70780.60	69469.99	…	68124.19	66823.17	
2024-04-06	71280.01	69582.18	69850.54	70780.60	…	68313.27	68124.19	
2024-04-07	69649.80	71280.01	69582.18	69850.54	…	68955.88	68313.27	
2024-04-08	65463.99	69649.80	71280.01	69582.18	…	72078.10	68955.88	

	rezago_30	rezago_31	rezago_32	rezago_33	rezago_34	\
Open Time						
2024-04-04	63724.01	68245.71	63113.97	61987.28	62387.90	
2024-04-05	66074.04	63724.01	68245.71	63113.97	61987.28	
2024-04-06	66823.17	66074.04	63724.01	68245.71	63113.97	
2024-04-07	68124.19	66823.17	66074.04	63724.01	68245.71	
2024-04-08	68313.27	68124.19	66823.17	66074.04	63724.01	

```
            rolling_mean_3  rolling_mean_7  rolling_mean_14
Open Time
2024-04-04    66638.353333    68611.084286     68242.345000
2024-04-05    67423.896667    68321.095714     68529.772143
2024-04-06    68401.470000    68223.070000     68880.200000
2024-04-07    68692.336667    67948.838571     69033.800000
2024-04-08    69958.796667    68230.295714     69158.085000

[5 rows x 38 columns]
```

Modelado:

Definimos funciones auxiliares para dividir y graficar los datos:

[41]:
```python
def timeseries_train_test_split(X, y, test_size):
    test_index = int(len(X) * (1 - test_size))

    X_train = X.iloc[:test_index]
    y_train = y.iloc[:test_index]
    X_test = X.iloc[test_index:]
    y_test = y.iloc[test_index:]

    return X_train, X_test, y_train, y_test
```

Serie original:

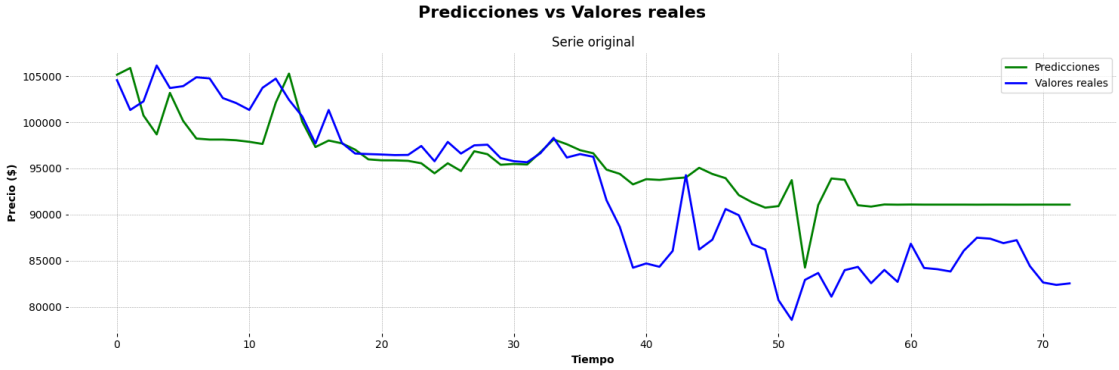
[42]:
```python
y = df["y"]
X = df.drop(columns=["y"])

X_train, X_test, y_train, y_test = timeseries_train_test_split(
    X, y, test_size=0.2
)  # 80% train, 20% test

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

xgb = XGBRegressor(random_state=42, verbosity=0)
xgb.fit(X_train_scaled, y_train)
```

[42]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             feature_weights=None, gamma=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=None, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
             max_leaves=None, min_child_weight=None, missing=nan,
```

```
monotone_constraints=None, multi_strategy=None, n_estimators=None,
n_jobs=None, num_parallel_tree=None, …)
```

Predicciones:

[43]:
```
y_pred = xgb.predict(X_test_scaled)

plt.figure(figsize=(15, 5))
plt.plot(y_pred, label="Predicciones", color="green")
plt.plot(y_test.values, label="Valores reales", color="blue")
plt.suptitle("Predicciones vs Valores reales", fontsize=TITLE_FONTSIZE)
plt.title("Serie original")
plt.xlabel("Tiempo", fontsize=LABEL_FONTSIZE)
plt.ylabel("Precio ($)", fontsize=LABEL_FONTSIZE)
plt.legend(loc="best")
plt.tight_layout()
plt.grid(True)
plt.show()
```



Resultados:

[44]:
```
metrics = evaluate_model(y_pred, y_test, model_name="XGBoost (ST original)")
metrics_list.append(metrics)

