FACULTAD DE ESTUDIOS ESTADÍSTICOS

GRADO EN ESTADISTICA APLICADA

Curso 2024/2025

Trabajo de Fin de Grado

TÍTULO: Diseño y Evaluación de Estrategias de Trading Algorítmico en Bitcoin mediante Modelos Predictivos de Aprendizaje Automático.

***Alumno*: Francisco Javier Bello Pérez**

***Tutor*: Javier Castro Cantalejo**

Junio (o Septiembre) de 2025

UNIVERSIDAD COMPLUTENSE
MADRID

Tabla de contenido

Abstract

This Final Degree Project explores the design and evaluation of algorithmic trading strategies for Bitcoin using machine learning prediction models. In the context of a highly volatile and continuous financial market, the study builds a robust and automated data processing pipeline, incorporating technical indicators, macroeconomic variables, and engineered features derived from historical price series. Several models are developed and tested, from simple regression trees to ensemble techniques such as Bagging, Random Forest, Gradient Boosting, and XGBoost. These models aim to predict directional movements of the price, particularly impulses and pullbacks, and are assessed through tailored performance metrics like average return per operation and directional accuracy. Furthermore, the study simulates both simple and adaptive investment strategies to evaluate the predictive power of the models in real trading scenarios. The final model selection relies on a forward-chaining cross-validation framework, focusing not only on profitability but also on robustness against market volatility. The results lay the groundwork for future developments, such as incorporating deep learning models, GARCH volatility models, and strategy segmentation according to market regime (bullish, bearish, or sideways). This work highlights the potential of machine learning for financial forecasting, as well as the challenges of building consistent and interpretable models in dynamic and uncertain environments.

Resumen

Este Trabajo de Fin de Grado explora el diseño y la evaluación de estrategias de trading algorítmico para Bitcoin utilizando modelos de predicción basados en aprendizaje automático. En el contexto de un mercado financiero altamente volátil y continuo, el estudio construye una canalización robusta y automatizada de procesamiento de datos, incorporando indicadores técnicos, variables macroeconómicas y características derivadas de series históricas de precios. Se desarrollan y prueban varios modelos, desde árboles de regresión simples hasta técnicas de ensamblado como Bagging, Random Forest, Gradient Boosting y XGBoost. Estos modelos tienen como objetivo predecir los movimientos direccionales del precio, en particular los impulsos y retrocesos, y se evalúan mediante métricas de rendimiento diseñadas específicamente, como el retorno medio por operación y la precisión direccional.
Además, el estudio simula estrategias de inversión tanto simples como adaptativas para evaluar el poder predictivo de los modelos en escenarios reales de trading. La selección final del modelo se basa en un esquema de validación cruzada con estructura temporal (forward-chaining), enfocándose no solo en la rentabilidad, sino también en la robustez frente a la volatilidad del mercado. Los resultados sientan las bases para desarrollos futuros, como la incorporación de modelos de aprendizaje profundo, modelos de volatilidad tipo GARCH y la segmentación de estrategias según el régimen del mercado (alcista, bajista o lateral). Este trabajo pone de manifiesto el potencial del aprendizaje automático para la predicción financiera, así como los desafíos de construir modelos consistentes e interpretables en entornos dinámicos e inciertos.

1. Introducción

1.1. Contexto y justificación del estudio

El mercado de las criptomonedas ha tenido un gran crecimiento en los últimos años, atrayendo a inversores, traders y entusiastas de todo el mundo. Con un mercado altamente volátil y en constante evolución, la capacidad de analizar y comprender los movimientos de precios se ha vuelto esencial para tomar decisiones en este espacio financiero.

Para llevar a cabo un análisis riguroso y efectivo de este mercado, es fundamental contar con una base de datos completa y bien estructurada que contenga datos históricos de precios junto con una variedad de indicadores técnicos relevantes. Estos indicadores proporcionan información muy importante sobre la dinámica del mercado, ayudando a identificar patrones, tendencias y posibles puntos de inflexión en el comportamiento de los activos digitales.

El presente Trabajo de Fin de Grado se desarrolla en el ámbito del trading algorítmico, un entorno que ofrece particularidades técnicas y estructurales que lo hacen especialmente atractivo para el desarrollo y validación de modelos predictivos avanzados mediante técnicas de machine learning (Athey, S., & Imbens, G., 2019).

Una de las principales motivaciones para trabajar con este tipo de activos reside en que este mercado opera de forma continua las 24 horas del día, los 7 días de la semana (24/7). Esta disponibilidad permanente de datos permite aplicar modelos de aprendizaje automático sobre series temporales más densas y con mayor granularidad, lo que aumenta el potencial para encontrar patrones temporales y oportunidades operativas.

Además, el ecosistema cripto se caracteriza por un alto grado de transparencia y descentralización, lo cual permite el acceso a información que en los mercados tradicionales suele ser privada o restringida (Nakamoto, S., 2008). En este sentido, cabe destacar la posibilidad de consultar en tiempo real los denominados libros de órdenes (order books). Estos registros muestran el conjunto de órdenes de compra y venta abiertas para un activo, indicando tanto los precios como los volúmenes. El análisis de estos datos puede aportar información clave sobre la presión de oferta y demanda y detectar posibles zonas de liquidez o manipulación de precios. Sin embargo, por limitaciones técnicas y de presupuesto, esta fuente de datos no ha sido incorporada, pero representa una vía prometedora para futuras extensiones del modelo.

Asimismo, otro aspecto diferenciador del mercado de criptomonedas es la disponibilidad de métricas derivadas del comportamiento interno de las propias redes blockchain, conocidas como métricas on-chain (King, J. C., Dale, R., & Amigó, J. M. ,2024). Estas incluyen variables como el volumen de transacciones en la red, la actividad de las carteras, el número de direcciones activas o la dominancia del Bitcoin frente al resto de criptomonedas (altcoins). Esta última, por ejemplo, permite identificar flujos de capital entre activos y fases de rotación de mercado, lo cual puede mejorar la precisión de los modelos predictivos. No obstante, debido a restricciones presupuestarias y técnicas, este tipo de métricas no se han incluido en este trabajo. Se espera que, en futuras investigaciones, con mayor disponibilidad de recursos y acceso a plataformas de pago, puedan incorporarse estos datos para construir modelos más robustos y representativos de la dinámica real del mercado.

En resumen, la elección del mercado de criptomonedas responde tanto a motivos técnicos (mayor disponibilidad y transparencia de los datos) como operativos (mercado ininterrumpido y altamente dinámico), lo que lo convierte en un campo de pruebas ideal para evaluar la eficacia de modelos de machine learning aplicados al trading algorítmico (López de Prado, M. ,2018). La riqueza de fuentes de información accesibles como libros de órdenes, métricas on-chain, datos históricos y en tiempo real ofrece un marco idóneo para continuar desarrollando modelos predictivos cada vez más precisos y adaptativos.

Adicionalmente, otro factor clave en la decisión de diseñar un algoritmo de trading radica en superar una de las principales limitaciones del trading convencional: el gran factor emocional y la constante de vigilancia de las operaciones u oportunidades. La toma de decisiones en los

mercados financieros se ve perjudicada por el miedo, la aversión a la pérdida o la euforia, lo que puede perjudicar a la estrategia inicial (García, D., & Schweitzer, F. ,2015). Además, la naturaleza impredecible y continua del mercado de criptomonedas exige una supervisión constante, lo cual no siempre es viable operar a mano. Por esto último, el desarrollo de un sistema automatizado permite aplicar reglas objetivas y reproducibles, operando de forma continua sin intervención humana. (U.S. Senate Committee on Homeland Security and Governmental Affairs. ,2024). Por tanto, este proyecto no solo busca optimizar los resultados operativos, sino también avanzar hacia un enfoque más racional, sistemático y escalable en la toma de decisiones de inversión (Pardo, Á., 2020).

Si bien el análisis y modelado de variables financieras como las que aquí se presentan podrían plantearse mediante series temporales, como modelos GARCH para capturar la volatilidad condicional, o modelos ARIMAX para incorporar variables explicativas exógena, en este trabajo hemos optado por un enfoque basado en modelos de machine learning. El motivo principal es por la flexibilidad que estos modelos ofrecen para capturar relaciones no lineales y complejas entre variables, y en la posibilidad de construir en cada instante del tiempo un conjunto de variables explicativas que recogen la información pasada del mercado. Así, cada observación se trata como un ejemplo independiente, aunque estructurado temporalmente, lo que permite aplicar técnicas modernas de predicción como son los árboles de regresión, Random Forest, Bagging, Gradient Boosting o XGBoost.

1.2. Objetivo de la investigación

1.1.1. Objetivo general

El objetivo principal de esta investigación es desarrollar una estructura robusta de análisis cuantitativo orientada a la predicción de movimientos bursátiles, en particular impulsos y retrocesos del precio, mediante la implementación desde modelos más simples como árboles hasta modelos de aprendizaje automático como Gradient Boosting y XGBoost. Se busca establecer una metodología escalable que permita evaluar distintas estrategias de inversión algorítmica basadas en predicciones, con el fin de construir las bases para futuras investigaciones.

1.1.2. Objetivos Específicos

Para lograr el objetivo general, se proponen los siguientes objetivos específicos:

- **Determinar fuentes de recogida de datos:** Investigar las posibles fuentes de recogida de datos. Nos va a interesar que sean gratuitas y que puedan recogerse tanto datos históricos como datos a tiempo real. Los históricos para la creación de los modelos y los de tiempo real para la puesta en práctica en futuro las estrategias.
- **Automatización del proceso de datos:** Diseñar un algoritmo capaz de realizar la extracción de datos en tiempo real, su preprocesamiento e imputación, y la creación de las variables y transformarlas para la posterior aplicación en los modelos predictivos.
- **Desarrollo de modelos de predicción:** Implementar una serie de modelos predictivos con complejidad creciente empezando por árboles de regresión, Bagging, Random Forest, hasta Gradient Boosting y XGBoost para predecir impulsos y retrocesos de activos financieros.
- **Comparación de estrategias de inversión:** Simular y evaluar estrategias de inversión algorítmica desde enfoques simples (entrada y salida fija tras X velas) hasta estrategias más complejas.
- **Evaluación cuantitativa de resultados:** Analizar el rendimiento de los modelos utilizando métricas diseñadas para cuantificar la calidad del modelo de inversión, como la rentabilidad promedio por operación (directional_error_test), porcentaje de operaciones acertadas en datos de validación y su relación con la volatilidad del activo.
- **Proyección futura:** Sentar las bases para el desarrollo de sistemas más complejos de predicción y gestión de señales en mercados financieros, incorporando en futuras líneas de trabajo redes neuronales profundas y técnicas avanzadas de ensamblado de modelos.

1.3. Hipótesis y preguntas de la investigación orientadas a la predicción

Para esta investigación vamos a establecer las siguientes hipótesis, que se pondrán a prueba a lo largo del estudio de las predicciones y análisis comparativo de modelos:

- **H1:** Los modelos avanzados como Gradient Boosting y XGBoost presentan una mayor capacidad predictiva para identificar impulsos y retrocesos en comparación con modelos más simples como árboles de decisión o Bagging. Esto lo mediremos con la rentabilidad promedio por operación.
- **H2:** Las estrategias de inversión que ofrecen mejores resultados son las más sencillas o tiene una gran importancia estudiar como ajustar diferentes parámetros realizando estrategias más complejas.

1.4. Metodología

1.4.1. Diseño del estudio

En este estudio adoptaremos un enfoque más cuantitativo, de tipo aplicado y carácter experimental, en el que se implementa y valida un sistema automatizado de inversión, en un mercado real. Se entrena un conjunto de modelos de predicción sobre datos reales de mercado, evaluando su rendimiento en condiciones similares a las de un entorno real.

1.4.2. Etapas metodológicas

1. **Extracción y tratamiento de datos:** Se implementa un algoritmo automatizado que recoge datos en tiempo real, incluyendo precios, variables de contexto como el VIX y tipos de interés.

2. **Creación de variables predictoras:** Se generan variables derivadas de las variables de los precios iniciales. Aquí nacen variables como los indicadores técnico y patrones de velas desde la librería ta-lib y variables construidas a mano como impulsos y retrocesos de Fibonacci. Además, crearemos variables que capturan relaciones o señales potencialmente no lineales con el objetivo de mejorar la calidad de las predicciones. Posteriormente, se lleva a cabo una limpieza de datos y una imputación adecuada para garantizar la consistencia del conjunto. Y, por último, ingeniería de variables para intentar reducir la dimensionalidad de nuestra base de datos.

3. **Construcción de modelos:** Se construyen diferentes modelos con complejidad creciente:
    - Árboles de decision
    - Bagging
    - Random Forest
    - Gradient Boosting
    - XGBoost

4. **Diseño y evaluación de estrategias:** A partir de las predicciones generadas, se diseñan varias estrategias de inversión simuladas, que van desde:
    - **Reglas simples:** entrar al mercado y salir tras un número fijo de velas.
    - **Reglas adaptativas:** entrar en función de la señal del modelo y salir en función de condiciones adicionales (intervalos de confianza de predicción, umbrales de perdida máximo, tener en cuenta el coste por operación para entrar a mercado…).

5. **Métricas de evaluación:** Las estrategias se evalúan con métricas orientadas a la aplicación real, como:
    - Rentabilidad promedio por operación (Directional_error_test)
    - Porcentaje de operaciones rentables
    - Diferencias significativas entre modelos (tests estadísticos)
    - Correlación de los resultados con la volatilidad del activo (volatility_rolling, hablamos

de ella en la página 27).
- Sesgo y varianza de las diferentes métricas con validación cruzada temporal de ventanas fijas.

6. **Pruebas de automatizado:** Estas pruebas de automatización de operaciones se pondrían en marcha una vez encontremos una modelo de inversión rentable y estable. En este caso lo dejamos como futuras líneas de investigación.

2. Algoritmo de extracción de datos

2.1. Fuentes de recogida de información

Como principal fuente de recogida de datos nos decantaremos por Binance. La elección de Binance como fuente es por su posición destacada en el mercado criptográfico actual. Binance es uno de los exchanges de criptomonedas más grandes y populares del mundo, ofreciendo una amplia gama de activos digitales y una sólida infraestructura tecnológica.

Con un gran volumen de transacciones y liquidez en numerosos pares de trading, Binance proporciona datos confiables y actualizados para análisis y modelización en tiempo real. Como plataforma centralizada, Binance opera dentro del marco legal establecido en las jurisdicciones donde tiene presencia, cumpliendo con las regulaciones pertinentes para garantizar la seguridad y transparencia de las operaciones realizadas en su plataforma.

En el contexto de análisis financiero y trading algorítmico, el acceso a datos en tiempo real y a gran escala es fundamental. Una de las herramientas más potentes para este fin son las API's (Application Programming Interfaces), que permiten la conexión directa entre un programa y un proveedor de servicios de datos. A través de peticiones HTTP estructuradas, podemos solicitar datos y recibirlos en formato estructurado (normalmente JSON), que luego puede procesarse automáticamente (Binance, 2024).

Una API es un conjunto de reglas y definiciones que permiten que dos aplicaciones se comuniquen entre sí. En el contexto de los mercados financieros, muchas plataformas de trading (como Binance, Coinbase, Alpha Vantage o Yahoo Finance) exponen APIs que permiten al usuario acceder a:

- Precios históricos y en tiempo real.
- Volumen y profundidad de mercado.
- Ejecución de órdenes (si se usan claves privadas).
- Información de indicadores técnicos, entre otros.

Binance proporciona una API pública para acceder a datos históricos y en tiempo real de distintos mercados (spot, futuros, etc.). Para la extracción de datos se utiliza normalmente la API REST, que permite:

- Obtener precios históricos por velas (OHLCV).
- Acceder al último precio de un activo.
- Consultar el volumen, profundidad y spreads

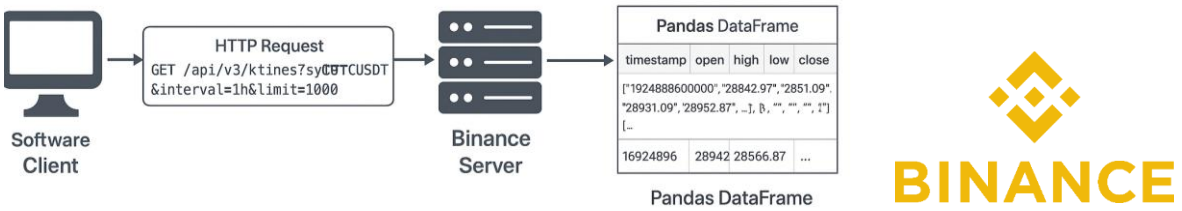


Ilustración 1. Ejemplo Ilustrativo Extracción de datos - Binance

2.2. Diseño base de datos

Para construir la base de datos final, es necesario atravesar varias etapas fundamentales. El proceso comienza con la extracción de datos desde el exchange de Binance, centrados en las cinco variables básicas que definen una vela japonesa en una temporalidad específica, en este caso una hora.

Además, se incorporan variables externas relevantes para el comportamiento del mercado, como el índice de volatilidad VIX y las decisiones sobre tipos de interés.

A partir de esta información bruta, se inicia la creación de variables derivadas mediante distintas técnicas:

- **Ajuste de indicadores técnicos** en diferentes ventanas temporales, generando una nueva variable por cada combinación de indicador y ventana.
- **Recodificación de variables**, que incluye tanto transformaciones como estandarizaciones necesarias para su posterior análisis.
- **Construcción de la variable objetivo**, también mediante procesos de recodificación y análisis direccional del precio.

Una vez recogidas todas las fuentes, se realiza la imputación de valores faltantes y depuración, para luego integrarlas en una base de datos principal unificada. Y por último:

- **Feature engineering**, integrando distintas naturalezas de variables, como patrones de velas, indicadores técnicos y señales implícitas en la evolución de otras variables.

El resultado de todo este proceso es la base de datos final, lista para ser utilizada en modelos predictivos. En los siguientes dos apartados veremos cómo nace la base de datos, con las variables sobre el precio necesarias para la creación de los indicadores técnicos y como ajustamos estas variables en función de una ventana de recogida de datos pasados.

2.2.1. Base de datos bruta

Obtendremos las siguientes variables de la llamada al servidor de Binance a través de la API's, en este caso me he decantado por velas de 1h (una hora):

- **Open:** precio de apertura en la vela
- **High:** máximo de la vela
- **Low:** mínimo de la vela
- **Close:** precio de cierre
- **Volumen:** volumen operado en la vela

Una vela japonesa es una representación gráfica utilizada en análisis técnico para mostrar la evolución del precio de un activo financiero en una temporalidad específica. Cada vela resume cuatro precios clave: el precio de apertura (open), el precio de cierre (close), el precio más alto (high) y el precio más bajo (low) durante ese intervalo, en este caso en intervalos de 1 hora. Visualmente, la vela consta de un "cuerpo", que va desde el precio de apertura hasta el de cierre, y "sombras" o "mechas" que se extienden hasta el máximo y mínimo alcanzado. Si el cierre es mayor que la apertura, la vela suele representarse en color verde o blanco (vela alcista) y si el cierre es menor, en rojo o negro (vela bajista). Este formato facilita la identificación de patrones de comportamiento del mercado, como impulsos, retrocesos o cambios de tendencia.

En segundo lugar, tendremos indicadores técnicos que calcularemos en función de las velas pasadas, son herramientas basadas en datos históricos de precios y volumen de un activo, diseñadas para ayudar a tomar decisiones sobre posibles movimientos futuros del mercado. Debido a que se calculan utilizando datos pasados, los indicadores técnicos hablan sobre el comportamiento anterior del precio.

Esto significa que, aunque pueden ofrecer señales sobre tendencias y patrones de mercado, siempre están un paso detrás del precio actual. Los indicadores se utilizan para confirmar movimientos del mercado, identificar tendencias, y predecir posibles reversiones, pero su naturaleza retrospectiva significa que no pueden predecir el futuro con total seguridad, sino que proporcionan probabilidades basadas en patrones históricos. Para esta base de datos dispondremos de más de 140 indicadores técnicos que podremos utilizar para modelizar, los cuales los encontraremos dentro de la librería de ta-lib (TA-Lib).

Podemos encontrar diferentes tipos de indicadores técnicos en función de la naturaleza de cada uno:

1. **Indicadores de Tendencia:** Estos indicadores ayudan a identificar la dirección general del mercado, si está en una tendencia alcista, bajista o lateral.
    - Media Móvil (MA).
    - Media Móvil Simple (SMA).
    - Media Móvil Exponencial (EMA).
    - Indicador de Convergencia/Divergencia de la Media Móvil (MACD).
    - Parabolic SAR.

2. **Indicadores de Momentum:** Estos indicadores miden la velocidad de los movimientos del precio, ayudando a identificar si una tendencia es fuerte o débil.
    - Índice de Fuerza Relativa (RSI)
    - Estocástico
    - Índice de Movimiento Direccional (DMI)

3. **Indicadores de Volatilidad:** Estos indicadores miden la variabilidad del precio de un activo, ayudando a los traders a comprender el riesgo y la magnitud de los movimientos de precios.
    - Bollinger Bands
    - ATR (Average True Range)

4. **Indicadores de Volumen:** Estos indicadores analizan el volumen de trading, ayudando a confirmar la fortaleza de una tendencia.
    - Volumen en Balance (OBV)
    - Índice de Flujo de Dinero (MFI)

En tercer lugar, los patrones de velas japonesas. A diferencia de una vela japonesa individual, estos patrones son herramientas gráficas que recogen una ventana de velas pasadas, estudiamos sus comportamientos para interpretar el sentimiento del mercado y anticipar posibles movimientos de precio.

Sin embargo, debemos tener en cuenta lo siguiente; evaluar la confiabilidad del patrón en función del contexto del mercado, también debemos combinar los patrones con indicadores técnicos para validar las señales (Realizado posteriormente con las técnicas de Feuture Engineering).

TA-Lib detecta más de 60 patrones. Algunos de los utilizados serán:

1. **Patrones de Reversión Bajista:** Patrones que indican un cambio de una tendencia alcista a una bajista.
    - CDL2CROWS (Two Crows): Dos velas negras consecutivas tras una vela blanca. Señala agotamiento de la tendencia alcista.
    - CDL3BLACKCROWS (Three Black Crows): Tres velas negras consecutivas con mínimos más bajos. Confirma un cambio a una tendencia bajista.
    - CDLDARKCLOUDCOVER (Dark Cloud Cover): Una vela negra cierra dentro del cuerpo de la vela blanca anterior, indicando presión de venta.
    - CDLEVENINGSTAR (Evening Star): Tres velas: una alcista, una doji/pequeña, y una bajista, señalando un posible cambio de tendencia.
    - CDLHANGINGMAN (Hanging Man): Una vela con cuerpo pequeño y sombra inferior

larga, en la cima de una tendencia alcista.

2. **Patrones de Reversión Alcista**: Patrones que indican un cambio de una tendencia bajista a una alcista.
    - CDL3WHITESOLDIERS (Three White Soldiers): Tres velas blancas consecutivas con máximos más altos. Señala fortaleza alcista.
    - CDLHAMMER (Hammer): Vela con un cuerpo pequeño y sombra inferior larga. Aparece tras una tendencia bajista.
    - CDLMORNINGSTAR (Morning Star): Tres velas: una bajista, una pequeña/doji, y una alcista, indicando un posible cambio alcista.
    - CDLPIERCING (Piercing Pattern): Una vela blanca cierra por encima del 50% del cuerpo de una vela negra anterior.

3. **Patrones de Continuación**: Patrones que sugieren la continuación de una tendencia actual.
    - CDLMATHOLD (Mat Hold): Una pausa temporal durante una tendencia alcista, seguida de una vela fuerte.
    - CDLRISEFALL3METHODS (Rising/Falling Three Methods): Una serie de velas pequeñas en dirección opuesta a la tendencia principal, que luego continúa.

4. **Patrones de Indecisión:** Patrones que reflejan la falta de dirección clara en el mercado.
    - CDLDOJI (Doji): Cierre y apertura casi iguales, mostrando equilibrio entre compradores y vendedores.
    - CDLSPINNINGTOP (Spinning Top): Cuerpo pequeño con sombras largas, señal de indecisión.
    - CDLLONGLEGGEDDOJI (Long-Legged Doji): Doji con sombras muy largas, mostrando alta volatilidad e indecisión.

5. **Patrones de Gaps**: Patrones que aparecen con huecos entre velas consecutivas, generalmente indicando fuerza de mercado.
    - CDLTASUKIGAP (Tasuki Gap): Gap en la dirección de la tendencia, seguido de una vela que cierra parcialmente el hueco.
    - CDLGAPSIDESIDEWHITE (Up/Down-Gap Side-by-Side White Lines): Líneas blancas que aparecen después de un gap, mostrando fuerza direccional. 6. Patrones Específicos y Raros Patrones menos comunes que pueden señalar situaciones específicas.
    - CDLABANDONEDBABY (Abandoned Baby): Doji aislado con gaps antes y después, indicando reversión (alcista o bajista).
    - CDLTRISTAR (Tristar Pattern): Tres dojis consecutivos que sugieren un cambio de tendencia.
    - CDLCONCEALBABYSWALL (Concealing Baby Swallow): Cuatro velas negras que indican posible reversión alcista en tendencia bajista.

Otra variable que añadiremos en nuestra base de datos son los retrocesos de Fibonacci. Los retrocesos de Fibonacci son una herramienta clave en el análisis técnico, utilizada para identificar posibles niveles de soporte y resistencia en los movimientos del precio de un activo. Estos niveles se derivan de la famosa sucesión de Fibonacci y las proporciones matemáticas asociadas a ella, como 23.6%, 38.2%, 50%, 61.8% y 78.6%, las cuales reflejan áreas donde el precio podría corregirse antes de continuar su tendencia inicial.

En esencia, los retrocesos de Fibonacci ayudan a comprender cómo los mercados financieros tienden a moverse en patrones cíclicos, marcados por impulsos y correcciones. Su flexibilidad para aplicarse en cualquier temporalidad los convierte en una herramienta indispensable tanto para traders como para analistas cuantitativos.

Por último, nos fijaremos en algunas variables macroeconómicas de interés que nos pueden aportar información acerca del sentimiento del mercado a nivel tradicional (para futuras investigaciones nos gustaría añadir más variables macroeconómicas, como noticias, conferencias

FED o BCE, valor de otros activos financieros, como SP500, el oro, NASDAQ, NVIDIA…):

- Decisión de tipos de interés
- VIX (Índice de Volatilidad SP500)

Para las variables macroeconómicas en función de la naturaleza de las variables tendremos diferentes variantes:

- **Tipos de interés:** Para los tipos de interés recogeremos la decisión de los tipos de interés por parte de la FED para las proyecciones a futuro ya sea a nivel mensual como anual. (['1 Mo', '3 Mo', '6 Mo', '1 Yr', '2 Yr', '3 Yr', '5 Yr', '7 Yr', '10 Yr', '20 Yr', '30 Yr']). Estos datos se actualizan cada cierto tiempo, siendo fechas concretas marcadas por la FED.
- **VIX (Índice de Volatilidad SP500):** Uno de los índices más importantes a nivel internacional sobre las tecnologías, mostrando el sentimiento del mercado tradicional. Pues una forma de medir este sentimiento es con el índice de volatilidad propio del SP500. Obtendremos datos históricos diarios. Tendremos la información de dicho indicador de forma diaria.