print(f"MAE: {metrics['mae']:.1f}")
print(f"MAPE: {metrics['mape']:.4f}")
print(f"RMSE: {metrics['rmse']:.1f}")

print(f"R2: {xgb.score(X_test_scaled, y_test)}")
```

```
MAE: 4319.0
MAPE: 0.0461
RMSE: 65.7
R2: 0.4938462833874311
```

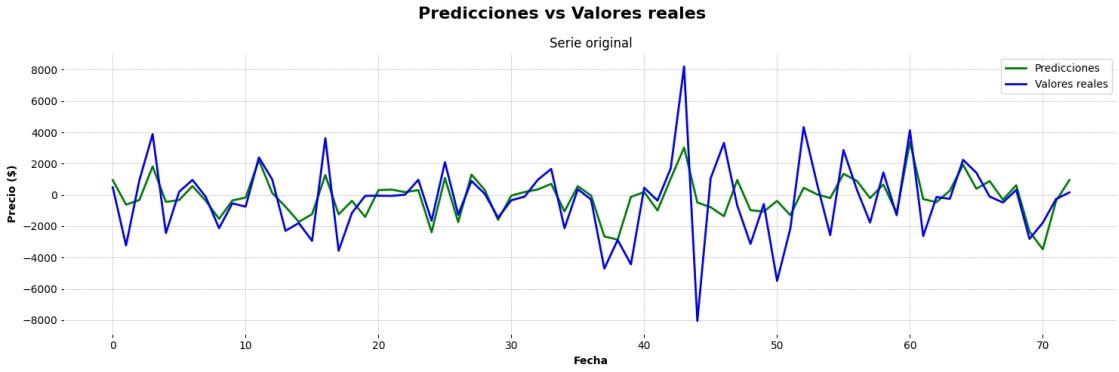
Serie diferenciada:

[45]:
```
y = df_delta["y"]
X = df_delta.drop(columns=["y"])

X_train, X_test, y_train, y_test = timeseries_train_test_split(
    X, y, test_size=0.2
)  # 80% train, 20% test

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

xgb = XGBRegressor(random_state=42, verbosity=0)
xgb.fit(X_train_scaled, y_train)
```

[45]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             feature_weights=None, gamma=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=None, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
             max_leaves=None, min_child_weight=None, missing=nan,
             monotone_constraints=None, multi_strategy=None, n_estimators=None,
             n_jobs=None, num_parallel_tree=None, …)
```

Predicciones:

[46]:
```
y_pred = xgb.predict(X_test_scaled)

plt.figure(figsize=(15, 5))
plt.plot(y_pred, label="Predicciones", color="green")
plt.plot(y_test.values, label="Valores reales", color="blue")
plt.suptitle("Predicciones vs Valores reales", fontsize=TITLE_FONTSIZE)
plt.title("Serie original")
plt.xlabel("Fecha", fontsize=LABEL_FONTSIZE)
plt.ylabel("Precio ($)", fontsize=LABEL_FONTSIZE)
plt.legend(loc="best")
plt.tight_layout()
plt.grid(True)
plt.show()
```



Resultados:

[47]:
```
metrics = evaluate_model(y_pred, y_test, model_name="XGBoost (ST diferenciada)")
metrics_list.append(metrics)

print(f"MAE: {metrics['mae']:.1f}")
print(f"MAPE: {metrics['mape']:.4f}")
print(f"RMSE: {metrics['rmse']:.1f}")

print(f"R2: {xgb.score(X_test_scaled, y_test)}")
```

```
MAE: 1220.1
MAPE: 2.9078
RMSE: 34.9
R2: 0.4210888795399247
```

Podemos observar que el modelo XGBoost tiene bajo performance dado los resultados obtenidos. Sin embargo, es importante tener en cuenta que el modelo no se ha ajustado adecuadamente y que se pueden realizar mejoras en el proceso de ajuste y selección de hiperparámetros.