2.2.2. Ajuste de los indicadores técnicos y retrocesos de Fibonacci.

Indicadores técnicos

En la modelización de indicadores técnicos, ajustar los parámetros en función de diferentes temporalidades (cortas, medias y largas) es crucial para capturar información relevante en distintos horizontes pasados. A continuación, se describen los tipos de indicadores, los parámetros ajustados y qué ventajas, desventajas e información aportan en cada caso:

1. **Indicadores de Tendencia:** Estos indicadores miden la dirección general del mercado. Ajustar su sensibilidad permite capturar tendencias más rápidas o estables. Vemos sus ventajas y desventajas en función de los parámetros de recogida de datos:
   - **Cortas (5-15 periodos):** Capturan cambios rápidos en la dirección de la tendencia.
     - Ventajas: Detectan movimientos repentinos.
     - Desventajas: Más ruido y señales falsas.
   - **Medias (20-50 periodos):** Proporcionan un equilibrio entre sensibilidad y estabilidad.
     - Ventajas: Buenas para identificar tendencias sostenidas.
     - Desventajas: Pueden reaccionar lentamente a cambios bruscos.
   - **Largas (100-200 periodos):** Filtran el ruido y muestran tendencias de largo plazo.
     - Ventajas: Útiles para detectar tendencias estructurales.
     - Desventajas: Muy lentos para reaccionar a nuevos movimientos.
2. **Indicadores de Momentum:** Estos indicadores miden la velocidad y fuerza de los movimientos del precio, siendo sensibles a cambios repentinos:
   - **Cortas (7-14 periodos):** Detectan sobrecompras y sobreventas en plazos breves.
     - Ventajas: Permiten identificar reversiones rápidas.
     - Desventajas: Pueden generar señales falsas en mercados laterales.
   - **Medias (20-50 periodos):** Suavizan las oscilaciones rápidas del mercado.
     - Ventajas: Menos sensibles al ruido del mercado.
     - Desventajas: Pueden ser menos efectivos en alta volatilidad.
   - **Largas (50-100 periodos):** Muestran la dirección del momentum en tendencias amplias.
     - Ventajas: Más confiables en mercados con movimientos estructurales.
     - Desventajas: Pueden retrasarse en identificar oportunidades de corto plazo.

3. **<u>Indicadores de Volatilidad</u>:** Evalúan la magnitud del movimiento de los precios y su dispersión.

    - **Cortas (10-20 periodos):** Reflejan cambios de volatilidad a corto plazo.
        - Ventajas: Útiles para detectar rupturas rápidas.
        - Desventajas: Altamente sensibles al ruido.

    - **Medias (20-50 periodos):** Capturan niveles moderados de volatilidad.
        - Ventajas: Buenas para mercados equilibrados.
        - Desventajas: menos útiles en mercados muy externos.

    - **Largas (50-100 periodos):** Miden la volatilidad estructural del mercado.
        - Ventajas: Ayudan a identificar fases de alta o baja volatilidad.
        - Desventajas: Pueden ignorar eventos recientes.

4. **<u>Indicadores de Volumen</u>:** Estos validan la dirección del movimiento con base en la actividad del mercado.

    - **Cortas (diarios a 10 periodos):** Detectan actividad inusual en horizontes breves.
        - Ventajas: Capturan aumentos bruscos de interés.
        - Desventajas: Sensibles a picos de volumen poco representativos.

    - **Medias (20-50 periodos):** Suavizan picos de volumen, mostrando tendencias de acumulación.
        - Ventajas: Confirmación más estable de movimientos.
        - Desventajas: Pueden perder detalles intradías.

    - **Largas (50-100 periodos):** Evalúan tendencias estructurales de volumen.
        - Ventajas: Identifican acumulación/distribución prolongada.
        - Desventajas: Poca utilidad en operaciones a corto plazo.

5. **<u>Indicadores de Fuerza</u>**: Miden la intensidad del movimiento en la dirección del precio.

    - **Cortas (5-15 periodos):** Reflejan rápidamente cambios en la fuerza del mercado.
        - Ventajas: Detectan rupturas en consolidaciones.
        - Desventajas: Pueden ser muy volátiles.

    - **Medias (20-50 periodos):** Ofrecen una visión más estable de la fuerza del mercado.
        - Ventajas: Más confiables para tendencias intermedias.
        - Desventajas: Menos sensibles a cambios abruptos.

    - **Largas (50-100 periodos):** Reflejan la fuerza en movimientos estructurales.
        - Ventajas: Confirman tendencias sostenidas.
        - Desventajas: Responden lentamente a nuevos eventos.

Retrocesos de Fibonacci

Como bien explicamos antes estos niveles de posibles soportes y resistencias vienen de la famosa sucesión de Fibonacci y las proporciones matemáticas asociadas a ella, como 23.6%, 38.2%, 50%, 61.8% y 78.6%. Los cuáles serán los puntos de posibles soportes o resistencias que calcularemos para los diferentes valores de los parámetros de recogida de los datos. Tendremos las siguientes temporalidades:

- **Cortas (Intradía o 1-2 días):** Número de velas (observaciones): 24-48 velas. Ideal para analizar movimientos recientes y rápidos dentro del día.
- **Medias (1-2 semanas):** Número de velas: 120-336 velas. Buenas para detectar correcciones o retrocesos dentro de tendencias que se desarrollan durante varios días. Los retrocesos en esta temporalidad suelen alinearse con movimientos estructurales

importantes.

- **Largas (1-3 meses):** Número de velas 720-2160 velas. Adecuado para identificar niveles clave en tendencias estructurales a largo plazo. Además, es útil para operadores de largo plazo que buscan niveles estratégicos de soporte/resistencia que podrían marcar giros importantes en la tendencia general.

2.3. Resumen código algoritmo:

El código de la extracción de la base de datos se encontrará en la ficha técnica junto a la memoria del TFG, aquí se les muestra en una hoja, las funciones principales hechas en Python, donde incluye la función de llamada al servidor de Binance:

(*Código detallado en el ANEXO1)

1. Configuración de la conexión a Binance:

```
exchange = ccxt.binance()
symbol = 'BTC/USDT'  # Activo del que quieres obtener datos
timeframe = '1h'  # Intervalo de velas, puede ser '1m', '5m', '1h', '1d', etc.
limit = 1000  # Límite de velas por solicitud (máximo 1000 en Binance)
max_records = 5000  # Número máximo de registros que deseas obtener
```

2. Algunas funciones principales de recogida y organización de datos:

```
def data_history(symbol, timeframe, since=None, limit=limit, max_records=max_recor
    all_data = []
    while len(all_data) < max_records:
        ohlcv = exchange.fetch_ohlcv(symbol, timeframe, since=since, limit=limit)
        if len(ohlcv) == 0:
            break
        all_data.extend(ohlcv)
        since = since = ohlcv[-1][0] + (ohlcv[1][0] - ohlcv[0][0])  # Incrementar
        # Incrementar para evitar duplicados
        if len(ohlcv) < limit:
            break
        time.sleep(exchange.rateLimit / 1000)  # Respetar el límite de velocidad d
    return all_data[:max_records]
```

3. Obtenemos los datos:

```
timeframe = '1h'
df = rep_data_history(symbol, timeframe)

#Verificar si hay huecos entre fechas
verificar_saltos(df, timeframe)

#Probar el relleno de huecos
df = rellenar_huecos(df, timeframe)

#Comprobar que se han llenado correctamente
verificar_saltos(df, timeframe)
```

4. Funciones de cálculos de las variables de Indicadores Técnicos – Talib
   (*Código de las funciones en la ficha técnica)

```
import warnings
warnings.filterwarnings('ignore')
df = moving_average(df)
df = rsi(df)
df = bollinger_bands(df)
df = parabolic_sar(df)
df = macd(df)
df = stoch(df)
df = adx_dmi(df)
df = add_atr(df)
```

5. Función para identificar los patrones de velas – Talib

```
df = add_candlestick_patterns(df)
```

6. Función para realizar las proyecciones Fibonacci

```
df = add_multiple_fibonacci_retracements(df)
```

7. Función web scraping tipos de interés - US Department of The Treasury

Dado que estos datos no se encuentran disponibles directamente a través de una API de acceso libre, se ha optado por utilizar técnicas de web scraping para su recopilación. En concreto, los datos se extraen de la página oficial del U.S. Department of the Treasury (https://home.treasury.gov), que publica diariamente los tipos de interés correspondientes a los diferentes tramos de vencimiento de los bonos del Tesoro.
El proceso de scraping se estructura en varias etapas:

1. Acceso y obtención del HTML de la página web mediante peticiones HTTP personalizadas con cabeceras dinámicas (User-Agent aleatorio).
2. Identificación de los años disponibles para descargar datos históricos.
3. Localización y extracción del enlace de descarga en formato CSV.
4. Automatización de descargas por año e incorporación de cada tabla a una lista.
5. Unificación de las tablas anuales en un único DataFrame que se ordena cronológicamente por fecha.

Este proceso permite contar con una base robusta y actualizada de los tipos de interés diarios de Estados Unidos, que serán utilizados como variables explicativas en los modelos de predicción. A continuación, se muestra una representación visual simplificada del flujo de trabajo seguido para la extracción de estos datos:

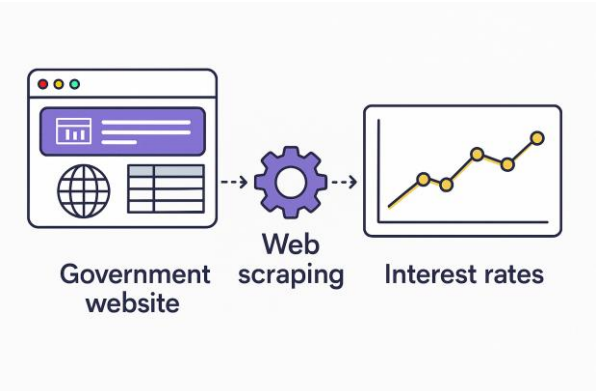


Ilustración 2. Ejemplo Ilustrativo Extracción Tipos de Interés Treasury US

```
df_tipos = get_interest_rate()
```

8. Función para añadir datos históricos VIX(Sp500):

```
vix_data_sorted = load_and_sort_vix_data()
```

```
vix_data_sorted.head()
```

	Date	OPEN_VIX	HIGH_VIX	LOW_VIX	CLOSE_VIX
6802	2017-01-03	14.07	14.07	12.85	12.85
6803	2017-01-04	12.78	12.80	11.63	11.85
6804	2017-01-05	11.96	12.09	11.40	11.67
6805	2017-01-06	11.70	11.74	10.98	11.32
6806	2017-01-09	11.71	12.08	11.46	11.56

3. Estudio preliminar y depuración de la información.

La limpieza e imputación de datos es una fase fundamental dentro del flujo de análisis cuantitativo y modelización.

Primero haremos un estudio de depuración de datos de las bases de datos, tenemos la primera correspondiente a las variables que nacen de la base de datos pura de origen de extracción de datos de la plataforma de Binance; la segunda los tipos de interés del Departamento del Tesoro de los Estados Unidos (US Department of the Treasury); y la tercera de la extracción de la variable macroeconómica del VIX.

Una vez tenemos limpios los DataFrame los uniremos, sacaremos algunos gráficos de algunas de las variables de nuestra base de datos y por último recodificaremos algunas variables.

3.1. Estudios y depuración de datos.

3.1.1. Datos perdidos de la API de Binance

Durante la recolección de datos de mercado (OHLCV) mediante la API de Binance, se han identificado ciertas ventanas horarias con datos ausentes. Esto ocurre de forma esporádica debido a caídas en los servidores o interrupciones temporales en el servicio de la API. Además de haber saltos temporales.

Para solventar este problema, se ha implementado un algoritmo en Python que primero verifica los saltos de tiempo y luego rellena los huecos con el último dato disponible (forward-fill). Este método asegura la continuidad temporal de las observaciones, sin introducir nuevos valores estimados artificialmente.
(*Código de todas las funciones detallas, depuración, imputación, recodificación y organización en la ficha técnica – ANEXO1)

3.1.2. Datos perdidos Tipos de Interés del Tesoro de EE.UU

Los datos de tipos de interés se han obtenido desde la fuente oficial del Departamento del Tesoro de los Estados Unidos (US Department of the Treasury). Durante la integración de estos datos, se ha observado que algunas variables presentan una alta tasa de valores faltantes, en especial:

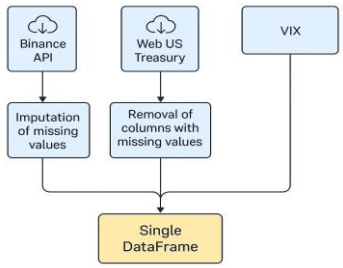
- Tipo de interés a 2 meses → 31.36% de valores perdidos.
- Tipo de interés a 4 meses → 76.37% de valores perdidos.

Dado el alto grado de incompletitud y que tenemos otras variables sin missing o con pocos missing con gran multicolinealidad con ellas, hemos decidido eliminarlas de la base de datos.

3.2. Integración de Todos los DataFrames

La integración de los datos de mercado (Binance), tipos de interés (Treasury) y VIX se

realiza de forma automática mediante un algoritmo que unifica todas las tablas en un único DataFrame final, alineado por timestamp y consistente a nivel temporal.



timestamp	open	high	low	close	volume	timestamp2	SMA_5	EMA_5
2017-08-17 04:00:00	4261.48	4313.62	4261.32	4308.83	47.181009	17395.166667	NaN	NaN
2017-08-17 05:00:00	4308.83	4328.69	4291.37	4315.32	23.234916	17395.208333	NaN	NaN
2017-08-17 06:00:00	4330.29	4345.45	4309.37	4324.35	7.229691	17395.250000	NaN	NaN
2017-08-17 07:00:00	4316.62	4349.99	4287.41	4349.99	4.443249	17395.291667	NaN	NaN
2017-08-17 08:00:00	4333.32	4377.85	4333.32	4360.69	0.972807	17395.333333	4331.836	4331.836

5 rows × 313 columns

Ilustración 3. Ejemplo Ilustrativo Unión Bases de Datos

3.3. Base de datos primera fase

Después de esta primera fase de la construcción de nuestra base de datos acabamos con datos que van desde el primer dato histórico que nos ofrece Binance, timestamp = 2017-08-17 04:00:00, hasta timestamp = 2024-12-09 11:00:00, siendo un total de 64111 observaciones.

El conjunto de datos empleado en este trabajo consta de un total de 314 variables. A continuación, se presenta una clasificación de dichas variables según su naturaleza:

1. Variables de precio y volumen (6 variables)

timestamp, open, high, low, close, volume: representan la información OHLCV (Open, High, Low, Close, Volume) junto con el timestamp original y transformado.

2. Medias móviles (28 variables)

Simples (SMA), Exponenciales (EMA), Ponderadas (WMA) y Suavizadas (SMMA) para los periodos 5, 7, 9, 14, 24, 48 y 72.
Total: 4 tipos × 7 periodos = 28 variables.

3. Índice de Fuerza Relativa (RSI) (30 variables)

Calculado sobre el precio de cierre y el precio típico, para periodos 5, 9, 14, 20 y 30.
Para cada combinación se incluyen 3 variables: el valor del RSI, una señal de sobrecompra (Overbought) y una señal de sobreventa (Oversold). El valor de sobreventa se añade cuando el RSI toma valores por debajo de 30 y sobrecompra cundo está por encima de 70.
Total: 2 tipos de precio × 5 periodos × 3 señales = 30 variables.

4. Bandas de Bollinger (15 variables)

Límites superior, medio e inferior para periodos 5, 10, 14, 20 y 30.
Total: 5 periodos × 3 bandas = 15 variables.

5. Parabolic SAR (9 variables)

Valores del SAR, señal de si está por debajo del precio, y cambio de tendencia para 3 configuraciones distintas (lenta, estándar y rápida).
Total: 3 configuraciones × 3 señales = 9 variables.

6. MACD (5 variables)

Línea MACD, señal, histograma y dos señales adicionales (MACD_above_signal, MACD_cross_signal). La primera indica si el MACD está por encima o por debajo de la línea de señal. Mientras que la segunda es si el MACD cruza la línea de la señal.

7. Estocástico (STOCH) (54 variables)

Se incluyen los valores %K y %D, señales de sobrecompra y sobreventa, y cruces alcistas y bajistas para múltiples combinaciones de parámetros:
Combinaciones: (5,3), (5,5), (5,9), (9,3), (9,5), (9,9), (14,3), (14,5), (14,9).
Total: 9 combinaciones × 6 variables por combinación = 54 variables.

8. Indicador Direccional (DMI) y ADX (27 variables)

Calculado para periodos 7, 14 y 21, incluyendo dmi_plus, dmi_minus, adx. Tendremos cruces alcistas/bajistas**,** Fuerza de la señal, tendencia fuerte y ausencia de tendencia, confirmación de tendencia

Total: 3 periodos × 9 variables = 27 variables.

9. ATR (Average True Range) (5 variables)

Periodos: 5, 14, 20, 30 y 50.

10. OBV (On Balance Volume) (3 variables)

Valor bruto (OBV), escalado (OBV_scaled) y divergencias (OBV_Divergence).

11. Patrones de Velas Japonesas (64 variables)

Detectados mediante funciones de la librería TA-Lib, incluye patrones de reversión, continuación y consolidación como CDLENGULFING, CDLDOJI, CDLMORNINGSTAR, CDLSHOOTINGSTAR, entre otros.

Total: 64 patrones individuales.

12. Retrocesos de Fibonacci (60 variables)

Calculados sobre ventanas de 24, 48, 120, 336, 720 y 1440 velas.

Para cada ventana se incluyen: el máximo y mínimo local, niveles del 0%, 23.6%, 38.2%, 50%, 61.8%, 78.6% y 100%.

Total: 6 ventanas × 10 variables por ventana = 60 variables.

13. Tipos de Interés del Tesoro de EE. UU. (12 variables)

Bonos a vencimientos de: 1 mes, 3 meses, 6 meses, 1, 2, 3, 5, 7, 10, 20 y 30 años.

14. Índice de Volatilidad (VIX) (4 variables)

Precio de apertura, máximo, mínimo y cierre del VIX: OPEN_VIX, HIGH_VIX, LOW_VIX, CLOSE_VIX.

Esta estructura de datos permite una modelización robusta, integrando múltiples dimensiones del comportamiento del precio, la tendencia, la volatilidad y la psicología del mercado. La riqueza de variables ofrece un entorno óptimo para el desarrollo y la evaluación de modelos predictivos en finanzas cuantitativas.

3.4. Algunos gráficos ilustrativos de las variables

1. Gráfico de la evolución del activo históricamente mediante velas japonesas (Open, High, Low, Close)

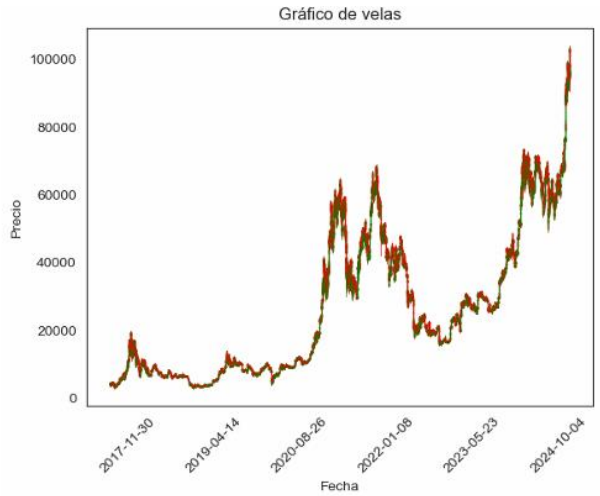


Ilustración 4. Evolución Precio Bitcoin 2017-2024

2. Indicadores técnicos Medias Móviles. Ej.: periodo = 14

Ilustración 5. Ejemplo Indicador Técnico Medias Móviles - periodo =14

3. Indicadores técnicos. Ejemplos: RSI, STOCH_SLOWK, STOCH_SLOWD, ADX, DMI_PLUS, DMI_MINIUS

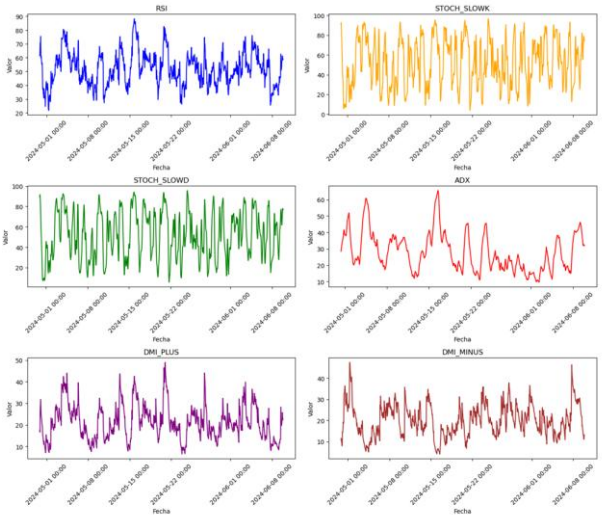


Ilustración 6. Ejemplo Indicadores técnicos

4. Patrón de reversión bajista: CDLEVENINGSTAR (Evening Star), patrón de reversión alcista: CDLHAMMER (Hammer), patrón de continuación: CDLRISEFALL3METHODS (Rising/Falling Three Methods), patrón de indecisión en el mercado: CDLDOJI (Doji) y patrón de fuerza/Gaps: CDLTASUKIGAP (Tasuki Gap)

Ilustración 7. Ejemplos Patrones Velas

5.Retrocesos de Fibonacci y Volumen: Parámetro 336 y VIX(Sp500).

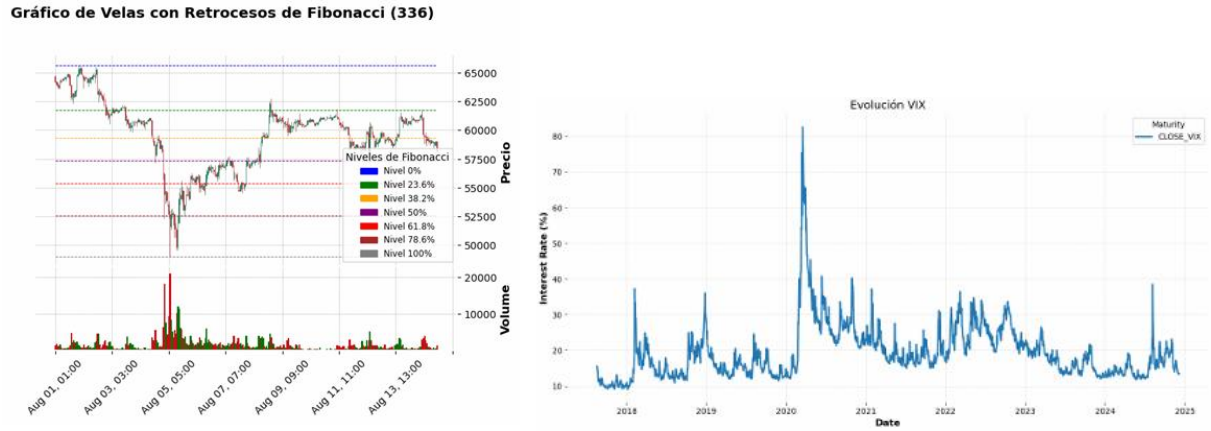


Ilustración 8. Ejemplo Retroceso de Fibonacci y Evolución VIX

6. Evolución de los tipos de interés.

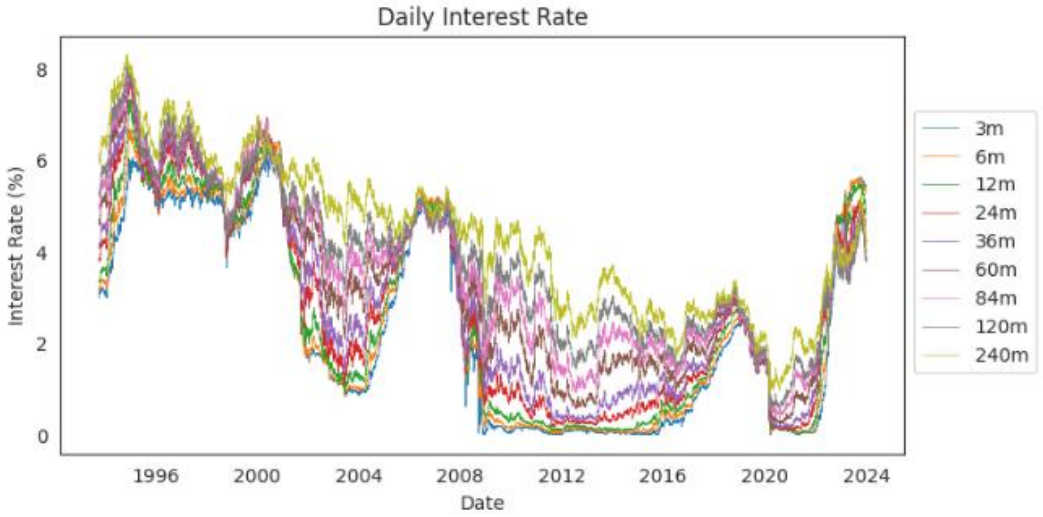


Ilustración 9. Evolución tipos de interés US

(Resto de gráficos junto con análisis en el anexo 2).

3.5. Recodificaciones variables

Para adaptar nuestra base de datos al contexto de operación en tiempo real y facilitar el aprendizaje de patrones por parte de los modelos predictivos, se realiza un proceso de recodificación de las variables. Este proceso se estructura en tres etapas principales:

3.5.1. Eliminación de variables no disponibles

Se eliminan todas aquellas variables que contienen información futura que no estará disponible en el instante de toma de decisiones, es decir, a tiempo real. Específicamente, se descartan las columnas asociadas al precio de cierre, máximos y mínimos del periodo actual (close, high, low), así como otras variables auxiliares que no aportan valor al modelo en producción, como index_x, index_y, timestamp2, y algunos precios del VIX (HIGH_VIX, LOW_VIX, CLOSE_VIX).

3.5.2. Desplazamiento de la información una observación hacia el futuro

Con el objetivo de simular un entorno operativo real, toda la información técnica disponible (salvo el precio de apertura open, el timestamp y OPEN_VIX) es desplazada una fila hacia adelante. Esto asegura que en un instante t, las variables predictoras contienen únicamente la información que estaría disponible en el instante t−1, es decir, justo antes de tomar la decisión.

3.5.3. Recodificación de variables técnicas en términos relativos al precio de apertura

Para facilitar el aprendizaje de patrones generalizables (y no memorizar valores absolutos de precios), las variables técnicas que representan niveles de precios son transformadas en porcentajes relativos al precio de apertura (open). Esta transformación se aplica sobre diferentes grupos de indicadores:

- **Medias móviles**: Las variables SMA, EMA, SMMA y WMA para distintos periodos se recodifican como el porcentaje de desviación respecto al precio de apertura.
- **Bandas de Bollinger**: Se calcula la distancia relativa del precio de apertura a los tres niveles característicos de las bandas (superior, media, e inferior), para diferentes ventanas temporales.
- **Parabolic SAR**: Se transforma la distancia entre el open y los distintos niveles del SAR en términos porcentuales, lo que permite capturar señales de cambio de tendencia de forma estandarizada.
- **Retrocesos de Fibonacci**: Tanto los niveles extremos (fib_high, fib_low) como los niveles intermedios de retroceso (0%, 23.6%, 38.2%, etc.) son recodificados en términos porcentuales respecto al open. Esto permite analizar la posición del precio relativo a niveles clave del análisis técnico a lo largo de diferentes escalas temporales (24, 48, 120, 336, 720 y 1440 observaciones previas).

3.5.4. Generación de la Variable Objetivo (target_pct)

Con el objetivo de modelar la evolución futura del precio en un contexto de trading algorítmico, se crea una variable objetivo (target) que representa la variación porcentual del precio de apertura 6 horas en el futuro, con respecto al precio de apertura actual. Esta variable se define como:

$$target_{pct} = \frac{open_{t+6} - open_t}{open_t}$$

Esta definición permite a los modelos anticipar tanto la dirección como la magnitud del movimiento del activo a corto plazo, en un marco de decisión que simula condiciones reales de mercado.

La elección de un horizonte de 6 horas se adopta como un punto intermedio entre una predicción demasiado inmediata (ruidosa e inestable) y una demasiado lejana (más incierta y dependiente de factores externos). Este horizonte se considera razonable como prueba inicial, permitiendo evaluar el comportamiento del modelo sin necesidad de esperar una ventana excesivamente prolongada.

Cabe destacar que esta estructura es completamente flexible: el algoritmo de procesamiento está diseñado para generar datos a distintas temporalidades y construir modelos con diferentes horizontes de predicción, por lo que en etapas posteriores se podrán realizar comparaciones con ventanas más cortas (por ejemplo, 1 o 3 horas) o más largas (por ejemplo, 12 o 24 horas), en función del objetivo estratégico del modelo y del tipo de operación que se desee simular (scalping, swing trading, etc.)