LSTM Las LSTM son un tipo de red neuronal diseñada para analizar datos secuenciales. Es especialmente buena para recordar detalles importantes durante periodos de tiempo. Esto la hace útil para predecir valores futuros en datos de series temporales, ya que puede identificar patrones complejos en los datos.

Carga de datos:

[48]:
```
# Cargamos el conjunto de datos con el formato correcto
df = pd.read_pickle(DATASET_PICKLE_FILE_PATH)
```

Analisis de datos + ingeniería de características:

[49]:
```
# Nos quedamos con el precio de cierre como una serie.
data = df['Close'].values
data = data.reshape(-1, 1)
print(f'Shape de los datos: {data.shape}')
```

```
Shape de los datos: (396, 1)
```

Modelado

Ahora, para la utilización de redes LSTM se espera que la entrada (recordando de NLP) sea una secuencia de n valores en donde se quiere predecir el valor $n+1$. En este caso, se utilizará una ventana w, en donde dado el dato desde el tiempo $t-w$ hasta el tiempo t, se debe predecir el valor en el tiempo $t+1$ (para el caso de la arquitectura que crearemos *many-to-one*). El tamaño de la ventana indica cuantos datos miramos cuando queremos realizar una predicción. Para este caso, elegimos $w = 35$, con lo que venimos trabajando. Para esto definimos una función que permite crear estas ventanas:

[50]:
```
def create_sequences(data, time_steps):
    X, y = [], []
    for i in range(len(data) - time_steps):
        X.append(data[i : i + time_steps, 0])
        y.append(data[i + time_steps, 0])
    return np.array(X), np.array(y)
```

Creamos las secuencias de datos para el modelo LSTM:

[51]:
```
# Normalizamos los datos entre 0 y 1
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)

# Utilizar una ventana de 35 días para predecir el precio del siguiente día
time_steps = 35
X, y = create_sequences(scaled_data, time_steps)
print(f'Shape de X: {X.shape}')
print(f'Shape de y: {y.shape}')

# Es necesario un reshape para la entrada de la LSTM (n_samples, time_steps,␣
 ↪n_features)
X = X.reshape(X.shape[0], X.shape[1], 1) # Creo que se puede utilizar una␣
 ↪función expand_dims de numpy para esto también.
print(f'Shape de X después del reshape: {X.shape}')

# Hacemos el spliting de los datos en train y test
# 80% para entrenamiento y 20% para test
train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
print(f'Shapes (X_train, X_test, y_train, y_test): {X_train.shape, X_test.
 ↪shape, y_train.shape, y_test.shape}')
```

```
Shape de X: (361, 35)
Shape de y: (361,)
Shape de X después del reshape: (361, 35, 1)
Shapes (X_train, X_test, y_train, y_test): ((288, 35, 1), (73, 35, 1), (288,),
```

(73,))

Definimos la arquitectura de la red LSTM (como prueba de concepto utilizamos una red simple):

[52]:
```
model = Sequential()
model.add(Input(shape=(time_steps, 1))) # Capa de entrada
model.add(LSTM(50, return_sequences=False)) # Capa LSTM con 50 unidades
model.add(Dense(1)) # Capa de salida con una unidad (predicción del precio)

# Compilamos el modelo con adam y mse como función de pérdida
model.compile(optimizer="adam", loss="mean_squared_error")
```

Mostramos la arquitectura generada:

[53]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 50)	10,400
dense (Dense)	(None, 1)	51

Total params: 10,451 (40.82 KB)

Trainable params: 10,451 (40.82 KB)

Non-trainable params: 0 (0.00 B)

Entrenamos el modelo en 20 épocas:

[54]:
```
history = model.fit(X_train, y_train, epochs=20, batch_size=32,␣
↪validation_data=(X_test, y_test))
```