3.5.5. Eliminación de Observaciones Iniciales

Como parte del preprocesamiento final, se eliminan las observaciones correspondientes a los dos primeros meses de datos históricos. Esta decisión responde al hecho de que varios de los indicadores calculados, especialmente aquellos que requieren ventanas temporales extensas como las proyecciones de retrocesos de Fibonacci, no pueden computarse de forma fiable hasta haber acumulado al menos 1440 observaciones (equivalente a 60 días si trabajamos con datos horarios). Mantener esas observaciones iniciales generaría valores nulos en muchas variables, lo cual afectaría negativamente el entrenamiento del modelo.

Tras la recodificación de la base de datos pasamos de 314 variables a 295 y de 64111 observaciones a 62565. Para acabar con el procedimiento de la construcción de la base de datos pasamos a la última parte de ingeniería de variables.

4. Feuture Engineering

En este bloque, nos centraremos en la creación de variables, combinando indicadores técnicos y patrones de velas con diferentes configuraciones de velocidad y ventanas temporales. El objetivo es generar alertas de sobrecompra, sobreventa, cambios de tendencia y continuaciones, integrando múltiples señales para obtener una representación más fiable del comportamiento del mercado. A través de estas transformaciones, intentaremos mejorar la calidad de las variables utilizadas en los modelos y, en consecuencia, optimizar la predicción de los movimientos del precio de Bitcoin, reduciendo el número de variables de nuestra base de datos.

Además, reduciremos la dimensionalidad de nuestra base de datos ya que resumiremos todas las señales de indicadores técnicos y patrones de velas en estas variables nuevas.
(*Códigos de construcción de estas variables en ANEXO1)

4.1. Construcción de la variable Sobrecompra y Sobreventa

Para intentar mejorar una posible detección de condiciones de mercado en sobrecompra/sobreventa, desarrollaremos una variable compuesta denominada ***sobrecompra*** y ***sobreventas***. Esta variable integrar múltiples señales provenientes de indicadores técnicos y patrones de velas, combinando distintos enfoques para capturar con mayor precisión los momentos en los que el precio podría estar en niveles extremos.
En su construcción, utilizaremos los siguientes elementos:

- **Índice de Fuerza Relativa (RSI):** Se considerar el RSI basado en el precio de cierre y en el RSI_typicalprice, utilizando ventanas de 5, 9, 14, 20 y 30 periodos para evaluar diferentes horizontes temporales de sobrecompra y sobreventa.
- **Estocástico:** Se incorporarán señales de sobreventa del Estocástico, con parámetros de K = 5, 9 y 14 periodos, con el objetivo de capturar condiciones en las que el retroceso del precio comienza a debilitarse. En caso contrario, señales de sobrecompra del Estocástico con el objetivo de detectar debilidad en el impulso del precio.
- **Patrones de velas de sobrecompra/sobreventa:** Se incluirán patrones característicos de agotamiento alcista, como Tres Soldados Blancos, Marubozu, Kicking y otros previamente identificados, que históricamente han mostrado señales de reversión bajista. Y otras que han mostrado una reversión alcista con señales de sobreventa como son los patrones Hammer, Hammer invertidos, Morning Dojistar…

Para adaptar esta variable a distintos escenarios de mercado, se implementarán tres niveles de sensibilidad, definidos en función de los valores de K en los indicadores técnicos:

- **Pequeño (rápido):** Se utilizarán ventanas de K = 5 y 9 periodos, lo que permitir capturar señales más rápidas y frecuentes, adecuadas para movimientos de corto plazo.
- **Mediano (intermedio):** Se consideraron valores de K = 14 periodos, proporcionando un equilibrio entre sensibilidad y robustez.
- **Grande (lento):** Se emplearán ventanas de K = 20 y 30 periodos, lo que filtrar señales más volátiles y detectar técnicamente situaciones de sobrecompra sostenida.

La combinación de estos indicadores y patrones en una técnica variable permitir mejorar la identificación de condiciones de sobrecompra o sobreventa, facilitando su integración en modelos de predicción y optimizando la detección de posibles cambios de tendencia en Bitcoin.

4.2. Construcción de la variable Change_trend_bull y Change_trend_bear

Uno de los aspectos clave en el análisis de los movimientos del mercado es la identificación

de cambios de tendencia, tanto al alza como a la baja. Para mejorar la capacidad predictiva de nuestro modelo, generaremos dos variables: ***change_trend_bull*** para detectar transiciones hacia una tendencia alcista y ***change_trend_bear*** para identificar cambios a una tendencia bajista.

Estas variables estarán construidas a partir de múltiples señales como el Parabolic SAR, que señala cambios de tendencia cuando sus valores cruzan el precio; el cruce alcista o bajista del Estocástico, que identifica momentos en los que el precio gana o pierde momentum; los cruces en el MACD, que marcan transiciones de impulso cuando la línea MACD cruza su señal; y el cruce en el DMI, que mide la fuerza direccional de la tendencia a partir del +DI y -DI. Además, incorporaremos patrones de velas japonesas asociados a cambios de tendencia.

4.3. Construcción de la variable trend_cont_bull y trend_cont_bear

Una vez identificado un cambio de tendencia en el precio de Bitcoin, es fundamental evaluar si dicha tendencia tiene probabilidades de continuar o si, por el contrario, puede debilitarse. Para ello, construiremos variables que capturen la fortaleza y confirmación de la tendencia utilizando múltiples indicadores técnicos. Definiremos dos variables principales:

trend_cont_bull: Indica la continuación de una tendencia alcista.
trend_cont_bear: Indica la continuación de una tendencia bajista.

Para detectar estas continuaciones, combinaremos diversos factores clave:

- **MACD Above Signal:** En una tendencia alcista, si el MACD se mantiene por encima de su línea de señal, esto sugiere que la presión compradora sigue dominando. De manera análoga, en una tendencia bajista, si el MACD se mantiene por debajo de su señal, la presión vendedora sigue siendo fuerte.
- **ADX y Fortaleza de Tendencia**: Utilizaremos el ADX para determinar si la tendencia es lo suficientemente fuerte como para continuar. Un ADX alto indica una tendencia salida, mientras que un ADX bajo sugiere posible debilitamiento.
- **Confirmación de Tendencia con ADX:** Además de medir la fortaleza de la tendencia, verificaremos si el ADX respalda la dirección predominante.

Patrones de Velas: Incorporaremos patrones de velas que indiquen confirmación de continuación alcista o bajista, asegurando que la variable integre señales adicionales del mercado.

4.4. Base de datos final tras la creación de las nuevas variables

Como paso final del proceso de ingeniería de variables (feature engineering), se llevó a cabo una depuración y síntesis de las señales derivadas de indicadores técnicos y patrones de velas japonesas. Inicialmente, la base de datos contenía 295 variables, de las cuales 122 correspondían a señales binarias relacionadas con condiciones específicas de indicadores técnicos (por ejemplo, sobrecompra, sobreventa, cruces de medias móviles, divergencias) y patrones de velas (como envolventes, martillos, estrellas, etc.).

Sustituidas estas por un conjunto de 26 variables categóricas sintetizadas, diseñadas para capturar la información clave de manera más eficiente. Estas nuevas variables resumen las señales más relevantes bajo cuatro grandes categorías: sobrecompra, sobreventa, cambios de tendencia y continuación de tendencia.

Esta consolidación permite mantener la información esencial para la modelización, reduciendo el riesgo de sobreajuste y mejorando la interpretabilidad del modelo. Como resultado de esta depuración, la base de datos final quedó compuesta por un total de 199 variables, optimizadas para su uso en modelos predictivos. Aun así, sigue siendo una base de datos con una alta dimensionalidad, realizaremos un análisis factorial para ver si conseguimos reducir la dimensionalidad.

5. Reducir dimensionalidad con análisis factorial

En este apartado vamos a intentar solucionar el problema de la alta dimensionalidad de nuestra base datos. Para ello vamos a realizar un análisis Factorial Mixto. El análisis será mixto ya que aparte de tener variables numéricas también tenemos variables categóricas y nos gustaría que se tuviera en cuenta su naturaleza a la hora de realizar el análisis factorial.

El criterio utilizado es el porcentaje de variabilidad acumulado. Esto nos interesa que sea lo más elevado posible ya que el mínimo de información pérdida nos puede llevar a fallos en la predicción y por tanto pérdidas económicas. Para este caso nos quedaremos con un 98% de la variabilidad.

Una vez seleccionado los factores, veremos en aquellos que acumulan una mayor variabilidad de la base de datos y por tanto explicabilidad, veremos que variables explican a cada dimensión.

5.1. Análisis Factorial Mixto (FAMD)

```
data_factorial <- data_recodificado |>
  select(-c('open', 'target_pct'))
res.famd <- FAMD(data_factorial, ncp= ncol(data_factorial), graph = FALSE)
```

Sacamos los autovalores asociados a cada uno de los factores, junto con la varianza explicada por cada uno y la acumulada:

Dimensión	eigenvalue	variance.percent %	cumulative.variance.percent %
Dim.1	74,077	25,369	25,37
Dim.2	19,542	6,692	32,06
Dim.3	18,530	6,346	38,41
Dim.4	11,826	4,050	42,46
Dim.5	11,416	3,910	46,37
Dim.6	8,422	2,884	49,25
Dim.7	7,678	2,630	51,88
Dim.8	4,672	1,600	53,48
Dim.9	4,409	1,510	54,99
Dim.10	4,025	1,378	56,37
…	…	…	…
Dim.125	0,306	0,105	97,87
Dim.126	0,283	0,097	97,97
Dim.127	0,281	0,096	98,06
Dim.128	0,270	0,093	98,15
Dim.129	0,262	0,090	98,24
Dim.130	0,245	0,084	98,33

Ilustración 10. Resultados autovalores Análisis Factorial Mixto

Podemos ver que podemos reducir la dimensión de nuestra base de datos a casi la mitad que vamos a tener explicado a casi el 98% de la variabilidad de nuestros datos. Por lo tanto, volvemos a realizar el análisis factorial quedándonos con 128 factores.

Los porcentajes de inercia explicados por cada dimensión FAMD:

```
fviz_eig(res.famd, ncp = 40)
```

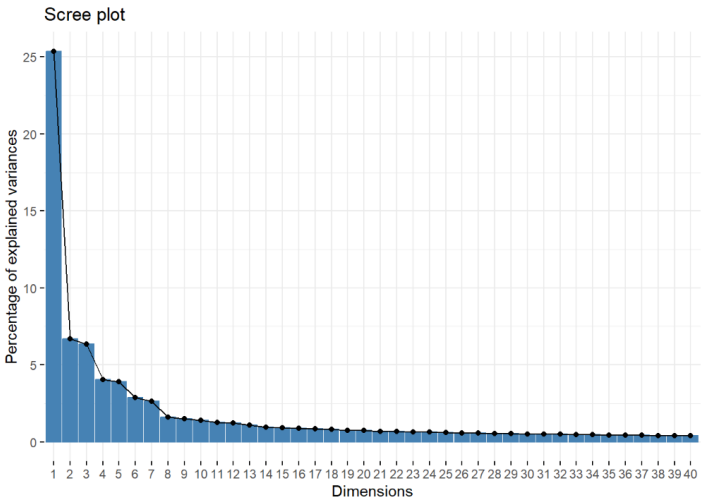


Ilustración 11. Porcentaje de inercia explicado por cada dimensión

Representamos las variables de las primeras dimensiones:

Ilustración 12. Representaciones variables en las 4 primeras dimensiones

Una de las características del análisis factorial es que podemos interpretar las dimensiones, aquí mostraremos de ejemplo las 4 primeras.

Vemos como para el primer factor las variables que representan son las medias móviles y las derivaciones de estas como pueden ser el RSI un índice de fuerza relativa en función de los diferentes tipos de medias móviles poder detectar cambios de tendencias, o fuerza de la tendencia o movimiento del precio.

Para la segunda dimensión podemos ver como son variables de caracter proyección. Estas variables según la regla de retrocesos de Fibonacci alcistas. Estas variables son una proyección en función del mínimo y máximo que se recogen de datos pasados (longitud de recogida de datos recogida en los parámetros).

Para la tercera dimensión podemos ver como son variables de caracter proyección. Estas variables según la regla de retrocesos de Fibonacci alcistas. Estas variables son una proyección en función del mínimo y máximo que se recogen de datos pasados (longitud de recogida de datos recogida en los parámetros).

Vemos como para la cuarta dimensión es sobre las diferentes variables sobre los tipos de interés.

5.2. Conclusiones análisis factorial

A lo largo de este trabajo, se consideró la posibilidad de reducir la dimensión del conjunto de datos mediante técnicas como el análisis factorial, con el objetivo de eliminar posibles redundancias entre las variables y simplificar la estructura predictiva. Sin embargo, tras llevar a cabo diversas pruebas comparativas con distintos modelos de aprendizaje automático, incluidos árboles de regresión, Bagging, Random Forest, Gradient Boosting y XGBoost, se concluyó que la base de datos completa, con todas las variables originales, ofrecía mejores resultados predictivos en términos de rentabilidad y precisión.

Aunque se probaron varias versiones del conjunto de datos con factores extraídos, la mayoría de los modelos obtenían un rendimiento inferior al entrenarse sobre las variables reducidas en comparación con el conjunto de datos original. Esta observación refuerza la hipótesis de que, en este contexto específico, la riqueza y diversidad de las variables originales proporciona al modelo más información útil para captar patrones complejos asociados a los impulsos y retrocesos del mercado.

Dado que los resultados de estas pruebas son numerosos y su análisis detallado podría desviar la atención de otros aspectos centrales del trabajo, se ha decidido no incluirlos en el cuerpo principal del TFG. No obstante, se destaca que esta elección metodológica se basa en una evaluación empírica sólida y en la búsqueda del mejor rendimiento predictivo posible dentro de las limitaciones del proyecto.

6. Análisis de la serie temporal financiera – Estacionariedad

6.1 Contextualización del problema

Antes de proceder con el análisis gráfico y cuantitativo de la serie temporal, es importante considerar un supuesto teórico que puede afectar de forma significativa la calidad de los modelos predictivos: la presencia de heterocedasticidad en la variable objetivo.

En el contexto del mercado de activos financieros y en particular, en mercados emergentes como el de las criptomonedas es habitual que la volatilidad de los retornos no se mantenga constante en el tiempo. A medida que el mercado madura, el volumen de operaciones crece y se incorporan nuevos participantes, se observa una evolución estructural en la dinámica del precio, lo que da lugar a periodos de alta y baja variabilidad en los retornos.

Además, existen eventos específicos en estos activos que alteran de forma abrupta su comportamiento histórico, como el fenómeno del halving, que reduce a la mitad la recompensa por bloque minado, generando cambios en la oferta y, en consecuencia, en la volatilidad del precio (BBVA, 2024).

Desde el punto de vista estadístico, estos cambios generan una violación del supuesto de homocedasticidad, fundamental en muchos modelos de regresión. En este caso, la varianza de los errores no se mantiene constante a lo largo del tiempo, lo que puede provocar que los modelos sobreajuste ciertas regiones de la serie mientras subestiman otras, reduciendo su capacidad de generalización.

Por ello, resulta pertinente evaluar si esta heterocedasticidad está presente en los datos y, en caso afirmativo, plantear estrategias que mitiguen su impacto en el proceso de modelado.

Para empezar, representamos gráficamente los datos de entrenamiento y los de prueba, tanto la variable precio OPEN como las variables target que se crean a partir de esta:

```
ggplot(data_train3, aes(x = timestamp, y = open)) +
  geom_line() +
  geom_smooth(method = "lm", se = FALSE) +
  labs(title = "Evolución Bitcoin 2017-2021 (Data Train)") +
  theme_minimal()
```

```
ggplot(data_test3, aes(x = timestamp, y = open)) +
  geom_line() +
  geom_smooth(method = "lm", se = FALSE) +
  labs(title = "Evolución Bitcoin 2021-Actualidad (Data Test)") +
  theme_minimal()
```

```
`geom_smooth()` using formula = 'y ~ x'
```

```
`geom_smooth()` using formula = 'y ~ x'
```

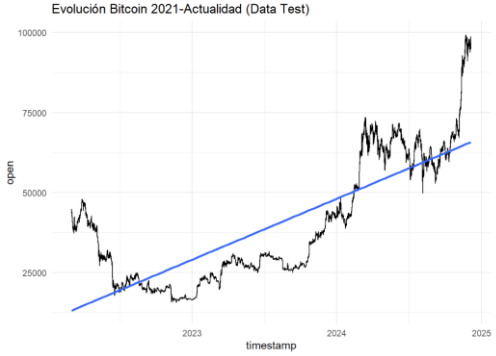


Ilustración 13. Evolución precio Bitcoin Train y Test

Vamos a ver las distribución de la variable target para los datos de entrenamiento y los de prueba para ver si es homogénea, es decir, que con los datos de entrenamiento puede aprender bien del pasado de patrones en función de las distintas variables para poder predecir en el futuro.

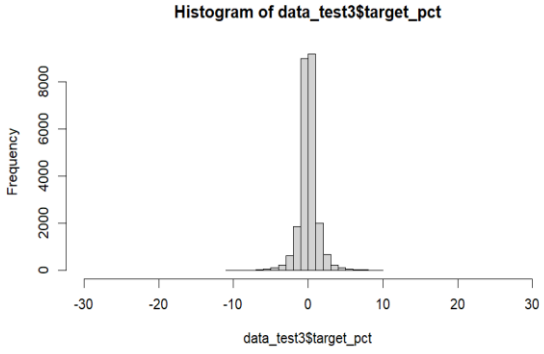
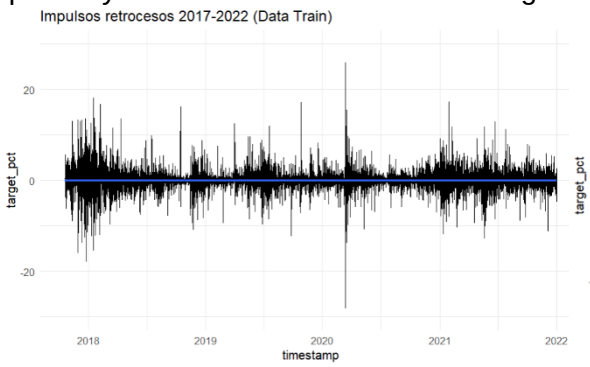


Ilustración 14. Histograma variable objetivo, data_train y data_test

Podemos ver como en los datos de entrenamiento podemos encontrarnos con periodos con mucha mayor volatilidad lo que nos crean observaciones con muchísima mayor amplitud en cuanto a impulsos y retrocesos. Lo vemos en los siguientes gráficos:



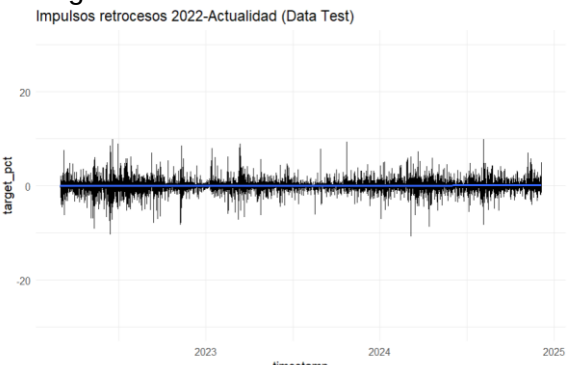


Ilustración 15. Impulsos y retrocesos, data_train y data_test

Aquí se puede ver de mejor manera está diferencia de volatilidad, entre ambos periodos que tiene como consecuencia los eventos de halving en los cuales la volatilidad del precio disminuye en los años venideros después de la ocurrencia de dicho evento. Además, cabe añadir un activo tiende a tener muchísima mayor volatilidad cuando el volumen del mercado es muchísimo menor ya que compras grandes por ballenas pueden llevar a subidas y bajadas buscar del precio de un activo.

De forma más concreta el halving de Bitcoin es un evento que puede generar un impacto significativo en la volatilidad del precio, principalmente debido a la reducción en la emisión de nuevos BTC y el efecto especulativo que este fenómeno conlleva.

Cada vez que ocurre un halving, la recompensa que reciben los mineros por añadir un bloque a la blockchain se reduce a la mitad. Esto implica una menor cantidad de nuevos BTC ingresando al mercado, lo que puede afectar la oferta disponible. Si la demanda de Bitcoin se mantiene constante o aumenta, esta reducción en la oferta puede generar presiones alcistas sobre el precio, impulsando su valorización.

Sin embargo, la volatilidad no solo se explica por la oferta y la demanda, sino también por la especulación que rodea a cada halving. Históricamente, estos eventos han generado expectativas alcistas, atrayendo a inversionistas y traders que buscan beneficiarse de una posible apreciación en el precio. Como resultado, se suelen observar incrementos pronunciados antes del halving, seguidos de correcciones después del evento, lo que contribuye a una alta volatilidad en el corto plazo (Cointelegraph, 2024).

Otro factor clave es el impacto en los mineros. La reducción de recompensas puede hacer que algunos mineros menos eficientes se retiren de la red si el precio de Bitcoin no sube lo suficiente para compensar sus costos operativos. Esto puede provocar una disminución en el hashrate (poder de cómputo de la red), generando incertidumbre en el mercado y aumentando la inestabilidad en el precio.

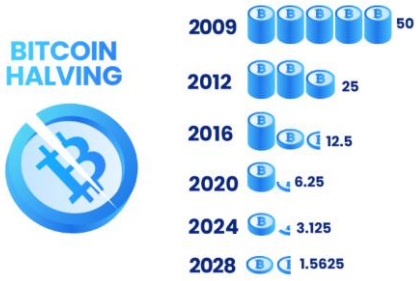


Ilustración 16. Ilustración Halving Bitcoin

```
halving_dates <- as.POSIXct(c( "2020-05-11 00:00:00",
                               "2024-04-20 00:00:00"),
                            format = "%Y-%m-%d %H:%M:%S", tz = "UTC")
```

Ilustración 17. Halving's Bitcoin - Evolución Precio

De forma gráfica podemos ver con claridad como existe una volatilidad diferenciada entre diferentes periodos de halving y por ende cada año, vemos esta volatilidad con gráficos de cajas:

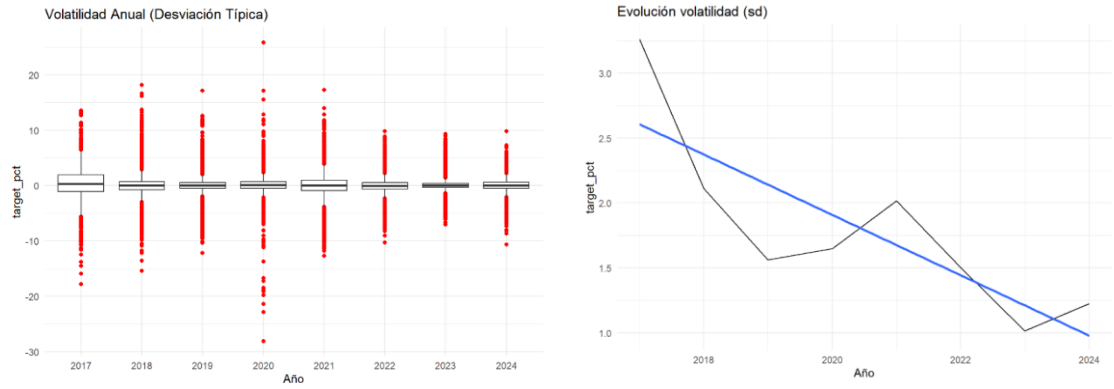


Ilustración 18. Evolución volatilidad. Diagrama de cajas y recta de regresión

En estos gráficos vemos como a lo largo de los años la volatilidad del activo a ido dismuyendo con la pasada de los años, menos en el año 2020 al ser un año más atípicos por eventos de cisnes negros o un periodo bastantes hawkish por parte de los banco centrales con continuadas subidas de tipos de interés creando bastante pánico en el mercado tanto tradicional como en el de las criptomonedas.

Podríamos buscar una forma de ajustar la volatilidad de nuestras observaciones de entrenamientos ya que estas se están ajustando con volatilidades muy distintas a las que se vayan a encontrar ahora mismo en la actualidad.

6.2. Solución para el problema de volatilidad

Con el objetivo de mitigar este efecto y permitir que los modelos predictivos (especialmente los basados en árboles) puedan capturar patrones estables en el tiempo, se propone una transformación de la variable objetivo utilizando una volatilidad móvil:

$$target\_scaled_t = \frac{target_t}{volatility\_rolling_t}$$

Donde:

- volatility_rolling es una media móvil de la desviación típica de la variable target, calculada sobre una ventana temporal definida (en este caso de 100 horas).
- Esta transformación permite que la magnitud de los retornos se interprete en relación con su contexto histórico reciente, normalizando los cambios estructurales en la varianza.

Esta estrategia es similar a una estandarización local y dinámica, adaptada a series temporales financieras.

```
data_estandarizado <- data_recodificado |>
  arrange(timestamp) |>
  mutate(volatility_rolling = rollapply(target_pct, width = 100, FUN = sd, fill = NA, align = "r
         target_scaled = target_pct / volatility_rolling) |>
  slice(-(1:100))
```

```
data_estandarizado <- data_estandarizado |>
  mutate(timestamp = as.POSIXct(timestamp, format = "%Y-%m-%d %H:%M:%S", tz = "UTC")) |>
  arrange(timestamp)

ggplot(data_estandarizado, aes(x = timestamp, y = volatility_rolling)) +
  geom_line() +
  geom_smooth(method = "lm", se = FALSE) +
  labs(title = "Evolución de la volatilidad") +
  theme_minimal()
```

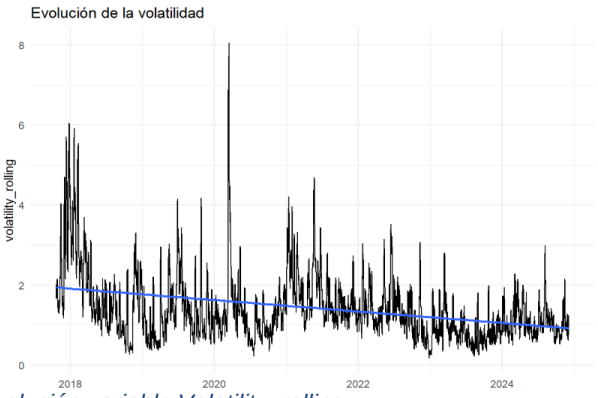


Ilustración 19. Evolución variable Volatility_rolling

Hacer los modelos con dichas volatilidades o impulsos y retrocesos estandarizada, hacemos las predicciones y luego deshacemos la transformación con la media móvil de volatilidad

para cuando vayamos a realizar la estrategia al mercado real. De esta manera vamos a aprender mucho mejor de los patrones pasados para aprender en el futuro solucionando este problema que nos muestra el halving y este cambio de volatilidad.