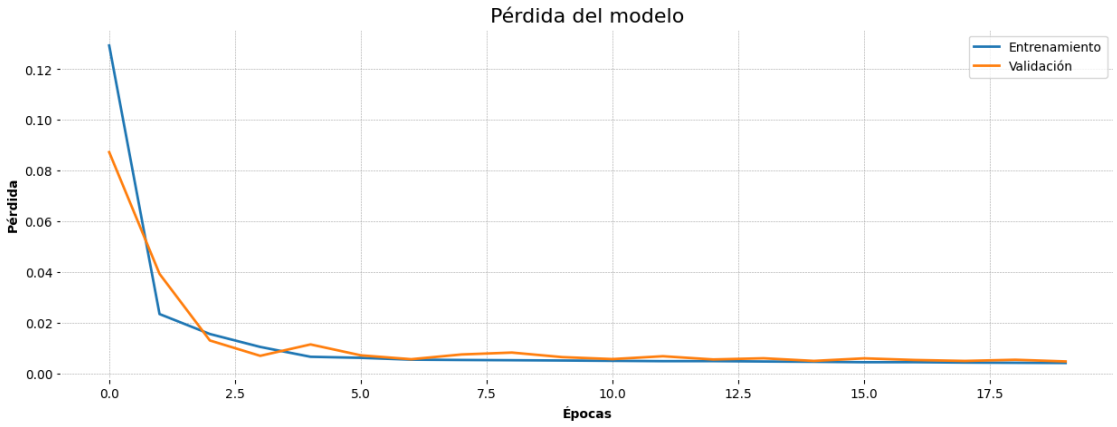
```
Epoch 1/20
9/9             1s 23ms/step - loss:
0.1661 - val_loss: 0.0873
Epoch 2/20
9/9             0s 8ms/step - loss:
0.0225 - val_loss: 0.0394
Epoch 3/20
9/9             0s 8ms/step - loss:
0.0192 - val_loss: 0.0131
Epoch 4/20
```

```
9/9             0s 9ms/step - loss:
0.0106 - val_loss: 0.0070
Epoch 5/20
9/9             0s 9ms/step - loss:
0.0060 - val_loss: 0.0115
Epoch 6/20
9/9             0s 8ms/step - loss:
0.0062 - val_loss: 0.0072
Epoch 7/20
9/9             0s 9ms/step - loss:
0.0068 - val_loss: 0.0057
Epoch 8/20
9/9             0s 8ms/step - loss:
0.0048 - val_loss: 0.0076
Epoch 9/20
9/9             0s 8ms/step - loss:
0.0054 - val_loss: 0.0083
Epoch 10/20
9/9             0s 9ms/step - loss:
0.0056 - val_loss: 0.0065
Epoch 11/20
9/9             0s 8ms/step - loss:
0.0042 - val_loss: 0.0058
Epoch 12/20
9/9             0s 8ms/step - loss:
0.0051 - val_loss: 0.0069
Epoch 13/20
9/9             0s 8ms/step - loss:
0.0061 - val_loss: 0.0056
Epoch 14/20
9/9             0s 8ms/step - loss:
0.0052 - val_loss: 0.0061
Epoch 15/20
9/9             0s 8ms/step - loss:
0.0049 - val_loss: 0.0050
Epoch 16/20
9/9             0s 8ms/step - loss:
0.0042 - val_loss: 0.0061
Epoch 17/20
9/9             0s 8ms/step - loss:
0.0048 - val_loss: 0.0054
Epoch 18/20
9/9             0s 8ms/step - loss:
0.0045 - val_loss: 0.0050
Epoch 19/20
9/9             0s 8ms/step - loss:
0.0044 - val_loss: 0.0055
Epoch 20/20
```

```
9/9             0s 8ms/step - loss:
0.0040 - val_loss: 0.0048
```

Mostramos la gráfica de entrenamiento:

[55]:
```
# Gráfica de entrenamiento y validación
plt.figure(figsize=(15, 5))
plt.plot(history.history["loss"], label="Entrenamiento")
plt.plot(history.history["val_loss"], label="Validación")
plt.title("Pérdida del modelo", fontsize=TITLE_FONTSIZE)
plt.xlabel("Épocas", fontsize=LABEL_FONTSIZE)
plt.ylabel("Pérdida", fontsize=LABEL_FONTSIZE)
plt.legend()
plt.show()
```