6.3. Distribución de la nueva variable objetivo "target_scaled"

Vemos ahora como se distribuye la nueva variable target_scaled en nuestros nuevos datos:

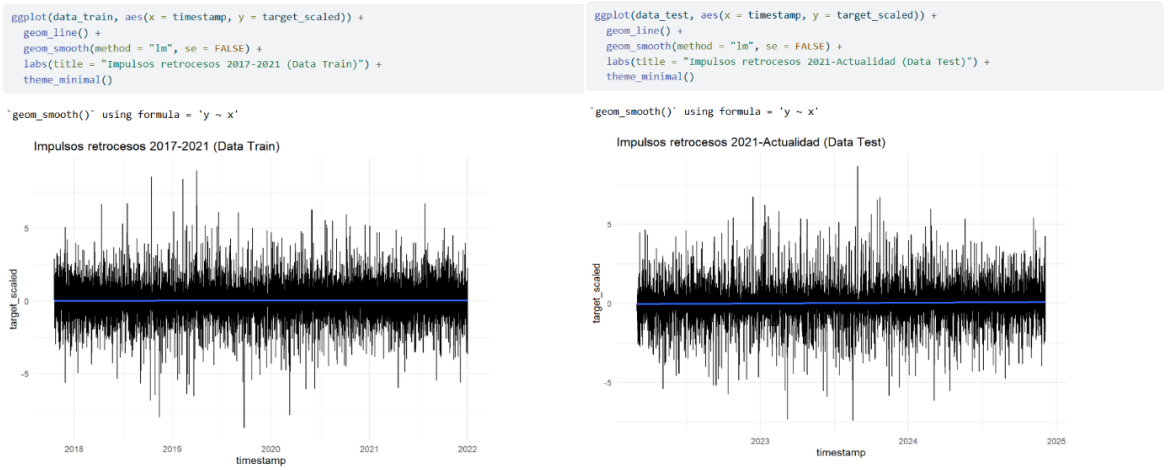


Ilustración 20. Impulsos y retrocesos estandarizados, data_train y data_test

Podemos ver ya que tienen comportamientos muchos más homogéneos entre los datos de entrenamiento con los de prueba. Para contrastar que estandarizando mejoramos la predicción y el aprendizaje de los modelos, lo veremos con primer modelo predictivo que vamos a

7. Presentación de los modelos de inversión diseñados.

Para abordar la predicción de impulsos y retrocesos en el precio de Bitcoin, se ha desarrollado una estrategia metodológica estructurada en tres fases, diseñada para equilibrar el rigor analítico con la viabilidad computacional del estudio. Dado el carácter altamente volátil y no lineal del mercado de criptomonedas, el uso de modelos flexibles y potentes se vuelve indispensable para capturar adecuadamente los patrones de comportamiento del precio.

En la primera fase, se han implementado y evaluado cinco familias de modelos de regresión: árboles de decisión, Bagging, Random Forest, Gradient Boosting y XGBoost. Todos estos modelos han sido entrenados y validados bajo una única estrategia de inversión común: la estrategia simple, basada en mantener la operación activa durante un periodo fijo de seis horas sin criterios adicionales de salida. Esta elección permite realizar una comparación homogénea entre modelos y centrarse únicamente en la capacidad predictiva de cada algoritmo. La métrica principal de evaluación en esta fase ha sido la ganancia promedio por operación (Directional_Error_Test). A partir de esta evaluación, se selecciona el modelo con mejor rendimiento como candidato para las siguientes fases.

En la segunda fase, se toma el mejor modelo identificado anteriormente y se realiza un nuevo ajuste de hiperparámetros más exhaustivo, integrando estrategias de inversión más complejas. Estas incluyen intervalos de confianza sobre la predicción, mecanismos de stop-loss basados en ratios dinámicos, y filtros que impiden abrir operaciones si el movimiento esperado no supera un umbral mínimo asociado al coste de transacción. Esta fase busca maximizar la rentabilidad de la estrategia.

En la tercera y última fase, se seleccionan los 15 mejores métodos de inversión surgidos de la combinación entre el modelo base elegido en la fase uno y las mejores estrategias optimizadas en la fase dos. Estos métodos se someten a una validación cruzada especialmente diseñada para series temporales: el "forward chain cross-validation con ventanas fijas", que permite mantener la estructura temporal de los datos y evaluar la estabilidad del modelo en distintos tramos de los datos de entrenamiento. Esta validación proporciona una visión más realista del desempeño

futuro del modelo y facilita la comparación robusta entre candidatos.

Cabe señalar que una alternativa más exhaustiva es probar todas las estrategias simples y complejas en cada uno de los modelos iniciales. Este método ha sido descartado por su inviabilidad práctica. El tiempo requerido para completar una sola grilla de Bagging con la estrategia simple puede oscilar entre cinco y siete días. Introducir múltiples variantes estratégicas multiplicaría los tiempos de computación hasta alcanzar meses enteros de espera por cada combinación. Por ello, esta metodología secuencial no solo responde a las limitaciones operativas del estudio, sino que también permite canalizar los recursos hacia aquellas combinaciones mejores en términos de rentabilidad y estabilidad.

7.1. Estrategias de inversión guiadas por la predicción de los modelos

En este apartado, definimos estrategias de trading basadas en nuestras predicciones para evaluar su rendimiento en datos de entrenamiento y prueba. Para ello, implementamos funciones que permiten probar diferentes enfoques y medir su efectividad en términos de rentabilidad y precisión en la detección de impulsos y retrocesos.

Cada estrategia se implementa como una función que toma las predicciones de los modelos XGBoost y Gradient Boosting. Estas funciones aplican reglas de decisión para generar señales de compra y venta, simulando operaciones en el mercado. El objetivo es identificar qué estrategias ofrecen mejores resultados y establecer criterios para optimizar su aplicación en escenarios reales. Las estrategias diseñadas son las siguientes:

7.1.1. Estrategia Simple (Baseline)

Este enfoque ha sido utilizado como referencia para validar los modelos. Consiste en abrir una operación en todas las observaciones y mantenerla activa durante 6 horas, saliendo automáticamente al final del período sin importar la evolución del precio en ese intervalo.

- **Objetivo:** Evaluar el error direccional promedio, es decir, si la predicción fue correcta en términos de dirección del movimiento.
- **Limitación principal:** No se optimiza la salida de la operación, ya que no se cierran posiciones antes de tiempo incluso si el precio alcanza el objetivo antes del instante x + 6 horas.

Ilustración 21. Ejemplo Ilustrativo Estrategia Simple

7.1.2. Estrategia con Intervalos de Confianza

Esta estrategia mejora la anterior estableciendo un intervalo de confianza sobre la predicción. En lugar de mantener la operación fija durante 6 horas, se monitorea hora por hora y se cierra la posición si el precio alcanza el intervalo de confianza antes del tiempo límite.

- **Ventaja principal:** Permite capturar beneficios antes de tiempo si el precio cumple la predicción antes del instante x + 6 horas.
- **Optimización:** Se probarán diferentes valores de Alpha (nivel de confianza) para determinar qué intervalo maximiza la rentabilidad.
- **Parámetros:** Los parámetros que evaluaremos junto a la predicción son c(0.01, 0.03, 0.05, 0.1)

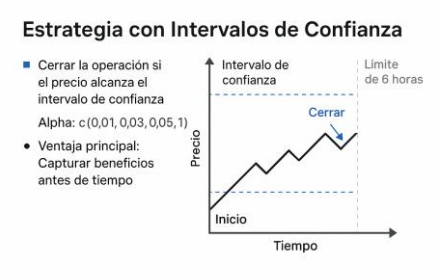


Ilustración 22. Ejemplo Ilustrativo Estrategia con Intervalos de Confianza

7.1.3. Estrategia con Intervalos de Confianza y Stop-Loss

Aquí añadimos un Stop-Loss basado en una ratio con la predicción de impulso o retroceso.

- **Ejemplo:** Si la predicción indica un impulso del 2%, se establece un Stop-Loss de -1% (ratio de 2:1).
- **Beneficio:** Control del riesgo limitando pérdidas en caso de que el precio se mueva en contra de la predicción.
- **Optimización:** Se probarán diferentes combinaciones de Alpha y Ratios para encontrar la mejor relación riesgo-beneficio.
- **Parámetros:** Los parámetros que evaluaremos junto a la predicción son c(1, 1.5, 2, 3)

Ilustración 23. Ejemplo Ilustrativo Estrategia con Intervalos de Confianza y StopLoss

7.1.4. Estrategia con Intervalos de Confianza, Stop-Loss y Coste de Transacción

Finalmente, añadimos un criterio de entrada condicionado por los costos de transacción.

- **Regla:** Si la predicción del impulso o retroceso predicho es menor que el coste de transacción, no se abre la operación, evitando pérdidas aseguradas.
- **Optimización:** Se explorará una grilla de distintos valores de alpha, ratios y umbrales de coste para determinar en qué condiciones la estrategia sigue siendo rentable.
- **Parámetros:** Los parámetros que evaluaremos junto a la predicción son c(2, 2.5, 3, 3.5,4)

Ilustración 24. Ejemplo Ilustrativo Estrategia con Intervalos de Confianza, Stop-Loss y Umbral Coste Transacción

Todas estas estrategias presentan una limitación en las simulaciones, ya que las decisiones se toman de manera determinista al inicio de cada vela horaria, sin posibilidad de intervenir dentro de la misma.

- En un mercado real, podríamos tomar decisiones en cualquier instante dentro de la hora.
- Por ello, el análisis inicial se realizará bajo esta decisión horaria, y luego en investigaciones

posteriores se evaluará la mejor combinación de modelo y estrategia con un backtesting estadístico más realista para acercarnos a las condiciones del mercado en vivo.

Estas estrategias nos permitirán comparar modelos y evaluar su desempeño de manera más efectiva, asegurándonos de seleccionar la mejor combinación antes de aplicar un análisis de backtesting real.

7.2. Métricas de evaluación de rendimiento

Dado el carácter práctico y aplicado de esta investigación, el objetivo no es únicamente optimizar la capacidad predictiva de los modelos según métricas estadísticas tradicionales como el R^2 o el RMSE. En su lugar, se ha diseñado un conjunto de métricas de evaluación adaptadas al contexto financiero, que permiten valorar el comportamiento del modelo bajo una estrategia de inversión simulada.

Además, es importante destacar que los criterios de evaluación utilizados para seleccionar los modelos varían a lo largo de las diferentes fases del estudio. En la Fase 1, el foco se sitúa exclusivamente en la métrica de rentabilidad promedio por operación (Directional_Error_Test), con el objetivo de identificar el modelo base con mayor capacidad predictiva bajo una estrategia simple.

No obstante, en la Fase 3, la selección del modelo final se realiza atendiendo a una evaluación multicriterio, que incorpora no solo la rentabilidad promedio, sino también el porcentaje de operaciones activadas, la variabilidad de los resultados obtenidos a través de la validación cruzada temporal con ventanas fijas, así como la correlación entre el rendimiento del modelo y la volatilidad del activo. Esta última se analiza empleando la variable volatility_rolling, la cual también se ha utilizado previamente para estandarizar la variable objetivo. Este enfoque integral busca garantizar que el modelo final no solo sea rentable en promedio, sino también estable, robusto frente a condiciones de mercado cambiantes y poco sensible a variaciones de volatilidad, reforzando así su aplicabilidad práctica en entornos reales de trading algorítmico.

Para el buen cálculo de estas métricas se ha deshecho el cambio de la variable respuesta, multiplicando de nuevo por volatility_rolling (media móvil de desviaciones típicas). A continuación, se describen las principales métricas utilizadas:

- **Directional Accuracy (Train y Test)**: Esta métrica mide la proporción de veces que el modelo acierta la dirección del movimiento del precio (es decir, si sube o baja) comparando el signo de la predicción con el signo del resultado observado tras aplicar la estrategia. Se calcula como:

$$Direction\ Accuracy = \frac{1}{n}\sum_{i=1}^{n}[sing(\widehat{y_i}) == sing(y_i^{estrategia})]$$

- **Directional Error (Train y Test)**: Es la métrica principal del estudio y representa la **rentabilidad promedio por operación**, considerando la dirección predicha y la magnitud del retorno alcanzado por la estrategia. Se define como:

$$Directional\ Error = \frac{1}{n}\sum_{i=1}^{n} sign(\hat{y}_i) \times y_i^{estrategia}$$

  Este indicador no solo penaliza errores de dirección, sino también operaciones en las que el retorno real fue pequeño a pesar de acertar la tendencia.

- **Bullish/Bearish Accuracy**: Estas métricas descomponen el acierto direccional en función del tipo de movimiento. *Bullish Accuracy* evalúa cuántas veces se acertaron correctamente los impulsos (movimientos positivos), mientras que *Bearish Accuracy* evalúa la precisión en detectar retrocesos (movimientos negativos). Este análisis diferenciado permite entender si el modelo es más eficaz detectando subidas o bajadas.

- **Percentage of Operations Activated**: Esta métrica refleja el porcentaje de observaciones sobre las que efectivamente se ejecutó una operación, tras aplicar un filtro de intensidad

basado en la predicción del modelo. En particular, se requiere que el valor absoluto de la predicción sea superior a un umbral mínimo definido como el coste de transacción multiplicado por una ratio de entrada (ratio_coste), es decir:

$$Operación\ activada\ si: |\hat{y}| > ratio_{coste} \times coste_{operación}$$

Este criterio permite evitar operar en momentos de alta incertidumbre o baja expectativa de rentabilidad.

Tras la realización de todas las pruebas de los modelos solo mostraremos en los primeros modelos correspondientes a la fase 1 de árboles de regresión, bagging y random forest las dos primeras métricas, las demás si aparecerán en los modelos más complejos como los de gradient boosting y XGBoost. Aun así, simplemente las métricas de *Bullish/Bearish Accuracy* como *Directional Accuracy (Train y Test)* serán meramente informativas no tendrán ninguna importancia en la toma de decisiones. Le daremos mucha más importancia en futuras investigaciones donde nos va a interesar captar estas direcciones de mercado antes de la rentabilidad por operación.

7.3. Partición de los datos en entrenamiento y prueba.

La división de los datos en conjuntos de entrenamiento y validación es un paso fundamental en cualquier proceso de modelado predictivo, ya que permite estimar la capacidad de generalización del modelo. En este caso, los datos de entrenamiento comprenden el periodo desde el origen de la base de datos, en 2017, hasta finales de 2021, mientras que los datos de prueba se extienden desde enero de 2022 hasta diciembre de 2024, última fecha disponible. Esta separación temporal garantiza que las predicciones se evalúen en un contexto realista, sin riesgo de fuga de información desde el futuro al pasado.

Además, para evitar dependencias temporales en el conjunto de validación, se han eliminado las primeras 1440 observaciones (equivalentes a 60 días en temporalidad horaria), ya que algunas de las variables predictoras utilizan ventanas de hasta 1440 observaciones pasadas para su cálculo (por ejemplo, en retrocesos de Fibonacci o medias móviles suavizadas). Esto asegura que la evaluación del modelo se realice con observaciones verdaderamente independientes del conjunto de entrenamiento.

Cabe destacar que todos los modelos han sido inicialmente entrenados y validados una sola vez sobre esta partición fija. Sin embargo, en la Fase 3 de este estudio, se ha implementado una evaluación más rigurosa mediante validación cruzada temporal tipo "forward chain" con ventanas fijas, en la que el conjunto de entrenamiento se divide en 6 bloques consecutivos. En cada iteración, un bloque se utiliza para entrenar el modelo y el siguiente para validarlo, lo que permite obtener una estimación más robusta y estable del comportamiento del modelo en diferentes momentos del tiempo.

8. Árboles de regresión.

Los árboles de regresión son modelos predictivos basados en estructuras jerárquicas en forma de árbol, utilizados para estimar valores continuos. Su funcionamiento se basa en dividir iterativamente el conjunto de datos en subconjuntos más homogéneos, mediante reglas de decisión sobre las variables independientes. En cada nodo del árbol, el algoritmo selecciona la variable y el punto de corte que minimizan la suma de errores cuadráticos dentro de los grupos resultantes (en este caso el MSE). Este proceso continúa hasta que se cumple algún criterio de parada, como un número mínimo de observaciones en las hojas o una profundidad máxima del árbol.

En las hojas terminales, el valor predicho corresponde al promedio de los valores de la variable objetivo en ese grupo. Esta estructura permite capturar relaciones no lineales y efectos de interacción entre variables, aunque los árboles individuales suelen ser inestables y propensos al sobreajuste si no se podan o combinan con técnicas como bagging o boosting.

Empezamos realizando un grid de los diferentes hiperparámetros que puede tener un árbol de regresión simple, como son el tamaño mínimo por hoja, criterio de podado y tamaño real

resultante del podado. Evaluaremos los diferentes hiperparámetros con la métrica de Directional_Error_Test.

Nos quedaremos con aquellos hiperparámetros que nos parezcan interesantes a la hora de empezar con modelos más interesantes como Bagging y Random Forest, culminando los modelos de árboles con los Gradient Boosting y XGBoost.

8.2. Diseño de la grilla

Para este modelo realizaremos tanto para la variable target_pct (impulsos y retrocesos en %) como para target_scaled la variable objetivo-estandarizada con el objetivo de demostrar que la variable estandarizada captura mejor los patrones del precio. Podríamos hacer las pruebas para todos los tipos de modelos, pero por falta de tiempo y espacio en el TFG supondremos que nos podremos guiar por los resultados que obtengamos en árboles de regresión.

```
# Definir grilla de hiperparámetros
grid <- expand.grid(
  minbucket = ceiling(seq(0.001, 0.01, length.out = 10) * nrow(data_train1)), # Tamaño de hoja i
  cp = seq(0, 0.1, length.out = 10)  # Criterio de poda
)
```

8.3. Análisis de los resultados de la grilla

Vemos los primeros 12 mejores modelos de la grilla, ordenados por la métrica de rentabilidad promedio por operación (Directional_Error_Test):

Modelo	V. Objet.	minbucket	cp	Estrategia	Direction_Train	Direction_Test
1	Target_pct	332	0	Simple	0,58	0,512
2	Target_pct	258	0	Simple	0,585	0,517
3	Target_pct	111	0	Simple	0,591	0,505
4	Target_pct	221	0	Simple	0,592	0,511
5	Target_pct	148	0	Simple	0,604	0,502
6	Target_pct	295	0	Simple	0,571	0,491
7	Target_pct	185	0	Simple	0,6	0,494
8	Target_pct	37	0,01	Simple	0,52	0,509
9	Target_pct	74	0,01	Simple	0,52	0,509
10	Target_pct	37	0	Simple	0,593	0,497
11	Target_pct	74	0	Simple	0,587	0,502
12	Target_pct	369	0	Simple	0,578	0,49

Modelo	Directional_Error_Train	Directional_Error_Test	real_minbucket
1	0,0811	0,0119	333
2	0,0946	0,0112	258
3	0,1438	0,0086	111
4	0,1131	0,0071	221
5	0,1424	0,0029	148
6	0,0840	0,0015	295
7	0,1256	0,0010	185
8	0,0064	0,0010	40
9	0,0020	0,0009	36808
10	0,2273	-0,0058	37
11	0,1764	-0,0045	74

12	0,0775	-0,0107	372

Ilustración 25. Resultados grid áboles de regresión factorial

Para estos primeros resultados de la grilla, a partir que empezar a utilizar el criterio de poda se me empiezan a agrupar todos los datos en una única observación como vemos en el real_minbucket.

Aquí podemos sospechar que si los datos están en diferente escala cuando creamos los modelos en Train entonces cuando validemos en Test las predicciones no serán buenas porque tienen diferente variabilidad, dicho de otra forma, que nuestro modelo presenta heterocedasticidad y por ende no es estacionario.

Vamos a ver ahora con el mismo grid de antes con las variables objetivo estandarizada, para ver si mejora la bondad con la métrica de rentabilidad promedio por operación "Directional_Error_Test":

Modelos	V. Objet.	minbucket	cp	Estrategia	Direction_Train	Direction_Test
1	Target_scaled	37	0	Simple	0,6843	0,5094
2	Target_scaled	74	0	Simple	0,6407	0,5002
3	Target_scaled	111	0	Simple	0,6162	0,5071
4	Target_scaled	148	0	Simple	0,6085	0,5125
5	Target_scaled	185	0	Simple	0,6027	0,5120
6	Target_scaled	221	0	Simple	0,5909	0,5058
7	Target_scaled	258	0	Simple	0,5834	0,5124
8	Target_scaled	295	0	Simple	0,5791	0,5031
9	Target_scaled	332	0	Simple	0,5735	0,5011
10	Target_scaled	369	0	Simple	0,5741	0,4992
11	Target_scaled	37	0,01	Simple	0,5198	0,5092
12	Target_scaled	74	0,01	Simple	0,5198	0,5092

Modelos	Directional_Error_Train	Directional_Error_Test	real_minbucket
1	0,2300	0,0176	37
2	0,1417	0,0135	74
3	0,1220	0,0107	111
4	0,1050	0,0810	148
5	0,0946	0,0489	185
6	0,0626	0,0318	221
7	0,0540	0,0092	258
8	0,0481	0,0068	295
9	0,0428	0,0007	332
10	0,0396	0,0002	373
11	0,0014	0,0006	36808
12	0,0014	0,0006	36808

Ilustración 26. Resultados grid Árboles de Regresión variable target estandarizada

Los resultados muestran una mejora significativa en la capacidad predictiva del modelo estandarizado, tanto en métricas de ajuste como en la capacidad direccional:

- Las métricas directional_train y directional_test, que evalúan la precisión en la predicción del sentido del movimiento del activo, también mejoraron notablemente.
- Además, se observó una reducción en los errores direccionales tanto en entrenamiento como en prueba (directional_error_train y directional_error_test), lo que sugiere que el modelo con la variable escalada no solo ajusta mejor, sino que también interpreta con mayor precisión los cambios de tendencia del precio.

Este resultado respalda la hipótesis inicial de que la presencia de heterocedasticidad estaba

afectando negativamente el rendimiento del modelo, y que su corrección mediante una estandarización adaptativa a la volatilidad mejora sustancialmente la robustez de las predicciones.

Hacemos una prueba gráfica con una muestra aleatoria de 200 observaciones de cómo se ajuste la variable target_pct y el target_scaled:

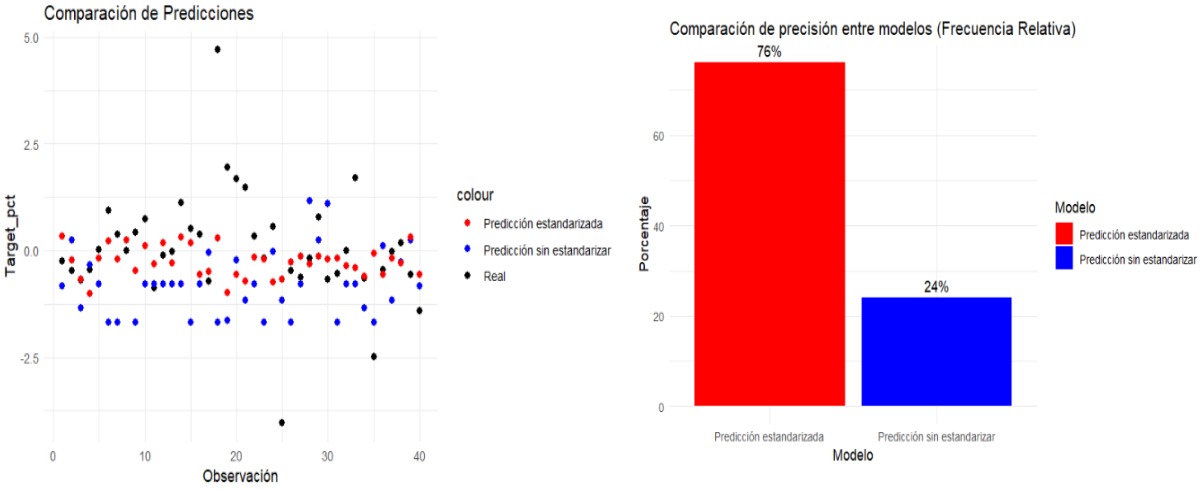


Ilustración 27. Target_pct vs Target_scaled

La prueba gráfica realizada con una muestra aleatoria de 200 observaciones refuerza los resultados previos obtenidos con las métricas de evaluación. En el gráfico de la izquierda se observa cómo las predicciones generadas a partir de la variable target_scaled (en rojo) se ajustan de forma más consistente al valor real en comparación con las predicciones sin estandarizar (en azul). Esta mejor adaptación se ve reflejada también en el gráfico de la derecha, donde el 76% de las predicciones más precisas corresponden al modelo con la variable estandarizada, frente al 24% del modelo sin estandarizar.

Esto confirma que la corrección de la heterocedasticidad mediante la estandarización adaptativa mejora la capacidad del modelo para capturar tanto el valor como la dirección del movimiento del activo, ofreciendo predicciones más robustas y confiables en distintos contextos de mercado.

Pasamos ahora a realizar un grid de Bagging y Random Forest. De los resultados que sacamos con los árboles de regresión es que los modelos que mejor ajustan son aquellos con un tamaño de hoja final y vemos como no hace falta realizar poda, que los árboles son bastante explicativos. De todas formas, en los siguientes modelos estudiaremos distintas variaciones de los tamaños finales de hoja para ver si siguen siendo mejores en este tipo de modelos.

9. Bagging y Random Forest

Si bien los árboles de regresión ofrecen interpretabilidad y simplicidad, un único árbol presenta alta varianza: pequeñas variaciones en los datos de entrenamiento pueden generar árboles muy distintos. Esto puede llevar a un problema de sobreajuste (overfitting) en datasets complejos o con ruido.

Para superar esta limitación surge el método de Bagging (Bootstrap Aggregating). Esta técnica consiste en generar múltiples subconjuntos del conjunto de entrenamiento mediante muestreo con reemplazo (bootstrap), ajustando un árbol de regresión independiente en cada subconjunto. Posteriormente, las predicciones de todos los árboles se combinan (en este caso, mediante promedio en regresión) para obtener una predicción final más estable. De esta forma, Bagging reduce la varianza del modelo y mejora su capacidad de generalización, manteniendo el sesgo bajo, ya que los árboles no se podan ni se regularizan fuertemente.

El modelo de Random Forest es una evolución del Bagging que incorpora una capa adicional de aleatoriedad: en cada división del árbol, en lugar de considerar todas las variables

disponibles, se selecciona aleatoriamente un subconjunto de variables candidatas para realizar la mejor partición. Esta estrategia disminuye aún más la correlación entre los árboles individuales, aumentando la diversidad del conjunto y mejorando la precisión general del modelo.

Solo vamos a predecir la variable objetivo-estandarizada tras la gran diferencia con la variable objetivo sin estandarizar vista en árboles de regresión.

9.1. Diseño de la grilla:

```
grid <- expand.grid(
  sample_size = seq(0.2, 1, by = 0.2) * nrow(data_train),  # Tama
  mtry = ceiling(seq(0.2, 1, by = 0.2) * (ncol(data_train) - 1)),
  ntree = seq(100, 300, by = 100),
  sizenode = c(0.005, 0.01, 0.02, 0.05)# N◆mero de ◆rboles
)
```

El parámetro sample_size controla el tamaño de la muestra utilizada en cada bootstrap, variando entre el 20% y el 100% del conjunto de entrenamiento. El parámetro mtry determina el número de variables consideradas aleatoriamente en cada partición del árbol, expresado como un porcentaje del total de variables predictoras. Esta aleatorización es especialmente relevante en Random Forest, ya que fomenta la diversidad entre árboles y mejora el rendimiento general del modelo.

Por otro lado, ntree indica el número total de árboles que se entrenarán, variando entre 100, 200 y 300. Un mayor número de árboles suele mejorar la estabilidad de las predicciones, aunque incrementa el coste computacional. Finalmente, sizenode establece el tamaño mínimo de las particiones terminales (nodos hoja) como un porcentaje del tamaño de entrenamiento. Valores más pequeños permiten capturar patrones locales más específicos, mientras que valores más altos favorecen árboles más simples y generalistas. Este diseño de grilla equilibra la exploración de configuraciones complejas y conservadoras, adaptándose a la naturaleza no lineal y ruidosa de los datos financieros analizados.

9.2. Resultados de la grilla

Vemos los primeros 12 mejores modelos de la grilla, ordenados por la métrica de rentabilidad promedio por operación (Directional_Error_Test):

Modelo	sample_size	mtry	ntree	nodesize	Estrategia
1	7362	117	200	185	Simple
2	14723	176	100	737	Simple
3	7362	234	200	369	Simple
4	22085	292	100	737	Simple
5	7362	292	300	185	Simple
6	29446	117	100	369	Simple
7	7362	59	100	185	Simple
8	22085	176	300	369	Simple
9	14723	117	300	185	Simple
10	14723	234	100	737	Simple
11	22085	59	200	369	Simple
12	14723	117	200	185	Simple

Modelo	Direction_Train	Direction_Test	Directional_Error_Train	Directional_Error_Test
1	0,6253	0,5022	0,4667	0,0101
2	0,6209	0,4937	0,4869	0,0088

3	0,6031	0,5005	0,4018	0,0078
4	0,6650	0,5009	0,6487	0,0078
5	0,6275	0,4966	0,4628	0,0059
6	0,7269	0,4971	0,8430	0,0056
7	0,6177	0,4997	0,4316	0,0051
8	0,7096	0,4939	0,7764	0,0049
9	0,7000	0,4966	0,7262	0,0036
10	0,6352	0,4992	0,5326	0,0020
11	0,7000	0,4977	0,7434	0,0011
12	0,6891	0,5045	0,6810	0,0009

Ilustración 28. Resultados grid Bagging y Random Forest

En esta fase se evaluaron distintos modelos de Random Forest y Bagging mediante una búsqueda en rejilla (grid search), variando los principales hiperparámetros: el tamaño de la muestra (sample_size), el número de variables consideradas por árbol (mtry), el número total de árboles (ntree) y el tamaño mínimo de los nodos hoja (nodesize). La métrica principal utilizada fue el Directional Error Test, que representa la ganancia media por operación en el conjunto de test, complementada por la precisión en la predicción de la dirección del precio. Los resultados muestran una fuerte sensibilidad a la elección de hiperparámetros, observándose en muchos casos un elevado sobreajuste: modelos con altas ganancias en entrenamiento, pero bajo rendimiento en test. Esto es evidente, por ejemplo, en el modelo con sample_size alto y mtry = 117, donde la ganancia en entrenamiento supera el 0.84, pero en test se reduce drásticamente a 0.0056.

El mejor modelo identificado presenta una combinación equilibrada de hiperparámetros (sample_size = 7361.6, mtry = 117, ntree = 200, nodesize = 185) y obtiene el mayor Directional_Error_Test del conjunto (0.0101), con una diferencia moderada respecto al entrenamiento. Este resultado sugiere que, en este caso, un tamaño de muestra reducido junto con un número medio de variables y árboles permite un mayor grado de aleatoriedad y evita el sobreajuste, mejorando la capacidad de generalización del modelo. En conjunto, los resultados resaltan la importancia de regular adecuadamente la complejidad del modelo, ya que configuraciones con mtry muy altos o tamaños de muestra excesivos tienden a ajustar el ruido del entrenamiento sin aportar mejoras en test.

En cuanto a que valores de los parámetros de random forest quedarnos, no podemos decidirnos por ninguno claro, ya que no han presentado resultados buenos, que nos puedan servir para el diseño del grid de modelos más complejos como los de gradient boosting o XGBoost, por lo tanto, volveremos a probar diferentes combinaciones de hiperparámetros de random forest para los siguientes modelos.

10. Gradient Boosting y XGBoost

En esta etapa del análisis nos enfocamos en dos modelos muy utilizados por su capacidad para capturar relaciones complejas y no lineales: Gradient Boosting y XGBoost. Ambos modelos pertenecen a la familia de los métodos de boosting, una técnica de ensamblado que construye modelos de manera secuencial, donde cada nuevo árbol trata de corregir los errores cometidos por los anteriores (Friedman, J. H., 2001). Esto contrasta con métodos como Bagging y Random Forest, que entrenan múltiples árboles en paralelo y promedian sus resultados para reducir la varianza del modelo (Chen, T., & Guestrin, C, 2016).

Mientras que Bagging busca robustez a partir de la combinación de modelos débiles entrenados de manera independiente, Gradient Boosting y XGBoost priorizan la mejora iterativa del rendimiento a través del aprendizaje de los residuos, logrando modelos generalmente más precisos, pero también más sensibles al sobreajuste si no se ajustan correctamente los hiperparámetros.

Dado que los grids de hiperparámetros explorados previamente para Random Forest y Bagging no arrojaron resultados suficientemente sólidos ni consistentes, diseñamos una nueva

grilla de hiperparámetros desde cero enfocada exclusivamente en modelos de boosting. En esta sección explicaremos cómo funcionan estos modelos, cuáles son los hiperparámetros clave para cada uno y su significado, y presentaremos los resultados obtenidos con los mejores modelos encontrados en esta nueva búsqueda, incluyendo sus métricas de error tanto en entrenamiento como en prueba.

10.1. Grilla Gradient Boosting

10.1.1. Diseño de la grilla:

```
grid <- expand.grid(
  sample_size = nrow(data_train),
  shrinkage = c(0.01, 0.05, 0.1),
  n.trees = c(100, 200, 300),
  sizenode = c(0.005, 0.01, 0.02, 0.05),
  mtry = 50,
  distribution = c("gaussian")
)
```

- **Sample_size:** Este valor asegura que el modelo entrene con todas las observaciones disponibles del conjunto de entrenamiento. Es decir, no se realiza submuestreo entre combinaciones.
- **Shrinkage = c(0.01, 0.05, 0.1):** Este es el learning rate del modelo, también conocido como tasa de aprendizaje. Cuanto más pequeño es este valor, más lento y conservador es el aprendizaje, pero puede permitir mejor generalización si se entrena con suficientes árboles.
    - 0.01: muy conservador
    - 0.05: moderado
    - 0.1: más rápido, pero mayor riesgo de sobreajuste
- **N.trees = c(100, 200, 300**): Número total de árboles que se construyen en el modelo. Más árboles permiten aprender patrones más complejos, pero incrementan el riesgo de sobreajuste si no se controlan bien otros hiperparámetros como shrinkage.
- **sizenode = c(0.005, 0.01, 0.02, 0.05):** Proporción del total de observaciones que debe haber como mínimo en cada nodo terminal (hoja) del árbol. Se transforma luego a número absoluto con:
  n.minobsinnode = ceiling(sizenode * nrow(data_train))
  Valores más pequeños permiten árboles más profundos con divisiones muy finas, lo que puede llevar al sobreajuste. Valores más altos generan árboles más generalistas.
- **mtry = 50:** Número de variables independientes (features) seleccionadas aleatoriamente para construir el modelo en cada iteración. Aunque GBM no selecciona variables aleatorias como Random Forest, aquí estás usando este valor para limitar las variables en la fórmula de entrenamiento (selected_vars), lo que actúa como una forma de regularización.
- **distribution = "gaussian":** Tipo de distribución de error que utiliza el modelo. "gaussian": indica que estamos ante un problema de regresión con errores normales (distribución normal de los residuos).

10.1.2. Resultados de la grilla

Vemos los primeros 12 mejores modelos de la grilla, ordenados por la métrica de rentabilidad promedio por operación (Directional_Error_Test):

modelo	Estrategia	sample_size	n.trees	sizenode	shrinkage	mtry
1	Simple	36808	100	0,02	0,1	50
2	Simple	36808	100	0,01	0,01	50
3	Simple	36808	100	0,02	0,01	50
4	Simple	36808	100	0,01	0,05	50
5	Simple	36808	100	0,05	0,01	50
6	Simple	36808	100	0,02	0,05	50
7	Simple	36808	200	0,05	0,01	50

8	Simple	36808	300	0,02	0,05	50
9	Simple	36808	200	0,02	0,05	50
10	Simple	36808	200	0,02	0,1	50
11	Simple	36808	300	0,05	0,01	50
12	Simple	36808	100	0,005	0,01	50

D_Train	D_Test	DE_Train	DE_Test	BullA_Train	BeaA_Train	BullA_Test	BeaA_Test
0,627	0,522	0,3312	0,0389	0,6744	0,5778	0,3537	0,6977
0,564	0,513	0,1714	0,0320	0,8226	0,2845	0,5908	0,4317
0,566	0,505	0,1717	0,0269	0,7815	0,3330	0,5167	0,4920
0,613	0,515	0,2981	0,0224	0,6856	0,5366	0,3909	0,6429
0,568	0,505	0,1731	0,0211	0,7682	0,3513	0,4057	0,6071
0,609	0,506	0,2896	0,0185	0,6688	0,5459	0,2941	0,7263
0,579	0,503	0,2074	0,0177	0,7137	0,4339	0,3331	0,6801
0,647	0,512	0,3780	0,0170	0,6923	0,5998	0,2534	0,7799
0,631	0,513	0,3408	0,0163	0,6853	0,5744	0,3068	0,7270
0,655	0,509	0,3997	0,0116	0,6931	0,6163	0,3288	0,6951
0,590	0,503	0,2340	0,0115	0,6996	0,4721	0,2554	0,7593
0,559	0,502	0,1588	0,0091	0,8534	0,2403	0,2955	0,7155

Ilustración 29. Resultados grid Gradient Boosting

(*D_Train: Directional_Train, D_Test: Directional_Test, DE_Train: Directional_Error_Train, DE_Test: Directional_Error_Test, BullA_Train: Bullish_Acurracy_Train, BullA_Test: Bullish_Acurracy_Test, BeaA_Train: Bearish_Acurracy_Train, BeaA_Test: Bearish_Acurracy_Test*)

En los resultados del grid de Gradient Boosting, observamos que los mejores modelos según la métrica Directional_Error_Test alcanzan valores cercanos a 0.0389 en el mejor caso (modelo 1), aunque la mayoría de los modelos top se sitúan entre 0.017 y 0.033. Este mejor modelo también muestra un buen equilibrio entre precisión direccional en test (D_Test = 0.522) y una aceptable precisión en fases alcistas (BullA_Test = 0.3537) y bajistas (BeaA_Test = 0.6977), lo que indica que el modelo es especialmente eficaz identificando escenarios bajistas. Modelos como el 7, 9 y 10 también destacan por combinar un buen DE_Test con una alta precisión en fases bajistas (BeaA_Test > 0.70), lo cual es valioso en estrategias de cobertura o trading en mercados volátiles.

Dado que la ganancia promedio por operación aún no alcanza niveles óptimos en la mayoría de las configuraciones del grid, y que el rendimiento varía de forma sensible dependiendo de combinaciones de shrinkage, sizenode y n.trees, procederemos a comparar estos resultados con los obtenidos mediante modelos XGBoost. Esta comparación nos permitirá identificar qué algoritmo ofrece una mejor capacidad predictiva y rentabilidad operativa dentro de nuestro marco de modelización.

10.2. Grilla XGBoost

10.2.1 Diseño de la grilla:

```
grid <- expand.grid(
  nrounds = c(20, 50, 100),
  eta = c(0.01, 0.05, 0.1),
  sizenode = c(0.005, 0.01, 0.05),
  colsample_bytree = c(0.25, 0.5, 0.75,1),
  lambda = c(0, 1, 5) ,        # Regularizaci◆n L2 (Ridge)
  alpha = c(0, 1, 5)
  #max_depth = c(3, 6, 9, 12)

)
```

Con el objetivo de optimizar la capacidad predictiva del modelo XGBoost, se diseñó la siguiente grilla de hiperparámetros:

- **nrounds = c(20, 50, 100):** Número total de iteraciones (árboles) que se construyen en el modelo de XGBoost. Cada árbol intenta corregir los errores de los anteriores.
  - 20: muy pocos árboles, aprendizaje muy rápido, pero menos preciso.
  - 50: equilibrio entre velocidad y ajuste.
  - 100: más árboles permiten capturar patrones más complejos, aunque aumentan el riesgo de sobreajuste si no se controla bien el resto de los parámetros.
- **eta = c(0.01, 0.05, 0.1):** Tasa de aprendizaje (learning rate) del modelo. Controla cuánto se corrige en cada iteración. Cuanto más pequeño es este valor, más gradual y cuidadoso es el aprendizaje.
  - 0.01: muy conservador, requiere más árboles para buen rendimiento.
  - 0.05: conservador/moderado.
  - 0.1: aprendizaje más rápido, pero con mayor riesgo de sobreajuste.
- **sizenode = c(0.005, 0.01, 0.05):** Proporción mínima de observaciones que debe contener cada hoja de los árboles. Se transforma en número absoluto con: min_child_weight = ceiling(sizenode * nrow(data_train)). Valores más pequeños permiten árboles más profundos y detallados (mayor riesgo de sobreajuste); valores mayores generan árboles más simples y generalistas.
- **colsample_bytree = c(0.25, 0.5, 0.75, 1):** Proporción de variables que se seleccionan aleatoriamente para construir cada árbol.
  - 0.25: cada árbol utiliza solo el 25% de las variables disponibles → árboles más diferentes entre sí (más diversidad).
  - 1: cada árbol usa todas las variables, puede ser más potente, pero menos diverso.
- **lambda = c(0, 1, 5):** Parámetro de regularización L2 (Ridge). Penaliza la magnitud de los coeficientes del modelo para evitar sobreajuste.
  - 0: sin penalización.
  - 1: regularización moderada.
  - 5: regularización fuerte.
- **alpha = c(0, 1, 5):** Parámetro de regularización L1 (Lasso). Incentiva la selección de variables más importantes al reducir a cero algunas variables menos relevantes.
  - 0: sin penalización.
  - 1: moderada selección de variables.
  - 5: selección más agresiva de variables (más parcidad).

10.2.2. Resultados de la grilla

Vemos los primeros 12 mejores modelos de la grilla, ordenados por la métrica de rentabilidad promedio por operación (Directional_Error_Test):

Modelo	sizenode	lambda	alpha	nrounds	eta	mtry	Estrategia
1	0,05	0	0	20	0,1	0,5	Simple
2	0,05	5	1	20	0,1	0,7	Simple
3	0,05	0	10	20	0,1	1	Simple
4	0,05	5	5	20	0,1	0,5	Simple
5	0,05	5	1	20	0,1	0,5	Simple
6	0,05	10	10	20	0,1	0,5	Simple
7	0,05	10	5	20	0,1	0,7	Simple
8	0,05	0	10	50	0,05	0,9	Simple
9	0,05	0	5	50	0,05	1	Simple
10	0,02	1	1	20	0,1	0,7	Simple
11	0,05	5	10	20	0,1	1	Simple
12	0,05	5	1	20	0,1	0,9	Simple

D_Train	D_Test	DE_Train	DE_Test	BullA_Train	BeaA_Train	BullA_Test	BeaA_Test
0,5601	0,5163	0,1451	0,0547	0,8964	0,1968	0,6564	0,3711

0,5583	0,5154	0,1446	0,0539	0,8888	0,2012	0,7073	0,3164
0,5571	0,5151	0,1438	0,0497	0,8875	0,2001	0,7034	0,3198
0,5577	0,5198	0,1446	0,0486	0,8926	0,1958	0,7652	0,2651
0,5606	0,5143	0,1485	0,0484	0,8991	0,1948	0,7762	0,2426
0,5585	0,5120	0,1500	0,0484	0,8952	0,1947	0,7380	0,2777
0,5607	0,5140	0,1501	0,0482	0,8869	0,2083	0,7556	0,2634
0,5726	0,5153	0,1900	0,0479	0,8186	0,3074	0,5224	0,5081
0,5737	0,5157	0,1904	0,0477	0,8146	0,3139	0,4867	0,5459
0,5608	0,5144	0,1631	0,0473	0,8885	0,2069	0,6841	0,3384
0,5589	0,5146	0,1452	0,0472	0,8869	0,2044	0,7019	0,3203
0,5594	0,5162	0,1468	0,0471	0,8840	0,2088	0,6906	0,3353

Ilustración 30. Resultados grid XGBoost

(*D_Train: Directional_Train, D_Test: Directional_Test, DE_Train: Directional_Error_Train, DE_Test: Directional_Error_Test, BullA_Train: Bullish_Acurracy_Train, BullA_Test: Bullish_Acurracy_Test, BeaA_Train: Bearish_Acurracy_Train, BeaA_Test: Bearish_Acurracy_Test*)

En los resultados del grid para XGBoost, observamos una mejora consistente en la métrica Directional_Error_Test respecto al modelo de Gradient Boosting tradicional. Los mejores modelos (como el modelo 1 y el modelo 2) alcanzan una ganancia promedio por operación de hasta 0.0547 y 0.0539 respectivamente, superando los valores obtenidos previamente con Gradient Boosting, cuyo mejor modelo se situó en 0.0389. Además, estos modelos de XGBoost muestran una buena precisión en fases alcistas (BullA_Test en torno a 0.70) aunque una menor capacidad predictiva en escenarios bajistas (BeaA_Test < 0.38 en general), lo cual sugiere un sesgo hacia tendencias positivas del mercado. Cabe destacar que el modelo 5 logra un balance interesante entre precisión alcista y bajista (0.7762 y 0.2426 respectivamente), con un DE_Test de 0.0484.

Pese a que XGBoost mejora levemente la rentabilidad media por operación, ningún modelo logra superar de forma holgada un umbral de ganancia suficiente como para cubrir costes operativos reales, que estimamos en torno al 0.05% por operación. Por tanto, concluimos que es necesario explorar nuevas estrategias que ayuden a guiar mejor la predicción y a captar relaciones no lineales más complejas que puedan mejorar sustancialmente la rentabilidad neta de las operaciones.

Damos por finalizada la primera fase de análisis de modelos de predicción donde nos vamos a quedar con la metodología de XGBoost, ahora pasamos a la segunda fase donde vamos a añadir las nuevas estrategias de inversión más complejas mencionadas en el apartado 7.1.

11. XGBoost con nuevas estrategias

Puesto que encontramos mejores resultados en el grid de XGBoost en comparación con los de Gradient Boosting solo realizaremos pruebas con XGBoost si disponieramos de más tiempo y mayor extensión de memoria probaríamos con todos los modelos desde árboles de regresión hasta este último.

En cuanto al diseño de la grilla realizaremos el mismo que el utilizado con la base de datos anterior, añadiendo para la evaluación de los hiperparámetros del modelo de predicción los hiperparámetros nombrados en el apartado de estrategias.

11.1. Resultados de la grilla

Vemos los primeros 12 mejores modelos de la grilla, ordenados por la métrica de rentabilidad promedio por operación (Directional_Error_Test):

Modelo	sizenode	lambda	alpha	nrounds	eta	mtry	Estrategia
1	0,05	5	0	20	0,1	0,25	IntervalosRatioCoste_4_0.1_1.5
2	0,05	5	0	20	0,1	0,25	IntervalosRatioCoste_4_0.05_1.5
3	0,05	5	0	20	0,1	0,25	IntervalosRatioCoste_4_0.1_2
4	0,05	5	0	20	0,1	0,25	IntervalosRatioCoste_4_0.03_1.5
5	0,05	5	0	20	0,1	0,25	IntervalosRatioCoste_4_0.1_3

6	0,05	5	0	20	0,1	0,25	IntervalosRatioCoste_4_0.05_2
7	0,05	5	0	20	0,1	0,25	IntervalosRatioCoste_4_0.05_3
8	0,05	5	0	20	0,1	0,25	IntervalosRatioCoste_4_0.1_1
9	0,05	5	0	20	0,1	0,25	IntervalosRatioCoste_4_0.01_1.5
10	0,05	5	0	20	0,1	0,25	IntervalosRatioCoste_4_0.05_1
11	0,05	5	0	20	0,1	0,25	IntervalosRatioCoste_4_0.03_2
12	0,05	5	0	20	0,1	0,25	IntervalosRatioCoste_4_0.03_3

Modelo	Percentage_Ops_Test	Direction_Train	Direction_Test	Directional_Error_Train	Directional_Error_Test
1	3,6222	0,5469	0,4880	0,1693	0,2006
2	3,6222	0,5437	0,5097	0,1551	0,1998
3	3,6222	0,5289	0,4664	0,1603	0,1982
4	3,6222	0,5416	0,5188	0,1570	0,1949
5	3,6222	0,5134	0,4424	0,1578	0,1945
6	3,6222	0,5285	0,4892	0,1476	0,1942
7	3,6222	0,5175	0,4652	0,1458	0,1905
8	3,6222	0,5764	0,5086	0,1801	0,1902
9	3,6222	0,5382	0,5405	0,1473	0,1896
10	3,6222	0,5707	0,5302	0,1654	0,1893
11	3,6222	0,5285	0,4983	0,1489	0,1893
12	3,6222	0,5193	0,4732	0,1466	0,1868

Modelo	BullA_Train	BeaA_Train	BullA_Test	BeaA_Test
1	0,9846	0,056	0,8188	0,1933
2	0,9846	0,056	0,8188	0,1933
3	0,9846	0,056	0,8188	0,1933
4	0,9846	0,056	0,8188	0,1933
5	0,9846	0,056	0,8188	0,1933
6	0,9846	0,056	0,8188	0,1933
7	0,9846	0,056	0,8188	0,1933
8	0,9846	0,056	0,8188	0,1933
9	0,9846	0,056	0,8188	0,1933
10	0,9846	0,056	0,8188	0,1933
11	0,9846	0,056	0,8188	0,1933
12	0,9846	0,056	0,8188	0,1933

Ilustración 31. Resultados grid XGBoost con nuevas variables y estrategias

A partir del grid realizado para el modelo XGBoost, se ha evaluado una serie de configuraciones identificadas por la nomenclatura IntervalosRatioCoste_R_A_S, donde:

- **R** representa la ratio de umbral por coste de operación, es decir, el umbral mínimo de rentabilidad esperado por operación en función de los costes (valor entero),
- **A** corresponde al nivel de significancia (alpha) del intervalo de confianza utilizado para la toma de decisiones,
- **S** es la ratio del stop-loss, indicando la proporción máxima de pérdida permitida antes de cerrar una operación.

El análisis de los resultados muestra que el Modelo 1, correspondiente a la estrategia IntervalosRatioCoste_4_0.1_1.5, ha obtenido el mejor desempeño en términos de rentabilidad promedio por operación (DE_Test), con un valor de 0.2006. Esto sugiere que una combinación de:

- Ratio de umbral por coste igual a 4.
- Nivel de significancia alpha = 0.1.

- Stop-loss con una ratio de 1.5.

es especialmente efectiva en las condiciones del conjunto de test evaluado.

Asimismo, se observa que estrategias con valores intermedios de alpha (0.05 y 0.1) y stop-loss moderados (entre 1.5 y 2) tienden a obtener los mejores resultados. Estas configuraciones parecen encontrar un buen equilibrio entre permitir movimientos favorables del precio y limitar las pérdidas, mientras que el umbral por coste elevado (R = 4) ayuda a filtrar operaciones poco rentables.

Los mejores resultados se concentran en los modelos que usan esta configuración base y varían únicamente en la estrategia de decisión (IntervalosRatioCoste). Esto refuerza la idea de que la lógica de entrada/salida basada en reglas económicas y estadísticas tiene más impacto que pequeños cambios en la arquitectura del modelo XGBoost en este caso concreto.

No obstante, es importante subrayar que estos resultados corresponden a una sola partición temporal de entrenamiento y test. Por ello, en etapas posteriores del trabajo se llevará a cabo una validación cruzada temporal, lo que permitirá analizar la estabilidad y robustez de cada estrategia a lo largo del tiempo. Este paso es esencial para garantizar que los modelos seleccionados no solo funcionan bien en un periodo concreto, sino que también son consistentes ante distintas condiciones de mercado.

12. Validación cruzada

12.1. Presentación de los modelos candidatos

En esta sección del estudio, se procederá a realizar una validación cruzada con estructura de serie temporal sobre un subconjunto seleccionado de modelos XGBoost. Concretamente, se analizarán:

1. Los 6 modelos con mejor rendimiento en test según la métrica Directional_Error_Test_Total, la cual representa la rentabilidad media por operación, teniendo en cuenta el porcentaje de operaciones que hace el modelo con la estrategia en un mes en media. Y los 2 mejores modelos, pero con tamaños de hoja del 1% y los del 0.5%.
2. Los 5 modelos con menor grado de sobreajuste, definidos como aquellos con menor diferencia absoluta entre Directional_Error_Train y Directional_Error_Test.

El objetivo es evaluar la variabilidad y el sesgo de la métrica Directional_Error_Test a través de diferentes particiones temporales, manteniendo la lógica temporal intacta (sin fugas de información futura).

Finalmente, con base en estos resultados, se seleccionará el modelo óptimo para la estrategia, considerando no solo su rendimiento puntual en test, sino también su robustez y consistencia a lo largo del tiempo.

Los modelos que analizaremos son los siguientes:

Resumen de Modelos - Mejores Hiperparámetros Grid

		Hiperparámetros del Modelo						Hiperparámetros Estrategia			Rendimiento y Sobreajuste			
ID	Modelo	Tamaño Hoja Proporción Total	Lambda Regulación	Alpha Regulación	Iteraciones	Eta	Proporción Variables	Ratio StopLose	Alpha Intervalo Confianza	ratio_coste	Sobreajuste	Mean(Rentabilidad) Op.	% Op.	Mean(Rentabilidad) Mensual
1	XGBoost	0.050	5	0	20	0.1	0.25	1.5	0.10	4.0	-0.03135242	0.20062213	3.622171	37.392890
2	XGBoost	0.050	5	0	20	0.1	0.25	1.5	0.05	4.0	-0.04476700	0.19981834	3.622171	36.701617
3	XGBoost	0.050	5	0	20	0.1	0.25	2.0	0.10	4.0	-0.03782776	0.19816250	3.622171	35.315103
4	XGBoost	0.050	5	0	20	0.1	0.25	1.5	0.03	4.0	-0.03788788	0.19490997	3.622171	32.732988
5	XGBoost	0.050	5	0	20	0.1	0.25	3.0	0.10	4.0	-0.03665400	0.19446305	3.622171	32.392286
6	XGBoost	0.050	5	0	20	0.1	0.25	2.0	0.05	4.0	-0.04662913	0.19418032	3.622171	32.178467
7	XGBoost	0.010	0	0	20	0.1	0.50	3.0	0.05	4.0	0.18055886	0.07139486	8.896415	2.876098
8	XGBoost	0.010	0	0	20	0.1	0.50	3.0	0.10	4.0	0.19493560	0.07004533	8.896415	2.561609
9	XGBoost	0.005	5	1	20	0.1	0.25	2.0	0.05	4.0	0.23101932	0.06815890	7.954733	1.789247
10	XGBoost	0.005	5	1	20	0.1	0.25	3.0	0.05	4.0	0.22097687	0.06712437	7.954733	1.632215
11	XGBoost	0.050	5	0	20	0.1	0.25	1.5	0.10	3.5	0.02654141	0.12981166	5.889642	24.154551
12	XGBoost	0.050	5	0	20	0.1	0.25	1.5	0.05	3.5	0.01678344	0.12751060	5.889642	21.998799
13	XGBoost	0.050	5	0	20	0.1	0.25	1.5	0.03	3.5	0.02281219	0.12339015	5.889642	18.580306
14	XGBoost	0.050	5	0	20	0.1	0.25	2.0	0.10	3.5	0.02550194	0.12303764	5.889642	18.312043
15	XGBoost	0.050	5	0	20	0.1	0.25	3.0	0.10	3.5	0.02466435	0.12173559	5.889642	17.351914

Ilustración 32. Mejores Modelos XGBoost + Estrategia Empleada

Estos son los modelos junto a las estrategias con los que aplicaremos esta validación cruzada. En cuanto a la mejor estrategia, vemos como la mejor que ha funcionado ha sido la que filtra tanto las operaciones que tienen un mínimo de certeza que vamos a superar el coste de operación, siendo este una ratio entre 3.5-4 veces mayor al coste por operación.

Hablando un poco de los modelos acerca del sobreajuste, los mejores modelos en cuanto a rentabilidad media por operación y rentabilidad promedio mensual se puede apreciar que tienen un sobreajuste negativo lo que quiere decir que funcionan mejor en el conjunto de datos de validación que en los de entrenamiento, esto se puede deber a puro azar, volatilidad del mercado o la misma tendencia o momento de esta. Con esto último me refiero a que un modelo puede aprender mejor para retrocesos, otros impulsos o movimientos laterales.

Para terminar de comentar los modelos antes de aplicar la validación cruzada, vemos como aquellos con un menor sobreajuste, tiene una rentabilidad promedio y mensual bastante aceptable. Si a través de la validación cruzada aseguramos resultados parecidos durante los distintos fold, podrían ser muy buenos candidatos como mejor modelo.

Centrándonos ahora un poco más en lo económico. En este mercado volátil, resulta fundamental valorar simultáneamente el nivel de ganancia promedio que puede ofrecer un modelo y su estabilidad. Un modelo con alta ganancia, pero alta varianza puede ser riesgoso y poco fiable en producción, mientras que uno con baja varianza, pero también baja ganancia puede no justificar su implementación operativa. De ahí la importancia de analizar el equilibrio entre retorno esperado y robustez.

Para elegir el mejor modelo, según un equilibrio de estas condiciones:

- Mayor ganancia media por operación
- Menor varianza, buscando aquellos más estables en los resultados de validación cruzada.
- No presente una correlación mayor a 0.1 ([-0.1,0.1]).