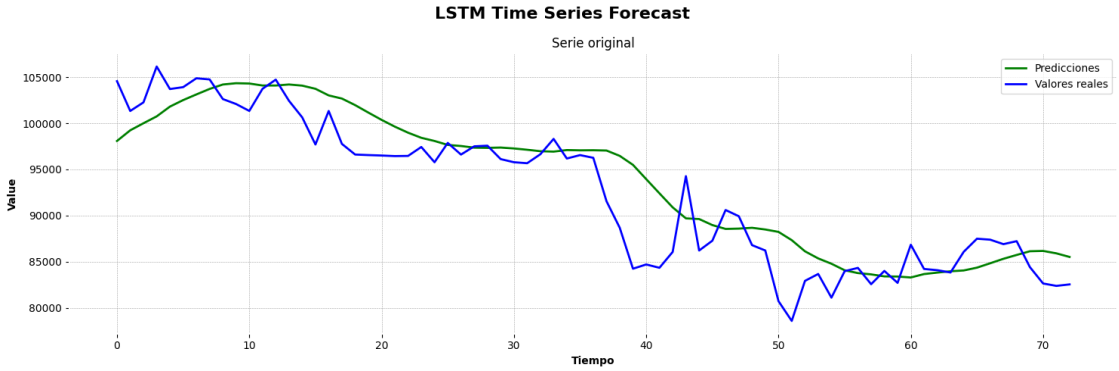
Podemos observar que el modelo ha convergido tanto en la función de pérdida como en validación, por lo que no se observa un sobreajuste. También se puede obseravar que luego de la época 10, la función de pérdida de validación se estabiliza, por lo que no sería necesario entrenar el modelo por más épocas.

Predicciones:

[56]:
```
y_pred = model.predict(X_test)

# Aplicamos la inversa de la normalización a las predicciones y los valores␣
 ↪reales (dado que le habíamos aplicado a todos los datos)
y_pred_rescaled = scaler.inverse_transform(y_pred)
y_test_rescaled = scaler.inverse_transform(y_test.reshape(-1, 1))

# Graficamos los resultados
plt.figure(figsize=(15,5))
plt.plot(y_pred_rescaled, label='Predicciones', color='green')
plt.plot(y_test_rescaled, label='Valores reales', color='blue')
plt.suptitle('LSTM Time Series Forecast', fontsize=TITLE_FONTSIZE)
```

```
plt.title("Serie original")
plt.xlabel('Tiempo', fontsize=LABEL_FONTSIZE)
plt.ylabel('Value', fontsize=LABEL_FONTSIZE)
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```

```
3/3             0s 41ms/step
```



Resultados:

[57]:
```
metrics = evaluate_model(y_pred, y_test, model_name="LSTM")
metrics_list.append(metrics)

print(f"MAE: {metrics['mae']:.1f}")
print(f"MAPE: {metrics['mape']:.4f}")
print(f"RMSE: {metrics['rmse']:.1f}")

print(f"R2: {model.evaluate(X_test, y_test)}")
```

```
MAE: 0.1
MAPE: 0.0696
RMSE: 0.2
3/3             0s 7ms/step - loss:
0.0045
R2: 0.004834292456507683
```

En comparación con XGBoost, podemos observar que el modelo obtuvo mejores valores en todas las métricas. Tampoco hubo sobre ajuste, por lo que se podría intuir que es un mejor modelo (esto mirando la serie original).

0.2.6 Conclusiones

A lo largo del proyecto realizamos el análisis de una serie de tiempo con el precio de cierre de la criptomoneda Bitcoin a partir de datos obtenidos de la plataforma Binance. Posterior al análisis

exploratorio de los datos, y en función del conocimiento de la serie obtenido por el proceso, ajustamos diferentes modelos a los datos y evaluamos su potencial predictivo para el dia siguiente. Las metricas de evaluación de las predicciones pueden verse a continuación.

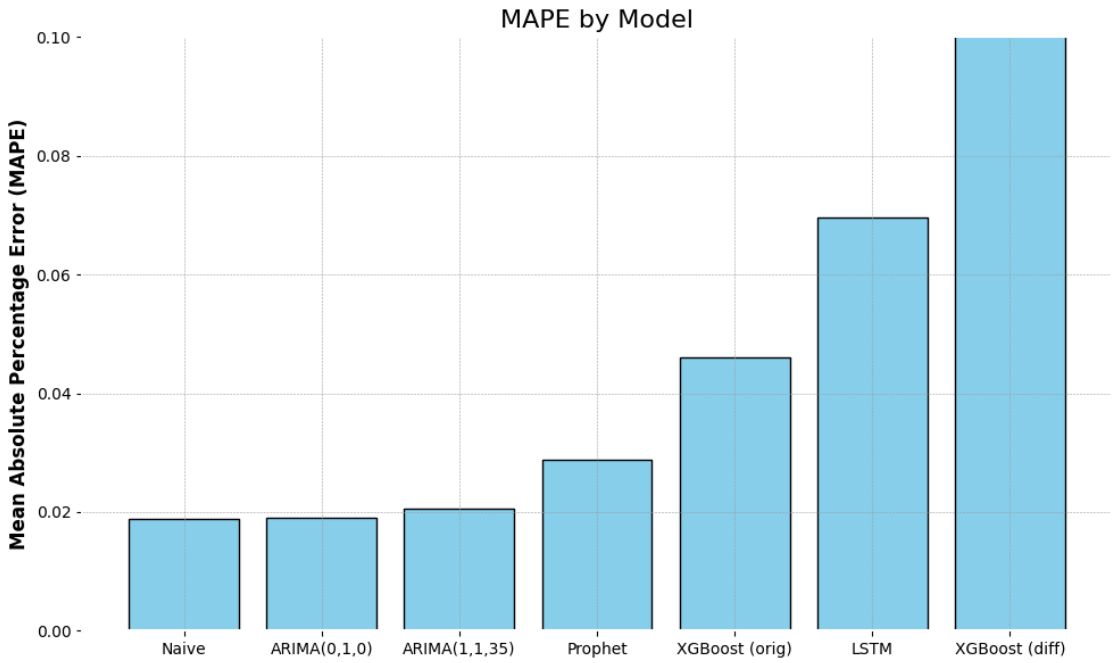
[58]:
```
df_metrics = pd.DataFrame(metrics_list)
df_metrics.round(3)
```

[58]:

	model	mae	mape	rmse
0	Naive	1735.726	0.019	41.662
1	ARIMA(1,1,35)	1891.662	0.021	43.493
2	ARIMA(0,1,0)	1735.726	0.019	41.662
3	Prophet	2582.551	0.029	50.819
4	XGBoost (ST original)	4319.032	0.046	65.719
5	XGBoost (ST diferenciada)	1220.061	2.908	34.929
6	LSTM	0.052	0.070	0.229

Ya que algunos modelos fueron ajustados sobre al serie de los retornos mientras que otros fueron ajustados sobre la serie original, el Mean Absolute Percentage Error (MAPE) nos otorga la mejor medida de comparación.

[59]:
```
# Sort if you want (optional)
df_sorted = df_metrics.sort_values("mape", ascending=True)

filt = df_metrics["model"] == "XGBoost (ST original)"
df_sorted.loc[filt, "model"] = "XGBoost (orig)"
filt = df_sorted["model"] == "XGBoost (ST diferenciada)"
df_sorted.loc[filt, "model"] = "XGBoost (diff)"

# Extract data
models = df_sorted["model"]
mae_values = df_sorted["mape"]

# Plot
plt.figure(figsize=(10, 6))
plt.bar(models, mae_values, color="skyblue", edgecolor="black")

# Labels and title
plt.title("MAPE by Model", fontsize=16)
plt.ylabel("Mean Absolute Percentage Error (MAPE)", fontsize=12)
plt.xticks(rotation=0)  # Rotate x labels if needed

plt.ylim(0, .1)

# Layout and show
plt.tight_layout()
plt.show()
```



Se observa que debido a la dificultad de modelar la serie de tiempo analizada, la utilización del un modelo Naive que siempre prediga el valor del dia anterior otorga la mejor performance. La volatilidad del precio de las criptomonedas suele verse fuertemente afectada por imprevisibles no recurrentes lo que genera que la sola utilización de modelos out-of-the-box no es suficiente para capturar esas dinámicas. Para mejorar el baseline, entonces, será necesario incorporar conocimiento de dominio en tiempo real al análisis de la serie temporal.

0.2.7 Mejoras futuras

- En el modelo XBoost, se podrían agregar más características, como más rezagos o más medias moviles. También se le puede agregar un poco de ruido a estas nuevas características e incluir la variable volumen para ver si afecta el precio de cierre. Finalmente, se debe realizar optimización de hiperparámetros.
- En el modelo LSTM, dado que en la época 10 ya converge, habría que agregar una función callback *earlyStopping* de Keras. También sería positivo realizar una búsqueda de hiperparámetros para encontrar la mejor arquitectura y los mejores hiperparámetros para el modelo. Por último, se podría agregar un dropout para evitar el sobreajuste y probar con la serie diferenciada.