Todos los modelos serán comparados más a fondo para evaluar si existen diferencias estadísticamente significativas en sus resultados. Antes de realizar la validación cruzada vamos a profundizar más en la importancia de tener modelos poco correlacionados con la volatilidad del activo.

12.2. Relación entre Volatilidad y Rentabilidad de los modelos

La volatilidad se entiende como el grado de variación de una serie de precios financieros en el tiempo, usualmente estimado mediante la desviación estándar de los retornos. En otras

palabras, cuantifica la dispersión estadística de los cambios de precio alrededor de su media. Periodos de alta volatilidad suelen caracterizarse por movimientos bruscos e impredecibles, mientras que en entornos de baja volatilidad los precios tienden a fluctuar en rangos más estrechos.

Estos niveles de volatilidad condicionan los fenómenos de impulsos y retrocesos en las series financieras, ya que mercados muy volátiles pueden exhibir tendencias fuertes y reversiones abruptas a la vez, lo que plantea retos adicionales para los modelos predictivos.

12.3. Validación cruzada por "*forward chain cross-validation con ventanas fijas*"

Para evaluar de forma robusta la capacidad predictiva de los modelos sobre variables financieras no estacionarias, se implementó una validación cruzada tipo "*forward chain cross-validation con ventanas fijas"*. Esta técnica respeta la estructura temporal de los datos, evitando la filtración de información futura o pasada en el entrenamiento una condición esencial en el estudio de patrones como impulsos y retrocesos, donde cualquier anticipo podría interferir de manera irreal los resultados del modelo.

Además, al mantener un tamaño constante en los bloques de validación, se logra una medición más estable y comparable de la rentabilidad media esperada por cada modelo. Este enfoque permite simular con precisión escenarios de inversión recurrentes en el tiempo, ayudando a determinar si un modelo no solo ajusta bien en el pasado, sino si también mantiene una capacidad de generalización útil para predecir futuros movimientos del mercado. Haremos esta validación cruzada con un fold de 6 validaciones. Los resultados son los siguientes:

Resultados de Validación Cruzada - Errores por Modelo

	Rentabilidades de Validación Cruzada (Folds) %					
Modelo	**Rent. Op. Fold 1**	**Rent. Op. Fold 2**	**Rent. Op. Fold 3**	**Rent. Op. Fold 4**	**Rent. Op. Fold 5**	**Rent. Op. Fold 6**
modelo_1	0.3713647	0.3965980	0.4114225	0.2841328	0.3391916	0.3438570
modelo_2	0.3684211	0.3628731	0.4016064	0.3168686	0.3419048	0.3370647
modelo_3	0.3410616	0.3396657	0.3674274	0.3305085	0.3111014	0.3338221
modelo_4	0.3639576	0.3753724	0.4101812	0.3176856	0.3249211	0.3532958
modelo_5	0.3257919	0.3227053	0.3434370	0.2818455	0.2980922	0.2972108
modelo_6	0.3421235	0.3293310	0.3744518	0.3193833	0.3074236	0.3179916
modelo_7	0.3523230	0.3358209	0.4052632	0.3254663	0.3104991	0.3219979
modelo_8	0.3359629	0.3604288	0.3768421	0.3252357	0.3150183	0.3063849
modelo_9	0.3774537	0.3890637	0.4114693	0.3733493	0.3326377	0.3435667
modelo_10	0.3488253	0.3527009	0.3917743	0.3447109	0.3109349	0.2987165
modelo_11	0.3583781	0.3307332	0.4040137	0.3251613	0.3333333	0.3552730
modelo_12	0.3734015	0.3381759	0.3994130	0.3335244	0.3275544	0.3350785
modelo_13	0.3688525	0.3413174	0.4077004	0.3021886	0.3239741	0.3322005
modelo_14	0.3144192	0.3068027	0.3735799	0.3079151	0.3140054	0.3250395
modelo_15	0.3150525	0.3161486	0.3403127	0.2777778	0.2948131	0.3016074

Ilustración 33. Resultados Validación Cruzada

Antes de estudiar el sesgo y la variabilidad de las métricas, se procede a filtrar aquellos modelos cuya correlación con la volatilidad del mercado sea baja, estableciendo un umbral absoluto de 0.1 para esta correlación. Esta decisión responde a la necesidad de identificar modelos robustos frente a cambios estructurales en el mercado. En contextos financieros, especialmente en activos como Bitcoin, la volatilidad puede variar drásticamente a lo largo del tiempo. Si un modelo muestra una fuerte correlación entre sus métricas de rendimiento (como rentabilidad promedio o porcentaje de operaciones) y la volatilidad, esto sugiere que su desempeño está condicionado a un entorno específico de mercado, por ejemplo, solo funciona bien cuando hay alta volatilidad, y podría deteriorarse rápidamente bajo otros regímenes.

Para evaluar esta dependencia, se calcula la correlación de Pearson entre la volatilidad media en cada fold de validación cruzada y las métricas clave obtenidas por cada modelo. Una correlación cercana a cero indica que el modelo mantiene un comportamiento constante ante distintos escenarios de volatilidad, lo cual es deseable en modelos destinados a la toma de decisiones automáticas. En cambio, modelos con correlaciones mayores a ±0.1 podrían estar sobreajustados a ciertos niveles de volatilidad, reduciendo su capacidad de generalizar y

aumentando el riesgo operativo si el régimen de mercado cambia.

Por tanto, este análisis no busca únicamente maximizar la rentabilidad esperada, sino también seleccionar modelos que sean estables y resilientes a la incertidumbre del mercado, favoreciendo así su implementación práctica en entornos reales de trading.

Vemos esta relación de volatilidad con estas métricas de manera gráfica y con el coeficiente de correlación:

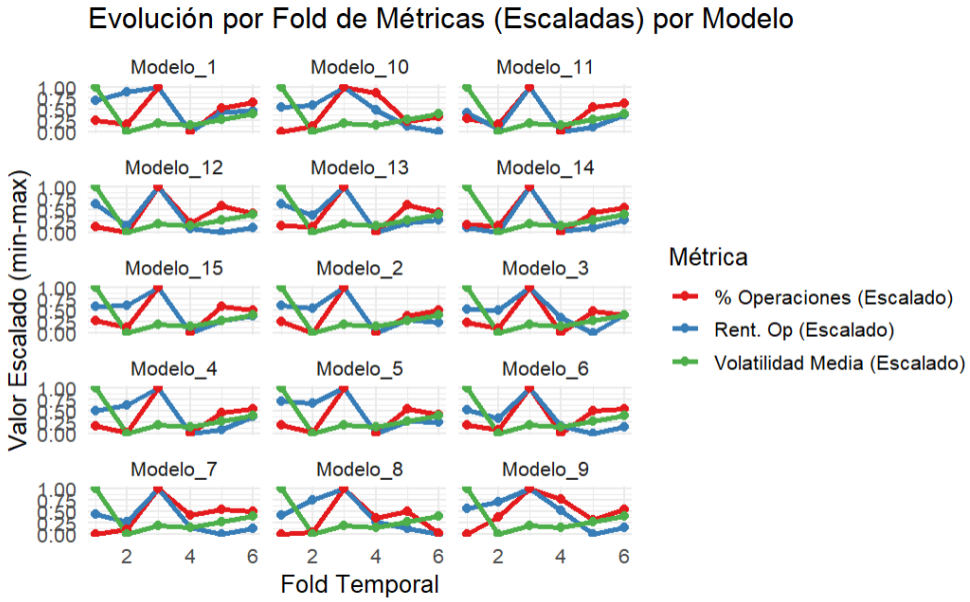


Ilustración 34. Evolución conjunta Volatilidad y Métricas de Validación Cruzada

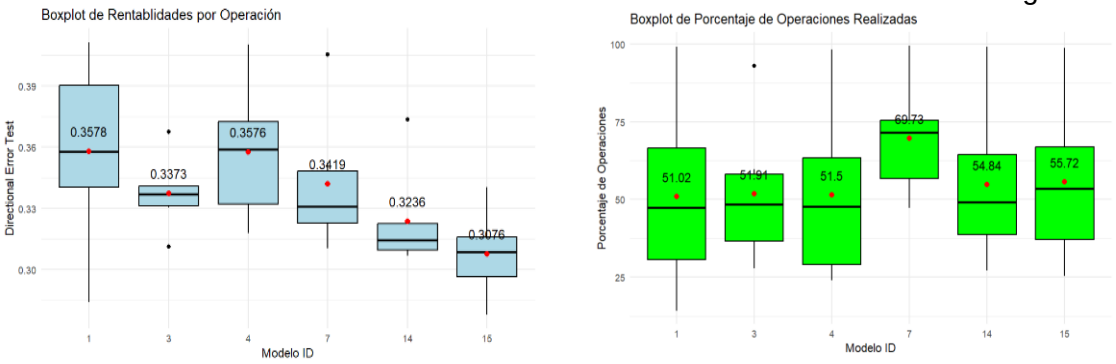
Correlaciones por Modelo

	Volatilidad - Métricas Validación Cruzada	
Modelo	**Correlación Rentabilidad Promedio**	**Correlación % Operaciones**
Modelo_12	0.318256731	-0.16247624
Modelo_13	0.256233485	-0.13736319
Modelo_11	0.201795034	0.01035414
Modelo_5	0.168796934	-0.03428028
Modelo_2	0.126995066	0.04432589
Modelo_6	0.109424179	-0.06416324
Modelo_15	0.093478399	-0.01994982
Modelo_7	0.062502358	-0.40800542
Modelo_1	0.033322918	-0.04753512
Modelo_3	0.002856963	-0.04378768
Modelo_4	0.001784407	-0.04449764
Modelo_14	-0.089130253	-0.12032150
Modelo_10	-0.127328792	-0.48104070
Modelo_9	-0.142739733	-0.64121085
Modelo_8	-0.261626888	-0.37591953

Ilustración 35. Correlaciones Volatilidad - Métricas de Validación Cruzada

Una vez identificados los modelos que cumplen con este umbral de correlación de 0.1, se procederá a analizar de manera específica sus niveles de sesgo y varianza. Este análisis permitirá valorar en qué medida dichos modelos, a pesar de su baja sensibilidad a la volatilidad, presentan estabilidad en sus predicciones (baja varianza) y adecuación al fenómeno subyacente (bajo sesgo). Así, se busca optimizar el equilibrio entre robustez frente a condiciones de mercado cambiantes y capacidad predictiva general.

Las métricas de validación cruzada de los modelos seleccionados serían los siguientes:



Resumen de Modelos - Resultados de Validación Cruzada

	Estadísticas del Modelo			
Modelo	**Rentabilidad Promedio Op**	**Sd Rentabilidad Op**	**Media % Operaciones**	**Sd % Operación**
1	0.3577611	0.04589594	51.01711	30.92564
4	0.3575690	0.03406069	51.49729	28.33569
7	0.3418951	0.03409592	69.72673	18.65196
3	0.3372644	0.01828656	51.90763	23.29562
14	0.3236270	0.02531628	54.84110	26.02668
15	0.3076187	0.02138934	55.71853	26.62802

Ilustración 36. Resultados Validación Cruzada Modelos no sensibles a la volatilidad

Para ayudar a la toma de decisiones nos apoyaremos de contrastes de igualdad de medias, para ver si hay diferencias significativas entre cada uno de los modelos.

$$H_0: Igualdad\ de\ medias\ H_1: Medias\ distintas$$

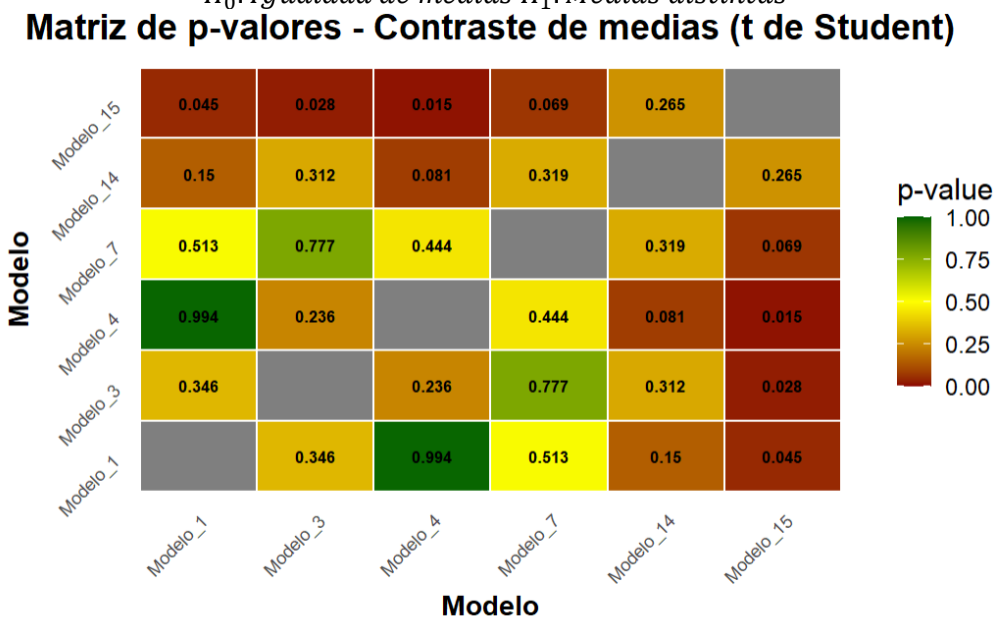


Ilustración 37. Matriz p-valores contraste de diferencia de medias entre modelos candidatos

12.4. Elección del mejor modelo

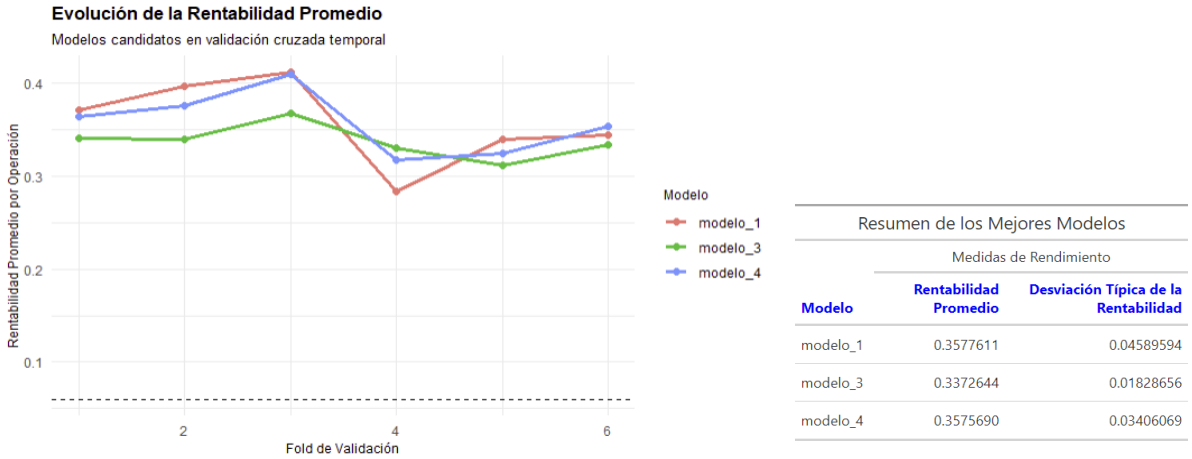
Fijándonos en primer lugar en las correlaciones, vemos como los mejores modelos que muestran una mayor independencia de la volatilidad, acercándose ambas correlaciones, tanto con la rentabilidad promedio por operación como con el porcentaje de operaciones que se activan son

el modelo_1, modelo_3 y modelo_4, lo que indican estos modelos que tienen una baja sensibilidad a los cambios en la volatilidad del mercado.

Centrándonos en las métricas de validación cruzada, valoran el sesgo y varianza a partir de la rentabilidad media por operación y su desviación típica. El modelo 1 destaca como el que presenta la mayor rentabilidad media (0.358), aunque con una desviación típica relativamente alta (0.0459), lo cual podría indicar cierta inestabilidad. En cambio, el modelo 3, si bien presenta una rentabilidad ligeramente inferior (0.337), se caracteriza por tener la menor desviación típica entre todos los candidatos (0.0183), lo que sugiere una mayor consistencia en sus resultados. Finalmente, el modelo 4 presenta un comportamiento intermedio, con la misma rentabilidad media que el modelo 1 (0.358), pero con una menor varianza (desviación típica de 0.0341), lo que lo convierte en un modelo equilibrado entre sesgo y varianza.

Además, se realizó una comparación estadística de medias entre estos modelos. Con un nivel de significación del 5%, no se encontró evidencia suficiente para rechazar la hipótesis nula de igualdad de medias, lo cual sugiere que no existen diferencias estadísticamente significativas en la rentabilidad media entre los modelos seleccionados.

A la luz de estos resultados, se concluye que los modelos 1, 3 y 4 constituyen las mejores alternativas para avanzar al análisis con la evolución de la rentabilidad promedio para diferentes fold.



Resumen de los Mejores Modelos

	Medidas de Rendimiento	
Modelo	Rentabilidad Promedio	Desviación Típica de la Rentabilidad
modelo_1	0.3577611	0.04589594
modelo_3	0.3372644	0.01828656
modelo_4	0.3575690	0.03406069

Ilustración 38. Evolución Rentabilidad promedio por fold modelos ganadores y tabla resumen

A pesar de no encontrarse diferencias estadísticamente significativas entre los tres modelos candidatos seleccionados (Modelo_1, Modelo_3, y Modelo_4), la inspección visual de la evolución de la rentabilidad por operación en la validación cruzada temporal sugiere que el Modelo_3 presenta una mayor estabilidad en sus resultados. En particular, destaca por mantener un rendimiento consistente y superior al umbral mínimo aceptable del 0.06%, donde las condiciones del mercado pueden cambiar rápidamente. Esta capacidad de adaptación y regularidad en escenarios recientes refuerza la robustez del modelo y su idoneidad para ser utilizado como base en un entorno de producción, frente a otros modelos que, aunque competitivos, presentan una mayor variabilidad o caídas puntuales en su rendimiento.

La Tabla anterior resume el rendimiento de los tres modelos seleccionados según sus estadísticas descriptivas clave. El Modelo_3 destaca por presentar tanto una de las medias más altas de rentabilidad como una desviación típica menor a todos candidatos, lo que sugiere una combinación óptima entre rentabilidad y estabilidad. Además, mantiene resultados superiores al umbral mínimo del 0.06% en todos los folds, lo que refuerza su idoneidad para ser implementado en entornos reales con datos más recientes y cambiantes.

13. Evaluación del Modelo Ganador

En este apartado se procede a la evaluación detallada del modelo seleccionado como ganador tras el proceso de validación cruzada. El objetivo principal es analizar en profundidad tanto

el comportamiento como el rendimiento del modelo final de XGBoost, en un contexto de predicción financiera.

En primer lugar, se presentan los hiperparámetros óptimos identificados durante el proceso de ajuste, así como la estrategia de operación empleada. Este análisis permite comprender las decisiones de configuración que han llevado al mejor desempeño del modelo.

A continuación, se examina la importancia relativa de las variables utilizadas en el entrenamiento. Este análisis resulta clave para identificar qué indicadores o características del mercado han sido más determinantes para el modelo en su capacidad predictiva, contribuyendo así a la interpretabilidad del sistema.

También se incorporan visualizaciones que ilustran cómo evoluciona el error a lo largo de las iteraciones del algoritmo de boosting, tanto en el conjunto de entrenamiento como en el conjunto de prueba. Este análisis gráfico permite entender cómo XGBoost mejora progresivamente sus predicciones y hasta qué punto es capaz de generalizar sin sobreajustarse a los datos.

Finalmente, se evalúa el comportamiento del modelo en un escenario simulado de operativa real utilizando los datos de prueba. Se representa gráficamente la evolución del precio del activo durante el periodo de prueba, comparándolo con la evolución teórica que habría seguido una estrategia guiada por las predicciones del modelo. Además, se contempla una tercera trayectoria que considera esta misma estrategia, pero descontando el coste medio por operación. Este enfoque permite valorar la viabilidad económica de aplicar el modelo en un entorno real de mercado, considerando las comisiones y otros costes asociados a la operativa.

13.1. Hiperparámetros del modelo ganador

Los hiperparámetros de nuestro modelo ganador son los siguientes:

Resumen de Modelos - Mejores Hiperparámetros Grid

		Hiperparámetros del Modelo						Hiperparámetros Estrategia			Rendimiento y Sobreajuste			
ID	Modelo	Tamaño Hoja Proporción Total	Lambda Regulación	Alpha Regulación	Iteraciones	Eta	Proporción Variables	Ratio StopLose	Alpha Intervalo Confianza	ratio_coste	Sobreajuste	Mean(Rentabilidad) %Op.	% Op.	Mean(Rentabilidad) %Mensual
3	XGBoost	0.05	5	0	20	0.1	0.25	2	0.1	4	-0.03782776	0.1981625	3.622171	35.3151

Ilustración 39. Hiperparámetros Modelo Ganador

13.1.1. Hiperparámetros del Modelo XGBoost

Parámetro	Valor	Interpretación
sizenode	0.05	Profundidad del árbol restringida (equivalente a subsample bajo), que favorece la regularización. Ideal para evitar sobreajuste en series temporales volátiles.
lambda	5	Penalización L2 relativamente fuerte: promueve pesos más pequeños en los nodos, lo que evita la dependencia excesiva de variables ruidosas.
alpha	0	No se aplica penalización L1. Indica que no se fuerza la selección de variables de forma espartana; útil si se asume que todas las variables pueden aportar algo.
nrounds	20	Número moderado de iteraciones: se evita sobre entrenamiento manteniendo el modelo simple y generalizable.
eta	0.1	Tasa de aprendizaje conservadora, adecuada para converger lentamente y evitar grandes ajustes en cada iteración.
mtry	0.25	Proporción de variables seleccionadas aleatoriamente en cada división. Aporta diversidad y reduce la correlación entre árboles, reforzando la robustez.

Ilustración 40. Interpretación Hiperparámetros Modelo Ganador

En conjunto, este modelo tiene un enfoque parsimonioso, diseñado para controlar el sobreajuste y ser robusto frente a la alta varianza del mercado. La regularización L2 y el muestreo parcial promueven generalización sin penalizar severamente la cantidad de información utilizada.

13.1.2. Hiperparámetros de la estrategia de Trading

Parámetro	Valor	Interpretación
ratio	2	Define un *stop-loss* que permite que el precio se desvíe hasta el doble del coste por operación antes de cerrarse. Da margen al mercado para oscilar antes de aceptar una pérdida.
alpha.interval	0.1	Intervalo de confianza del 10%. Se cierra la operación si el precio entra en esa banda. Es una salida anticipada por incertidumbre, incluso antes del stop-loss. Muy conservador.
ratio_coste	4	Umbral de entrada: la predicción debe superar 4 veces el coste de operar para abrir una operación. Establece una política de entrada muy exigente, que prioriza calidad sobre cantidad.

Ilustración 41. Interpretación hiperparámetros estrategia del mejor modelo

La estrategia está pensada para actuar solo cuando la señal es extremadamente fuerte, minimizando operaciones innecesarias y priorizando eficiencia sobre frecuencia. Este diseño es ideal para detectar impulsos y retrocesos claros, ya que filtra las micro oscilaciones de baja calidad. El intervalo de confianza estrecho (alpha.interval = 0.1) actúa como un trigger técnico de salida basada en la propia predicción del modelo, no en el precio por sí mismo, lo que introduce un enfoque de gestión probabilística del riesgo.

13.1.3. Evaluaciones métricas

La métrica de Directional Error Test = 0.198 lo que nos muestra que nuestra estrategia de inversión automatizada tiene una rentabilidad promedia por operación de casi un 0.2% por encima de los umbrales de coste por operación de los exchange que suelen ir desde un 0.03% hasta un 0.07%. Siendo por tanto una posible estrategia de inversión rentable. Ahora veremos que variables son las que mas influyen en este modelo y terminaremos viendo como se comporta nuestra estrategia en diferentes momentos del mercado.

13.2. Explicabilidad del modelo ganador

Una vez identificado el modelo ganador en términos de rendimiento predictivo y comportamiento estratégico, es fundamental comprender cómo este modelo toma decisiones y qué variables han sido más relevantes a la hora de predecir impulsos y retrocesos en la serie temporal. En este apartado, abordamos el análisis de explicabilidad del modelo XGBoost seleccionado, con el objetivo de interpretar qué factores han tenido mayor peso en sus predicciones.

La explicabilidad se ha convertido en una herramienta clave en entornos financieros, especialmente cuando se utilizan modelos de tipo caja negra como los algoritmos de boosting. A través del análisis de importancia de variables, buscamos no solo validar la coherencia del modelo con el comportamiento esperado del mercado, sino también identificar posibles sesgos o dependencias excesivas en determinadas características del conjunto de datos.

Para ello, se emplean técnicas específicas para modelos de árbol como la ganancia total (gain), la frecuencia de aparición en divisiones (frequency) o el cover (cobertura), que permiten cuantificar el aporte individual de cada predictor al desempeño general del modelo. Este enfoque nos permitirá responder a preguntas clave como:

- ¿Qué indicadores técnicos o macroeconómicos han tenido mayor impacto en las decisiones del modelo?
- ¿El modelo se apoya en variables estables o en señales de corto plazo?
- ¿Existen patrones de comportamiento común antes de los impulsos o retrocesos que el modelo haya aprendido?

A través de esta sección, se refuerza el componente interpretativo del trabajo, aportando una visión más transparente del funcionamiento del modelo y una guía práctica para futuros ajustes

o extensiones del sistema predictivo.

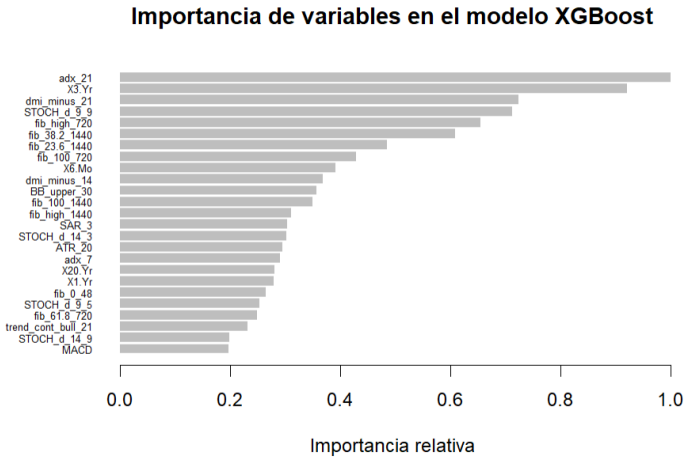


Ilustración 42. Variables más importantes en el modelo

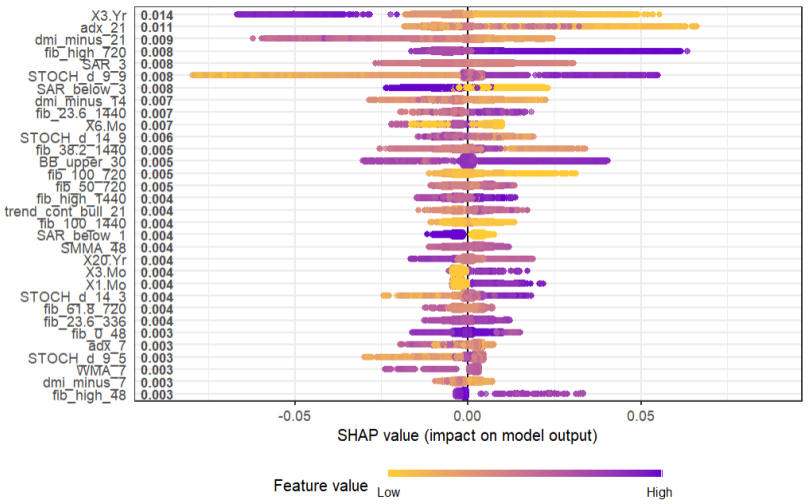


Ilustración 43. Importancia de cada variable a la predicción de cada observación XGBoost (SHAP value)

Para evaluar la relevancia de las variables predictoras utilizadas en el modelo XGBoost, se calcularon dos métricas complementarias: la importancia relativa basada en la frecuencia de uso en los árboles del modelo y los valores SHAP (SHapley Additive exPlanations) (Lundberg, S. M., & Lee, S.-I, 2017), que permiten interpretar el impacto de cada variable en las predicciones individuales del modelo.

En la Figura se muestran las 25 variables con mayor impacto en las predicciones del modelo XGBoost según los valores SHAP. Este gráfico no solo permite identificar qué variables son más relevantes, sino también entender cómo su valor (alto o bajo) influye en la salida del modelo. En este caso, la variable objetivo es la predicción de impulso o retroceso en el precio de Bitcoin.

La variable X3_Yr destaca como la más importante del modelo. Según el gráfico, cuando toma valores elevados (en color morado), el impacto SHAP tiende a ser negativo, lo que indica que valores altos de esta variable están asociados con menores predicciones de rentabilidad futura. Por el contrario, valores bajos (amarillo) tienden a estar asociados con predicciones más positivas, sugiriendo un posible efecto de advertencia o saturación cuando los tipos de interés a 3 años alcanzan niveles altos.

También destaca adx_21, un indicador de fuerza de tendencia en el mercado. En este caso, observamos que tanto valores muy altos como muy bajos pueden tener impactos negativos, lo que puede indicar que el modelo penaliza condiciones de extrema fuerza direccional (quizá por miedo a una corrección). Este comportamiento sugiere que los valores intermedios del ADX podrían ser más estables o "seguros" para operar.

Otra variable clave es dmi_minus_21, asociada con la presión bajista. Aquí se observa una

simetría inversa: valores altos (morados) contribuyen negativamente a la predicción, lo que concuerda con un escenario de retroceso o caída del mercado. Esto indica que cuando la presión bajista es elevada, el modelo predice con mayor probabilidad un movimiento negativo.

En cuanto a indicadores de tipo técnico como el STOCH_k_9_9 o el SAR_3, vemos que sus valores más extremos también tienden a producir impactos significativos, lo que refuerza su utilidad como disparadores técnicos en momentos clave del mercado.

En conjunto, los resultados sugieren que el modelo XGBoost da un mayor peso a señales de fuerza de tendencia, continuidad direccional y niveles técnicos clave en horizontes temporales amplios (como retrocesos de Fibonacci a 720 períodos), en lugar de basarse en señales rápidas o de corto plazo. Esto refuerza la hipótesis de que las condiciones de fondo (como la tendencia subyacente y la estabilidad macroeconómica) desempeñan un papel determinante en la formación tanto de impulsos como de retrocesos.

En general, la interpretación de estas variables no solo permite entender cómo el modelo está operando, sino también detectar qué condiciones el modelo ha aprendido a asociar con oportunidades de impulso o retroceso. Este análisis ayuda a validar que el modelo está incorporando señales coherentes con el comportamiento real del mercado, lo cual es esencial para justificar su uso en entornos reales de inversión.

13.3. Rendimiento en diferentes tendencias

Para evaluar el rendimiento de tu modelo de trading basado en XGBoost en diferentes condiciones de mercado, es esencial seleccionar periodos representativos de tendencias alcistas, bajistas y laterales. A continuación, se identifican tres tramos del precio de Bitcoin en temporalidad de 1 hora entre 2022 y diciembre de 2024, basados en datos históricos (Official Data Foundation, 2024):

- **Tendencia Bajista: Enero a diciembre de 2022**

Durante 2022, Bitcoin experimentó una caída significativa, pasando de aproximadamente $47,000 en marzo a menos de $20,000 a finales de año. Este periodo refleja una clara tendencia bajista, influenciada por factores macroeconómicos y eventos en el mercado de criptomonedas.

- **Tendencia Alcista: Enero a diciembre de 2023**

En 2023, Bitcoin mostró una recuperación notable, comenzando el año en torno a $16,530 y cerrando cerca de $42,258. Este incremento sostenido indica una tendencia alcista, posiblemente impulsada por un renovado interés institucional y mejoras en el entorno regulatorio.

- **Tendencia Lateral: Enero a junio de 2024**

Durante la primera mitad de 2024, el precio de Bitcoin fluctuó entre $60,000 y $70,000, sin una dirección clara. Este comportamiento sugiere una fase de consolidación o tendencia lateral, donde el mercado carece de un impulso definido hacia arriba o abajo.

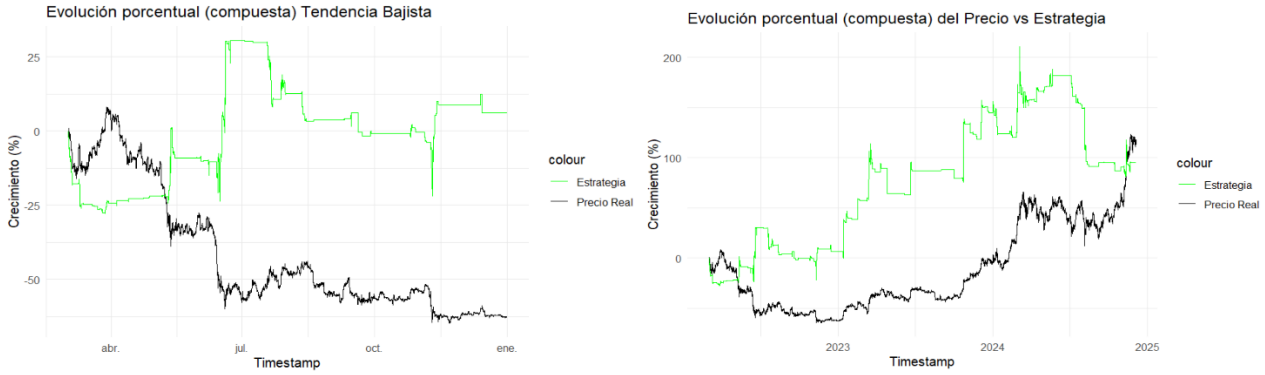


Ilustración 44. Rentabilidad Estrategia vs Rentabilidad precio activo %. Diferentes momentos del mercado

En la Figura anterior se presentan los resultados del rendimiento de la estrategia propuesta frente a la evolución real del precio de Bitcoin en tres contextos de mercado claramente diferenciados: tendencia bajista (2022-2023), tendencia alcista (2023) y movimiento lateral (2024). A pesar de que el modelo fue capaz de identificar ciertas oportunidades de operación, especialmente durante la fase bajista, el rendimiento general en los tres escenarios es insuficiente en comparación con la evolución del precio subyacente. En particular:

En la tendencia bajista, la estrategia logra capturar algunos movimientos de impulso a favor de la tendencia dominante, mostrando un rendimiento superior al de la evolución real del activo en determinados tramos. Sin embargo, se observa una alta concentración de operaciones estáticas o ineficaces en otras zonas del gráfico, lo que limita su crecimiento sostenido.

Durante la fase alcista, la estrategia no consigue adaptarse al cambio de régimen, permaneciendo con decisiones de bajo rendimiento e incapaz de seguir el crecimiento exponencial del activo. Esto sugiere una limitada capacidad de respuesta del modelo ante dinámicas de mercado que se desarrollan con velocidad.

En el contexto lateral, el modelo muestra un comportamiento errático, alternando entre operaciones perdedoras y periodos sin actividad operativa. Esto es consistente con las dificultades típicas de operar en rangos laterales, donde los patrones de impulso y retroceso son más difíciles de modelar.

Una posible explicación de estos resultados subóptimos radica en la naturaleza de las variables explicativas seleccionadas, las cuales, según el análisis de interpretabilidad del modelo, son estables en el tiempo y, por tanto, poco sensibles a cambios abruptos del mercado. Este tipo de variables puede aportar valor en escenarios estacionarios, pero dificultan el aprendizaje del modelo en horizontes de corto plazo, donde los patrones no lineales y el cambio de régimen son frecuentes.

En consecuencia, se hace evidente la necesidad de incorporar nuevas fuentes de información que reflejen mejor la estructura dinámica del mercado, tales como métricas on-chain (King, J. C., Dale, R., & Amigó, J. M. ,2024), indicadores de sentimiento, liquidez o comportamiento de los participantes institucionales, que pueden aportar contexto más profundo a los movimientos de precios.

Asimismo, este análisis pone de manifiesto la importancia de contar con una base de datos más robusta y rica, así como con un presupuesto computacional adecuado que permita experimentar con modelos más complejos, capaces de capturar relaciones no lineales y adaptarse a múltiples regímenes de mercado. A pesar de los resultados actuales, se ha logrado establecer una estructura sólida de validación y análisis, lo cual constituye una base prometedora para avanzar en el desarrollo de estrategias cuantitativas más eficientes en el futuro.

14. Limitaciones en la investigación

Los resultados obtenidos en este trabajo, aunque metodológicamente sólidos, reflejan las

limitaciones inherentes a un entorno de investigación con recursos restringidos. La falta de presupuesto, equipo y tiempo ha condicionado la capacidad para desarrollar un bot de trading que alcance rentabilidades significativas en diferentes condiciones de mercado.

En contraste, las grandes empresas de trading algorítmico invierten sumas multimillonarias en infraestructuras avanzadas para optimizar sus operaciones. Por ejemplo, XTX Markets ha destinado 1.000 millones de euros a un centro de datos de alto rendimiento en Finlandia, con el objetivo de reducir la latencia de ejecución y mejorar la eficiencia del mercado mediante el uso de inteligencia artificial y aprendizaje automático (Bloomberg News, 2024, 21 de marzo).

Además, gigantes tecnológicos como Amazon, Microsoft y Google han anunciado inversiones que superan los 21.000 millones de dólares en centros de datos en España, impulsadas por la creciente demanda de capacidades de computación para inteligencia artificial y servicios en la nube. Estas infraestructuras permiten a las grandes firmas procesar y analizar enormes volúmenes de datos en tiempo real, una capacidad que está fuera del alcance de proyectos con recursos limitados (Expansión, 2024, 29 de enero).

Esta disparidad en recursos subraya la importancia de contar con una infraestructura adecuada para desarrollar estrategias de trading algorítmico efectivas. Aunque los resultados de este TFG no alcanzan las rentabilidades deseadas, establecen una base sólida para futuras investigaciones que, con mayores recursos, podrían explorar modelos más complejos y bases de datos más robustas.

15. Conclusiones y próximas líneas de investigación

15.1. Conclusiones

Este trabajo ha explorado la construcción de un sistema de trading algorítmico basado en técnicas de aprendizaje automático para la predicción de impulsos y retrocesos en el mercado de Bitcoin, utilizando modelos desde árboles de regresión y random forest hasta modelos más complejos como XGBoost junto a una estrategia de inversión heurística. A lo largo del estudio se ha comprobado que, si bien los modelos implementados han demostrado capacidad para adaptarse a ciertos patrones del mercado, los resultados de rentabilidad no han sido los esperados en todos los contextos.

Una de las principales conclusiones radica en que las variables explicativas más relevantes identificadas a través de técnicas de interpretabilidad como SHAP resultaron ser muy estables en el tiempo. Aunque esto puede ser beneficioso para capturar relaciones estructurales del mercado en horizontes largos, limita severamente la capacidad del modelo para adaptarse a dinámicas rápidas o cambios de régimen, algo especialmente frecuente en mercados financieros como el de criptomonedas.

Este hallazgo recalca la necesidad de incorporar variables más dinámicas, como métricas on-chain (por ejemplo, el número de direcciones activas, volumen en exchanges, o el MVRV ratio), que reflejen mejor el comportamiento de los participantes en el ecosistema de Bitcoin. Estas métricas pueden extraerse de fuentes como Glassnode, CryptoQuant o IntoTheBlock, y tienen un potencial notable para mejorar la contextualización del precio y anticipar movimientos extremos, aportando valor añadido a los modelos predictivos en contextos no lineales y altamente volátiles.

Desde el punto de vista metodológico, también se identificaron limitaciones derivadas de la complejidad del problema: la predicción de movimientos financieros cortoplacistas no sólo requiere modelos más complejos, sino también procesos de validación más robustos. En este sentido, se destaca la necesidad de adoptar técnicas específicas para series temporales financieras, como la validación cruzada con ventanas fijas hacia adelante (forward chaining), que respeta la estructura temporal de los datos y evita fugas de información.

Asimismo, la estandarización y la transformación de las variables (por ejemplo, mediante diferenciación para lograr estacionariedad) siguen siendo pasos cruciales en la mejora de la calidad predictiva del modelo, sobre todo al trabajar con series altamente no estacionarias como los precios

financieros.

A pesar de no haber obtenido una estrategia de inversión rentable en todos los tramos de mercado analizados, el trabajo sienta las bases para un marco sólido de validación y análisis del rendimiento, que puede ser escalado con mayor presupuesto, más datos y tecnologías más avanzadas. De hecho, en el sector financiero actual, las empresas que logran beneficios sostenibles mediante trading algorítmico invierten millones de euros en infraestructura, como centros de datos de baja latencia, servidores colocados físicamente cerca de bolsas y acceso prioritario a datos de mercado. Un ejemplo es la empresa Jump Trading, que ha realizado fuertes inversiones en infraestructura tecnológica para obtener ventajas competitivas, tal como se recoge en el artículo de Bloomberg (Bloomberg News**.**, 2014, 23 de julio).

Este contraste deja claro que, para construir modelos realmente competitivos, es imprescindible contar con un equipo multidisciplinar, acceso a fuentes de datos avanzadas y una financiación adecuada. La falta de presupuesto, tiempo y recursos limita el alcance de un proyecto académico, pero no invalida el potencial de seguir construyendo una arquitectura predictiva robusta.

Además, es importante destacar que esta investigación se ha articulado en torno a dos hipótesis fundamentales. En primer lugar, se partió de la premisa (H1) de que modelos avanzados como Gradient Boosting y XGBoost ofrecerían una mayor capacidad predictiva frente a alternativas más sencillas como los árboles de decisión o técnicas de Bagging. Aunque los resultados obtenidos en esta investigación no han logrado validar completamente esta hipótesis en términos de rentabilidad constante, sí se ha observado una mayor capacidad de estos modelos para adaptarse a la complejidad de los datos financieros y capturar relaciones no lineales, en especial cuando se evalúan mediante técnicas de validación cruzada adecuadas al contexto temporal, como el método de ventanas fijas.

La segunda hipótesis (H2), referida a una mayor rentabilidad y tasa de aciertos asociada al uso de estrategias más complejos, ha quedado parcialmente apoyada: si bien los modelos más sofisticados mostraron mayor precisión en ciertos tramos del mercado, los resultados generales ponen de relieve que la rentabilidad no depende únicamente del modelo, sino de factores clave como la calidad de las variables explicativas y la estructura de la estrategia de inversión

15.2. Futuras líneas de investigación – Largo plazo

Para continuar mejorando este sistema, se proponen las siguientes líneas de investigación o mejora de la estructura y bases de datos:

- **Integración de modelos más complejos:** como redes neuronales recurrentes (RNN, LSTM) o modelos de deep learning con atención, que pueden capturar dependencias temporales de largo plazo y no linealidades difíciles de modelar con algoritmos tradicionales.
- **Ensamblado de modelos:** combinar modelos complementarios, como ensamblados de XGBoost con redes neuronales o modelos especializados en predecir duración, dirección o magnitud de los impulsos.
- **Inclusión de nuevas fuentes de datos**: incorporar métricas on-chain, datos de sentimiento extraídos de redes sociales, volumen y profundidad de mercado (order book), volatilidad implícita, y otros factores externos como el ciclo de halving de Bitcoin, que históricamente ha tenido un fuerte impacto en su precio
- **Modelos generativos y de detección de régimen:** explorar modelos que detecten cambios estructurales en el mercado para adaptar la estrategia automáticamente según el contexto (por ejemplo, cambio entre mercado alcista, bajista o lateral).
- **Backtesting realista y ejecución de órdenes:** desarrollar simulaciones que tengan en cuenta el deslizamiento, el impacto en el mercado y el tiempo de ejecución real, para evaluar la viabilidad del sistema de forma más fidedigna.

En conclusión, aunque los resultados actuales no han alcanzado niveles de rentabilidad

consistentes, este trabajo ha contribuido significativamente a entender las complejidades del trading algorítmico, y ofrece una hoja de ruta sólida para continuar la investigación y mejorar el rendimiento futuro. La clave estará en la combinación de técnicas avanzadas, fuentes de datos representativas y una infraestructura adecuada.

15.3. Futuras líneas de investigación – Corto plazo - Modelos fraccionados según el régimen de mercado y combinación con modelos GARCH

Una de las líneas de investigación más prometedoras para mejorar la capacidad predictiva de los modelos y la rentabilidad de las estrategias consiste en la creación de modelos fraccionados adaptados a diferentes regímenes de mercado. El análisis de los resultados obtenidos a lo largo de este trabajo sugiere que los patrones de comportamiento del precio no son homogéneos en todos los contextos; los impulsos y retrocesos tienden a comportarse de manera distinta dependiendo de si el mercado se encuentra en una fase alcista, bajista o lateral. Por ello, se propone como futura extensión la construcción de tres modelos especializados, cada uno entrenado exclusivamente con datos pertenecientes a uno de estos contextos. Esta segmentación permitiría a cada modelo aprender patrones específicos de su régimen, mejorando la sensibilidad y precisión de las predicciones en tiempo real.

Para implementar esta estrategia, sería necesario diseñar un sistema de detección automática del régimen de mercado, que clasifique de forma dinámica cada observación en una de las tres categorías. Una vez identificado el estado del mercado, el sistema aplicaría el modelo específico correspondiente, ajustado y entrenado únicamente para ese tipo de entorno. Esta aproximación ofrece una mayor capacidad de adaptación y permite incorporar de forma más efectiva las diferencias estructurales que caracterizan a cada fase del ciclo de precios.

Complementariamente, se propone explorar el uso de modelos GARCH (Generalized Autoregressive Conditional Heteroskedasticity) para la estimación y predicción de la volatilidad condicional del activo. Estos modelos son ampliamente utilizados en series financieras por su capacidad para capturar la heterocedasticidad. Su incorporación permitiría modelar de manera explícita la variabilidad del mercado, integrando esta información como input adicional en los modelos de predicción de impulsos y retrocesos. Combinando la direccionalidad aprendida por modelos como XGBoost o redes neuronales con predicciones de volatilidad GARCH, podría obtenerse una estrategia más robusta que tenga en cuenta tanto la magnitud esperada como el riesgo asociado a cada movimiento del mercado.

16. Bibliografía

- Athey, S., & Imbens, G. (2019). Machine learning methods that economists should know about. *Annual Review of Economics, 11*(1), 685–725. https://doi.org/10.1146/annurev-economics-080217-053433

- BBVA. (2024). Nuevo 'halving' a la vista: así es el proceso que regula el bitcoin. Recuperado de https://www.bbva.com/es/innovacion/nuevo-halving-a-la-vista-asi-es-el-proceso-que-regula-el-bitcoin/

- Binance. (2024). *Binance API Documentation*. Recuperado de https://binance-docs.github.io/apidocs

- Bloomberg News. (2024, 21 de marzo). *XTX Markets pledges $1 billion for AI-powered data center in Finland*. Bloomberg. https://www.bloomberg.com/news/articles/2025-01-22/gerko-s-xtx-to-build-1-billion-data-hub-in-machine-learning-bet

- Bloomberg News**.** (2014, 23 de julio). *Don't Tell Anybody About This Story on HFT Power Jump Trading*. Recuperado de: https://www.bloomberg.com/news/articles/2014-07-23/don-t-tell-anybody-about-this-story-on-hft-power-jump-trading

- Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. En *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785–794). https://doi.org/10.1145/2939672.2939785

- Coinbase. (2024). ¿Qué es un halving de Bitcoin? Recuperado de https://www.coinbase.com/es-es/learn/crypto-basics/what-is-a-bitcoin-halving

- Cointelegraph. (2024). *Halving de Bitcoin - ¿Cómo funciona el ciclo de halving y por qué es importante?* Recuperado de https://es.cointelegraph.com/learn/bitcoin-halving-how-does-the-halving-cycle-work-and-why-does-it-matter

- Expansión. (2024, 29 de enero). *Amazon, Microsoft y Google invertirán 21.000 millones en centros de datos en España*. https://www.expansion.com/economia-digital/2024/05/27/66538181e5fdea96458b456d.html

- Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics, 29*(5), 1189–1232. https://doi.org/10.1214/aos/1013203451

- García, D., & Schweitzer, F. (2015). Social signals and algorithmic trading of Bitcoin. *arXiv preprint arXiv:1506.01513*. https://arxiv.org/abs/1506.01513

- King, J. C., Dale, R., & Amigó, J. M. (2024). Blockchain metrics and indicators in cryptocurrency trading. *arXiv preprint arXiv:2403.00770*. https://arxiv.org/abs/2403.00770

- López de Prado, M. (2018). *Advances in Financial Machine Learning*. Wiley.

- Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. En *Advances in Neural Information Processing Systems* (pp. 4765–4774). https://proceedings.neurips.cc/paper_files/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf

- Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. https://bitcoin.org/bitcoin.pdf

- Official Data Foundation. (2024). *Bitcoin Price History Chart*. Recuperado de https://www.officialdata.org/bitcoin-price

- Pardo, Á. (2020). Trading algorítmico: El futuro de los mercados financieros. *Revista de Análisis Financiero, 32*(1), 45–60.

- TA-Lib. (n.d.). *Python wrapper for TA-Lib*. Recuperado de TA-Lib

- U.S. Department of the Treasury. (2024). *Daily Treasury Yield Curve Rates*. Recuperado de https://home.treasury.gov

- U.S. Senate Committee on Homeland Security and Governmental Affairs. (2024). *Hedge fund use of AI: Report* (pp. 28–29). https://www.hsgac.senate.gov/wp-content/uploads/2024.06.11-Hedge-Fund-Use-of-AI-Report.pdf

ANEXO 1. Código de la extracción de los datos, depuración, imputación, organización y feuture engineering.

1. Cargamos librerías

Categoría	Librerías
Recogida de datos	`ccxt` , `yfinance` , `requests` , `BeautifulSoup` , `fake_useragent`
Imputación y normalización	`pandas` , `numpy` , `sklearn.preprocessing.StandardScaler`
Depuración y análisis	`math` , `scipy.optimize.minimize`
Creación de variables	`talib` , `datetime` , `timedelta` , `pandas`
Representación gráfica	`matplotlib.pyplot` , `mplfinance.original_flavor.candlestick_ohlc` , `matplotlib.dates` , `seaborn` , `plotly.graph_objs` , `plotly.offline` , `matplotlib.ticker.FuncFormatter`
Configuración gráfica	`matplotlib` , `sns.set_style("white")` , `plotly`

```
library(reticulate)

Warning: package 'reticulate' was built under R version 4.4.3

reticulate::py_config()

python:         C:/Users/34609/AppData/Local/R/cache/R/reticulate/uv/cache/archive-v0/UXk0Wu26q3
AFsHxiJyHKV/Scripts/python.exe
libpython:      C:/Users/34609/AppData/Local/R/cache/R/reticulate/uv/python/cpython-3.11.12-wind
ows-x86_64-none/python311.dll
pythonhome:     C:/Users/34609/AppData/Local/R/cache/R/reticulate/uv/cache/archive-v0/UXk0Wu26q3
AFsHxiJyHKV
virtualenv:     C:/Users/34609/AppData/Local/R/cache/R/reticulate/uv/cache/archive-v0/UXk0Wu26q3
AFsHxiJyHKV/Scripts/activate_this.py
version:        3.11.12 (main, Apr  9 2025, 04:03:34) [MSC v.1943 64 bit (AMD64)]
Architecture:   64bit
numpy:          C:/Users/34609/AppData/Local/R/cache/R/reticulate/uv/cache/archive-v0/UXk0Wu26q3
AFsHxiJyHKV/Lib/site-packages/numpy
numpy_version:  2.2.5

NOTE: Python version was forced by VIRTUAL_ENV

import ccxt
 import pandas as pd
 from datetime import datetime, timedelta
 import time
 import talib as ta
 import matplotlib.pyplot as plt
 from mplfinance.original_flavor import candlestick_ohlc
 import matplotlib.dates as mdates
 import math
 import requests
 import yfinance as yf
 from bs4 import BeautifulSoup          # for web scraping and parsing HTML
 from fake_useragent import UserAgent  # provides a fake User-Agent header for web s
 import seaborn as sns
 sns.set_style("white")
 import numpy as np
 import matplotlib as mpl
 from matplotlib.ticker import FuncFormatter
 from sklearn.preprocessing import StandardScaler
 # Plotly
```

```
 from plotly.offline import init_notebook_mode, iplot, plot
 import plotly as py
 init_notebook_mode(connected=True)
 import plotly.graph_objs as go
 from scipy.optimize import minimize
 import mplfinance as mpf
 from matplotlib.lines import Line2D
 import ccxt
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier, export_text, DecisionTreeRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, precision_s
core, recall_score, f1_score, roc_auc_score, roc_curve, auc
from sklearn.metrics import make_scorer, mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.decomposition import FactorAnalysis
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

2. Configuración de la conexión a Binance

```
exchange = ccxt.binance()
 symbol = 'BTC/USDT'  # Activo del que quieres obtener datos
 timeframe = '1h'  # Intervalo de velas, puede ser '1m', '5m', '1h', '1d', etc.
 limit = 1000  # Límite de velas por solicitud (máximo 1000 en Binance)
 max_records = 5000  # Número máximo de registros que deseas obtener
```

3. Funciones de recogida de datos, depuración e imputación

```
# Función para extraer datos históricos con paginación
def data_history(symbol, timeframe, since=None, limit=limit, max_records=max_records):
    all_data = []
    while len(all_data) < max_records:
        ohlcv = exchange.fetch_ohlcv(symbol, timeframe, since=since, limit=limit)
        if len(ohlcv) == 0:
            break
        all_data.extend(ohlcv)
        since = since = ohlcv[-1][0] + (ohlcv[1][0] - ohlcv[0][0])  # Incrementar por el interva
lo entre las velas
        # Incrementar para evitar duplicados
        if len(ohlcv) < limit:
            break
        time.sleep(exchange.rateLimit / 1000)  # Respetar el límite de velocidad de la API
    return all_data[:max_records]

def amplitud(timeframe):
    intervalos_n = {
        '1m': 18*60,    # 1 minuto en ms
        '5m': 18*12,   # 5 minutos en ms
        '15m': 18*4,  # 15 minutos en ms
        '1h': 18,# 1 hora en ms
        '4h': math.ceil(18/4),#4 hora en ms
        '6h': math.ceil(18/6),#6 hora en ms
        '12h': math.ceil(18/12),#12 hora en ms
        '1d': 2, # 1 día en ms
        '1w': 1, # 1 semana en ms
        '1M': 1 # 1 mes en ms
    }
    return intervalos_n.get(timeframe)

def timeframe_to_ms(timeframe):
    intervalos_ms = {
        '1m': 60000,    # 1 minuto en ms
        '5m': 300000,   # 5 minutos en ms
        '15m': 900000,  # 15 minutos en ms
```

```
        '1h': 3600000,# 1 hora en ms
        '4h': 3600000*4,#4 hora en ms
        '6h': 3600000*6,#6 hora en ms
        '12h': 3600000*12,#12 hora en ms
        '1d': 86400000, # 1 día en ms
        '1w': 86400000*7, # 1 semana en ms
        '1M': 86400000*30 # 1 mes en ms
    }
    return intervalos_ms.get(timeframe)

def rep_data_history(symbol, timeframe):

    #En función del timeframe sacamos el tamaño de n para no tener problemas de amplitud
    n = amplitud(timeframe)

    #Ajustar el timestamp en función de la temporalidad a la hora de recoger los datos
    time_ms = timeframe_to_ms(timeframe)

    # Obtener datos históricos primer bloque
    bloques = 1
    ohlcv = data_history(symbol, timeframe)
    df_completo = pd.DataFrame(ohlcv, columns=['timestamp', 'open', 'high', 'low', 'close', 'vol
ume'])

    #Ajustes del timestamp
    # Convertir max_records horas a milisegundos
    hours_in_ms = (max_records) * time_ms

    while bloques < n:
        #Saco el nuevo bloque
        # Retroceder desde el primer timestamp del dataframe
        since = int(df_completo['timestamp'][0] - hours_in_ms)
        bloque_nuevo = data_history(symbol, timeframe, since=since)
        df_nuevo = pd.DataFrame(bloque_nuevo, columns=['timestamp', 'open', 'high', 'low', 'clos
e', 'volume'])
        # Añadir el nuevo bloque al dataframe completo
        df_completo = pd.concat([df_completo, df_nuevo])

        # Ordenar por timestamp de más antiguo a más reciente
        df_completo = df_completo.drop_duplicates(subset='timestamp').sort_values(by='timestamp'
).reset_index(drop=True)
        # Incrementar el contador de bloques
        bloques += 1

    return df_completo

def verificar_saltos(df, timeframe):
    # Convertir el timeframe a milisegundos
    intervalos = {
        '1m': 60000,    # 1 minuto en ms
        '5m': 300000,   # 5 minutos en ms
        '15m': 900000,  # 15 minutos en ms
        '1h': 3600000,# 1 hora en ms
        '4h': 3600000*4,#4 hora en ms
        '6h': 3600000*6,#6 hora en ms
        '12h': 3600000*12,#12 hora en ms
        '1d': 86400000, # 1 día en ms
        '1w': 86400000*7, # 1 semana en ms
        '1M': 86400000*30 # 1 mes en ms
    }
    intervalo_esperado = intervalos.get(timeframe, None)
    if intervalo_esperado is None:
        raise ValueError(f"Timeframe '{timeframe}' no reconocido. Usa uno de {list(intervalos.ke
ys())}.")

    # Calcular diferencias entre timestamps consecutivos
    df['diff'] = df['timestamp'].diff()

    # Verificar dónde las diferencias no coinciden con el intervalo esperado
```

```
    saltos = df[df['diff'] != intervalo_esperado]

    # Resultado
    if saltos.empty:
        print("No hay saltos en los datos.")
    else:
        print(f"Se encontraron {len(saltos)} saltos en los datos:")
        print(saltos[['timestamp', 'diff']])

    # Eliminar la columna auxiliar antes de devolver
    df.drop(columns=['diff'], inplace=True)

#La siguiente función es para rellenar los huecos entre observaciones
def rellenar_huecos(df, timeframe):
    """
    Rellena los huecos en un dataframe de una serie temporal usando la observación anterior.

    :param df: DataFrame con columnas ['timestamp', 'open', 'high', 'low', 'close', 'volume']
    :param timeframe: String indicando el intervalo de tiempo (e.g., '1m', '5m', '1h')
    :return: DataFrame con los huecos rellenados.
    """
    # Convertir el timeframe a milisegundos
    intervalos = {
        '1m': 60000,    # 1 minuto en ms
        '5m': 300000,   # 5 minutos en ms
        '15m': 900000,  # 15 minutos en ms
        '1h': 3600000,  # 1 hora en ms
        '4h': 3600000*4,# 4 horas en ms
        '6h': 3600000*6,# 6 horas en ms
        '12h': 3600000*12, # 12 horas en ms
        '1d': 86400000, # 1 día en ms
        '1w': 86400000*7, # 1 semana en ms
        '1M': 86400000*30 # 1 mes en ms
    }
    intervalo_esperado = intervalos.get(timeframe, None)
    if intervalo_esperado is None:
        raise ValueError(f"Timeframe '{timeframe}' no reconocido. Usa uno de {list(intervalos.ke
ys())}.")

    # Crear un rango completo de timestamps esperados
    timestamp_inicio = df['timestamp'].min()
    timestamp_fin = df['timestamp'].max()
    timestamps_completos = pd.DataFrame({
        'timestamp': list(range(timestamp_inicio, timestamp_fin + intervalo_esperado, intervalo_
esperado))
    })

    # Hacer un merge para identificar los huecos
    df_completo = timestamps_completos.merge(df, on='timestamp', how='left')

    # Rellenar los valores NaN con el último valor válido hacia adelante
    df_completo[['open', 'high', 'low', 'close', 'volume']] =
    df_completo[['open', 'high', 'low', 'close', 'volume']].fillna(method='ffill')

    return df_completo
```

4. Obtenemos los datos históricos para temporalidad de 1h.

```
timeframe = '1h'
df = rep_data_history(symbol, timeframe)
#Verificar si hay huecos entre fechas
verificar_saltos(df, timeframe)
#Probar el relleno de huecos
df = rellenar_huecos(df, timeframe)
#Comprobar que se han llenado correctamente
verificar_saltos(df, timeframe)
```

```
Se encontraron 29 saltos en los datos:
          timestamp         diff
0      1502942400000          NaN
493    1504738800000   25200000.0
3354   1515042000000    7200000.0
4190   1518170400000  122400000.0
7470   1530014400000   39600000.0
7495   1530108000000    7200000.0
7650   1530691200000   28800000.0
10216  1539939600000   14400000.0
10833  1542186000000   28800000.0
13658  1552377600000   25200000.0
15189  1557925200000   39600000.0
17386  1565863200000   32400000.0
19538  1573617600000   10800000.0
19824  1574654400000   10800000.0
21646  1581217200000    7200000.0
21895  1582131600000   21600000.0
22224  1583319600000    7200000.0
23463  1587787200000   10800000.0
24997  1593320400000   14400000.0
28718  1606719600000    7200000.0
29229  1608573600000   18000000.0
29309  1608865200000    7200000.0
30462  1613019600000    7200000.0
31011  1614999600000    7200000.0
32090  1618891200000   10800000.0
32211  1619337600000   14400000.0
34845  1628834400000   18000000.0
35974  1632906000000   10800000.0
48962  1679666400000    7200000.0
Se encontraron 1 saltos en los datos:
       timestamp  diff
0  1502942400000   NaN
```

5. Funciones de cáculo de las variables Indicadores Técnicos - Talib

```python
def add_obv_with_divergence(df):
    """
    Añade el indicador técnico OBV, su versión estandarizada y detecta divergencias positivas y negativas.

    Args:
    df (pd.DataFrame): DataFrame que contiene las columnas 'close' y 'volume'.

    Returns:
    pd.DataFrame: DataFrame con las columnas adicionales 'OBV', 'OBV_scaled', y 'Divergence'.
    """
    # Calcula el OBV
    df['OBV'] = ta.OBV(df['close'], df['volume'])

    # Estandariza el OBV
    scaler = StandardScaler()
    df['OBV_scaled'] = scaler.fit_transform(df[['OBV']])

    # Inicializa la columna para divergencias
    df['OBV_Divergence'] = 0  # 0 significa sin divergencia

    # Detecta divergencias (positivas = 1, negativas = -1)
    for i in range(2, len(df)):
        # Precios y OBV actuales y pasados
        price_now = df['close'].iloc[i]
        price_prev = df['close'].iloc[i-1]
        obv_now = df['OBV'].iloc[i]
        obv_prev = df['OBV'].iloc[i-1]

        # Detecta divergencia positiva
        if price_now < price_prev and obv_now > obv_prev:
            df['OBV_Divergence'].iloc[i] = 1

        # Detecta divergencia negativa
        elif price_now > price_prev and obv_now < obv_prev:
            df['OBV_Divergence'].iloc[i] = -1
```

```
    return df


def adx_dmi(df, timeperiods=[7, 14, 21]):
    """
    Calcula indicadores ADX, DMI+, DMI-, y variables relacionadas para múltiples períodos.

    Parámetros:
        df (pd.DataFrame): DataFrame con columnas 'high', 'low', 'close'.
        timeperiods (list): Lista de períodos para calcular los indicadores.

    Retorna:
        pd.DataFrame: El DataFrame original con columnas adicionales para los indicadores.
    """
    for period in timeperiods:
        # Sufijo para las columnas de este período
        suffix = f"_{period}"

        # Calcular indicadores básicos: DMI+, DMI-, ADX
        df[f'dmi_plus{suffix}'] = ta.PLUS_DI(df['high'], df['low'], df['close'], timeperiod=peri
od)
        df[f'dmi_minus{suffix}'] = ta.MINUS_DI(df['high'], df['low'], df['close'], timeperiod=pe
riod)
        df[f'adx{suffix}'] = ta.ADX(df['high'], df['low'], df['close'], timeperiod=period)

        # Variables para detectar cruces entre DMI+ y DMI-
        df[f'cruce_alcista_dmi{suffix}'] = ((df[f'dmi_plus{suffix}'] > df[f'dmi_minus{suffix}'])
& \
                                            (df[f'dmi_plus{suffix}'].shift(1) <= df[f'dmi_minus{suffi
x}'].shift(1))).astype(int)
        df[f'cruce_bajista_dmi{suffix}'] = ((df[f'dmi_minus{suffix}'] > df[f'dmi_plus{suffix}'])
& \
                                            (df[f'dmi_minus{suffix}'].shift(1) <= df[f'dmi_plus{suffi
x}'].shift(1))).astype(int)

        # Variable para la fuerza de la tendencia: Diferencia entre DMI+ y DMI-
        df[f'fuerza_dmi{suffix}'] = abs(df[f'dmi_plus{suffix}'] - df[f'dmi_minus{suffix}'])

        # Variable para detectar si el ADX indica una tendencia fuerte (>25)
        df[f'trend_strong_adx{suffix}'] = (df[f'adx{suffix}'] > 25).astype(int)

        # Variable para detectar si no hay tendencia (ADX < 20)
        df[f'notrend_adx{suffix}'] = (df[f'adx{suffix}'] < 20).astype(int)

        # Variable combinada: ADX indica fuerza y cruce de DMI's en la misma dirección
        df[f'confirmacion_tendencia{suffix}'] = (
            df[f'trend_strong_adx{suffix}'] &  # ADX > 25
            (df[f'cruce_alcista_dmi{suffix}'] | df[f'cruce_bajista_dmi{suffix}'])).astype(int)#
Cruce detectado


    return df

def stoch(df):
    # Parámetros para el Estocástico
    k_periods = [5, 9, 14]    # Diferentes valores para el %K
    d_periods = [3, 5, 9]     # Diferentes valores para el %D

    # Iteramos por los diferentes parámetros de %K y %D
    for k in k_periods:
        for d in d_periods:
            # Calcular el Estocástico con los diferentes parámetros
            df[f'STOCH_k_{k}_{d}'], df[f'STOCH_d_{k}_{d}'] = ta.STOCH(
                df['high'], df['low'], df['close'], fastk_period=k, slowk_period=k, slowd_period
=d
            )
```

```
            # Crear variables de sobrecompra y sobreventa
            df[f'STOCH_overbought_{k}_{d}'] = (df[f'STOCH_k_{k}_{d}'] > 80).astype(int)
            df[f'STOCH_oversold_{k}_{d}'] = (df[f'STOCH_k_{k}_{d}'] < 20).astype(int)

            # Crear variables para detectar el cruce alcista y bajista
            df[f'STOCH_cross_bull_{k}_{d}'] = (
                (df[f'STOCH_k_{k}_{d}'] > df[f'STOCH_d_{k}_{d}']) &
                (df[f'STOCH_k_{k}_{d}'].shift(1) <= df[f'STOCH_d_{k}_{d}'].shift(1))
            )
            df[f'STOCH_cross_bear_{k}_{d}'] = (
                (df[f'STOCH_k_{k}_{d}'] < df[f'STOCH_d_{k}_{d}']) &
                (df[f'STOCH_k_{k}_{d}'].shift(1) >= df[f'STOCH_d_{k}_{d}'].shift(1))
            )
    return df

def macd(df):
    # Cálculo del MACD, Signal Line y Histograma
    df['MACD'], df['MACD_signal'], df['MACD_hist'] = ta.MACD(
    df['close'], fastperiod=12, slowperiod=26, signalperiod=9
    )

    # Crear una variable que indique si el MACD está por encima o por debajo de la Signal Line
    df['MACD_above_signal'] = (df['MACD'] > df['MACD_signal']).astype(int)

    # Crear una variable que detecte el cruce del MACD con la Signal Line
    df['MACD_cross_signal'] = df['MACD_above_signal'].diff().fillna(0) != 0
    return df

def parabolic_sar(df):
    # Configuraciones del Parabolic SAR
    sar_configs = [
        {'af_start': 0.01, 'af_max': 0.2},  # Configuración SAR lenta
        {'af_start': 0.02, 'af_max': 0.2},  # Configuración SAR estándar
        {'af_start': 0.03, 'af_max': 0.3},  # Configuración SAR rápida
    ]

    # Cálculo del Parabolic SAR y detección de tendencias
    for i, config in enumerate(sar_configs, start=1):
        af_start = config['af_start']
        af_max = config['af_max']

        # Calcular el Parabolic SAR
        df[f'SAR_{i}'] = ta.SAR(df['high'], df['low'], acceleration=af_start, maximum=af_max)

        # Crear una variable que indique si el SAR está por debajo del precio (1) o por encima (
0)
        df[f'SAR_below_{i}'] = (df[f'SAR_{i}'] < df['close']).astype(int)

        # Crear una variable que indique si ha habido un cambio de tendencia (True si hay cambio
, False si no)
        df[f'SAR_trend_change_{i}'] = df[f'SAR_below_{i}'].diff().fillna(0) != 0

    return df

def bollinger_bands(df, bollinger_periods = [5, 10, 14, 20, 30]):
    # Cálculo de las Bandas de Bollinger para cada período
    for period in bollinger_periods:
        upper_band, middle_band, lower_band = ta.BBANDS(
            df['close'],            # Precios de cierre
            timeperiod=period,      # Periodo
            nbdevup=2,              # Multiplicador de desviación superior
            nbdevdn=2,              # Multiplicador de desviación inferior
            matype=0                # Tipo de media (0 = SMA)
        )

        # Agregar las bandas al DataFrame
        df[f'BB_upper_{period}'] = upper_band
        df[f'BB_middle_{period}'] = middle_band
        df[f'BB_lower_{period}'] = lower_band
```

```
    return df

def rsi(df, rsi_periods = [5, 9, 14, 20, 30]):
    # RSI usando precio de cierre
    for period in rsi_periods:
        df[f'RSI_Close_{period}'] = ta.RSI(df['close'], timeperiod=period)
        # Agregar información sobre sobrecompra y sobreventa para RSI Close
        df[f'RSI_Close_{period}_Overbought'] = (df[f'RSI_Close_{period}'] > 70).astype(int)
        df[f'RSI_Close_{period}_Oversold'] = (df[f'RSI_Close_{period}'] < 30).astype(int)

    # RSI usando la media (High + Low + Close) / 3
    df['typical_price'] = (df['high'] + df['low'] + df['close']) / 3
    for period in rsi_periods:
        df[f'RSI_TypicalPrice_{period}'] = ta.RSI(df['typical_price'], timeperiod=period)
        # Agregar información sobre sobrecompra y sobreventa para RSI Typical Price
        df[f'RSI_TypicalPrice_{period}_Overbought'] = (df[f'RSI_TypicalPrice_{period}'] > 70).as
type(int)
        df[f'RSI_TypicalPrice_{period}_Oversold'] = (df[f'RSI_TypicalPrice_{period}'] < 30).asty
pe(int)

    df = df.drop('typical_price', axis = 1)
    return df

def moving_average(df, periodos = [5, 7, 9, 14, 24, 48, 72]):
    '''
    Media Móvil simple (MA)
    Media Móvil Exponencial (EMA)
    Media Móvil Ponderada (WMA)
    Media Móvil Suavizada (SMMA)

    Para los diferentes periodos (5, 7, 9, 14, 48, 72)
    '''
    # Cálculo de medias móviles
    for periodo in periodos:
        # SMA: Media Móvil Simple
        df[f'SMA_{periodo}'] = ta.SMA(df['close'], timeperiod=periodo)

        # EMA: Media Móvil Exponencial
        df[f'EMA_{periodo}'] = ta.EMA(df['close'], timeperiod=periodo)

        # WMA: Media Móvil Ponderada
        df[f'WMA_{periodo}'] = ta.WMA(df['close'], timeperiod=periodo)

        # SMMA: Media Móvil Suavizada (opción personalizada)
        # Para SMMA, usaremos un cálculo acumulativo en caso de que no sea nativa en TA-Lib
        smma = []
        alpha = 1 / periodo
        for i in range(len(df)):
            if i == 0:
                smma.append(df['close'][i])  # El primer valor es el cierre inicial
            else:
                smma.append((df['close'][i] * alpha) + smma[i-1] * (1 - alpha))
        df[f'SMMA_{periodo}'] = smma
    return df

def add_atr(df, atr_periods = [5, 14, 20, 30, 50]):
    """
    Añade columnas de ATR al DataFrame para diferentes períodos.

    Args:
    df (pd.DataFrame): DataFrame que contiene las columnas 'high', 'low', y 'close'.
    atr_periods (list): Lista de períodos para calcular ATR.

    Returns:
    pd.DataFrame: DataFrame con columnas adicionales para cada ATR.
    """
    for period in atr_periods:
        column_name = f"ATR_{period}"
```

```
        df[column_name] = ta.ATR(df['high'], df['low'], df['close'], timeperiod=period)
    return df
```

6. Cálculamos e incluimos las variables de indicadores técnicos a nuestro conjunto de datos

```
df = moving_average(df)
df = rsi(df)
df = bollinger_bands(df)
df = parabolic_sar(df)
df = macd(df)
df = stoch(df)
df = adx_dmi(df)
df = add_atr(df)
df = add_obv_with_divergence(df)
```

7. Función para identificar los diferentes patrones de velas - Talib

```
def add_candlestick_patterns(df):
    """
    Añade todos los patrones de velas de la librería talib como nuevas columnas en un DataFrame,
    recodificando los valores:
        - 1 para valores > 0 (patrones alcistas)
        - -1 para valores < 0 (patrones bajistas)
        - 0 para no patrón.

    Parámetros:
        df (pd.DataFrame): DataFrame con columnas ['open', 'high', 'low', 'close'].

    Retorna:
        pd.DataFrame: DataFrame con columnas adicionales para cada patrón de velas recodificados
.
    """
    # Lista de patrones de velas disponibles en talib
    candlestick_patterns = [
        'CDL2CROWS', 'CDL3BLACKCROWS', 'CDL3INSIDE', 'CDL3LINESTRIKE', 'CDL3STARSINSOUTH',
        'CDL3WHITESOLDIERS', 'CDLABANDONEDBABY', 'CDLADVANCEBLOCK', 'CDLBELTHOLD', 'CDLBREAKAWAY
',
        'CDLCLOSINGMARUBOZU', 'CDLCONCEALBABYSWALL', 'CDLCOUNTERATTACK', 'CDLDARKCLOUDCOVER',
        'CDLDOJI', 'CDLDOJISTAR', 'CDLDRAGONFLYDOJI', 'CDLENGULFING', 'CDLEVENINGDOJISTAR',
        'CDLEVENINGSTAR', 'CDLGAPSIDESIDEWHITE', 'CDLGRAVESTONEDOJI', 'CDLHAMMER', 'CDLHANGINGMA
N',
        'CDLHARAMI', 'CDLHARAMICROSS', 'CDLHIGHWAVE', 'CDLHIKKAKE', 'CDLHIKKAKEMOD',
        'CDLHOMINGPIGEON', 'CDLIDENTICAL3CROWS', 'CDLINNECK', 'CDLINVERTEDHAMMER', 'CDLKICKING',
        'CDLKICKINGBYLENGTH', 'CDLLADDERBOTTOM', 'CDLLONGLEGGEDDOJI', 'CDLLONGLINE', 'CDLMARUBOZ
U',
        'CDLMATCHINGLOW', 'CDLMATHOLD', 'CDLMORNINGDOJISTAR', 'CDLMORNINGSTAR', 'CDLONNECK',
        'CDLPIERCING', 'CDLRICKSHAWMAN', 'CDLRISEFALL3METHODS', 'CDLSEPARATINGLINES',
        'CDLSHOOTINGSTAR', 'CDLSHORTLINE', 'CDLSPINNINGTOP', 'CDLSTALLEDPATTERN', 'CDLSTICKSANDW
ICH',
        'CDLTAKURI', 'CDLTASUKIGAP', 'CDLTHRUSTING', 'CDLTRISTAR', 'CDLUNIQUE3RIVER',
        'CDLUPSIDEGAP2CROWS', 'CDLXSIDEGAP3METHODS'
    ]

    # Crear una copia para evitar modificar el dataframe original
    df = df.copy()

    # Añadir y recodificar cada patrón como una columna en el dataframe
    for pattern in candlestick_patterns:
        pattern_function = getattr(ta, pattern)
        df[pattern] = pattern_function(df['open'], df['high'], df['low'], df['close'])

        # Recodificar los valores
        df[pattern] = df[pattern].apply(lambda x: 1 if x > 0 else (-1 if x < 0 else 0))

    return df

df = add_candlestick_patterns(df)
```

8. Incluimos los patrones de velas a nuestro conjunto de datos

```
df = add_candlestick_patterns(df)
```

9. Función para añadir los retrocesos de Fibonacci

```
def add_multiple_fibonacci_retracements(df, windows=[24, 48, 120, 336, 720, 1440]):
    """
    Calcula los niveles de retrocesos de Fibonacci (0%, 23.6%, 38.2%, 50%, 61.8%, 100%)
    para diferentes ventanas de tiempo (lookback).

    Parámetros:
        df (pd.DataFrame): DataFrame con columnas ['high', 'low'].
        windows (list): Lista de diferentes valores para la ventana de retroceso.

    Retorna:
        pd.DataFrame: DataFrame con columnas adicionales para los retrocesos de cada ventana.
    """
    df = df.copy()

    for window in windows:
        # Crear los máximos y mínimos para la ventana
        df[f'fib_high_{window}'] = df['high'].rolling(window=window).max()
        df[f'fib_low_{window}'] = df['low'].rolling(window=window).min()

        # Calcular niveles de Fibonacci para esta ventana
        df[f'fib_0_{window}'] =
        df[f'fib_high_{window}']  # Nivel 0%

        df[f'fib_23.6_{window}'] =
        df[f'fib_high_{window}'] - 0.236 * (df[f'fib_high_{window}'] - df[f'fib_low_{window}'])

        df[f'fib_38.2_{window}'] =
        df[f'fib_high_{window}'] - 0.382 * (df[f'fib_high_{window}'] - df[f'fib_low_{window}'])

        df[f'fib_50_{window}'] =
        (df[f'fib_high_{window}'] + df[f'fib_low_{window}']) / 2  # Nivel 50%

        df[f'fib_61.8_{window}'] =
        df[f'fib_high_{window}'] - 0.618 * (df[f'fib_high_{window}'] - df[f'fib_low_{window}'])

        df[f'fib_78.6_{window}'] =
        df[f'fib_high_{window}'] - 0.786 * (df[f'fib_high_{window}'] - df[f'fib_low_{window}'])

        df[f'fib_100_{window}'] = df[f'fib_low_{window}']  # Nivel 100%

    return df
```

10. Añadimos los retrocesos de Fibonacci a nuestra base de datos

```
df = add_multiple_fibonacci_retracements(df)
```

11. Función para obtener las variables macroeconómicas de los tipos de interés en EEUU.

```
#Creamos la instancia para scrapear la web
def get_interest_rate():

    user_agent = UserAgent()

    headers = {
    "User-Agent": user_agent.random
    }

    res = requests.get("https://home.treasury.gov/resource-center/data-chart-center/interest-rat
es/TextView?type=daily_treasury_yield_curve&field_tdr_date_value=2023", headers=headers)
    html = BeautifulSoup(res.text, "lxml")
```

```
    #Ahora recogemos la lista de años disponibles para recoger en la web
    list_of_tag_years = html.find("select",
    {"data-drupal-selector": "edit-field-tdr-date-value"}).find_all("option")
    list_years = [tag.text for tag in list_of_tag_years[1:len(list_of_tag_years) - 1]]

    #El url que utilizaremos para descargar los datos de los tipos de interés
    link = html.find("div", {"class": "csv-feed views-data-export-feed"}).find("a").get("href")

    #Ahora estamos listos para scrapear la web con los datos sobre los tipos de interés
    df_list = []
    list_years_1 = list_years[0:9]
    for year in list_years_1:
        print(year)
        response = requests.get(link.replace("2023", year)).text.split("\n")

        list_table = [[*row.split(",")] for row in response[1:]]

        df = pd.DataFrame(list_table, columns=response[0].split(","))
        df.columns = df.columns.str.strip('"')
        df_list.append(df)
        #Concatenamos los data frames

    df_tipos = pd.concat(df_list)
    # Sort by Date
    df_tipos['Date'] = pd.to_datetime(df_tipos['Date'])
    df_tipos = df_tipos.sort_values('Date')
    return df_tipos
```

12. Funciones para imputar y depurar datos de los tipos de interés

```
#Función para hacer un summary de las variables del data  frame y ver valores perdidos, duplicad
os...
def summary_statistics(df):
    """
    Genera un resumen de estadísticas básicas y análisis de valores faltantes
    para las columnas de un DataFrame.

    Parámetros:
    df (pd.DataFrame): DataFrame con las columnas a analizar.

    Retorna:
    pd.DataFrame: DataFrame resumen con estadísticas y análisis de valores faltantes.
    """
    summary = {
        "Variable": [],
        "Total Valores": [],
        "Valores No Nulos": [],
        "Valores Nulos": [],
        "% Nulos": [],
        "Promedio": [],
        "Mediana": [],
        "Mínimo": [],
        "Máximo": []
    }

    for col in df.columns:
        if col == "Date":  # Ignorar la columna 'Date'
            continue

        # Convertir la columna a numérica, manejando errores
        df[col] = pd.to_numeric(df[col], errors='coerce')

        total = df[col].shape[0]
        non_nulls = df[col].notnull().sum()
        nulls = df[col].isnull().sum()
        nulls_percentage = (nulls / total) * 100

        summary["Variable"].append(col)
        summary["Total Valores"].append(total)
```

```
        summary["Valores No Nulos"].append(non_nulls)
        summary["Valores Nulos"].append(nulls)
        summary["% Nulos"].append(round(nulls_percentage, 2))
        summary["Promedio"].append(df[col].mean() if non_nulls > 0 else None)
        summary["Mediana"].append(df[col].median() if non_nulls > 0 else None)
        summary["Mínimo"].append(df[col].min() if non_nulls > 0 else None)
        summary["Máximo"].append(df[col].max() if non_nulls > 0 else None)

    return pd.DataFrame(summary)

#Función para rellenar los datos ausentes de las variables de tipos de interés por el valor ante
rior
def fill_missing_values(df):
    """
    Rellena valores faltantes en las columnas (excepto 'Date') de un DataFrame
    usando el valor anterior más cercano.

    Parámetros:
    df (pd.DataFrame): DataFrame con un índice de tipo DatetimeIndex y columnas a procesar.

    Retorna:
    pd.DataFrame: DataFrame con los valores faltantes rellenados.
    """
    # Excluir la columna 'Date' si está presente
    columns_to_fill = [col for col in df.columns if col != 'Date']

    # Usar `fillna` con método 'ffill' para rellenar valores faltantes hacia adelante
    df[columns_to_fill] = df[columns_to_fill].fillna(method='ffill')

    return df
```

13. Scrapeamos datos sobre los tipos de interés en EEUU, depuramos e imputamos

```
df_tipos = get_interest_rate()
summary = summary_statistics(df_tipos)
#Vemos como a 2 y 4 meses hay un 31.36% de datos ausentes y un 76.37% de datos ausentes para 4 m
eses
#Nos planteamos eliminar las variables

#Rellenamos los datos perdidos del dataframe de tipos de interés:
df_tipos_rellenos = fill_missing_values(df_tipos)
summary = summary_statistics(df_tipos_rellenos)

#Vemos que siguen habiendo datos perdidos para 2 meses y 4 meses. Por lo tanto, los eliminamos d
e nuestra base de datos.
temporalidades_no_validas = ['2 Mo', '4 Mo']
df_tipos_rellenos = df_tipos_rellenos.drop(temporalidades_no_validas, axis = 1)
summary = summary_statistics(df_tipos_rellenos)
```

14. Cargamos datos históricos VIX (Sp500)

```
def load_and_sort_vix_data(vix_csv_path = 'VIX_History.csv'):
    """
    Carga los datos del VIX desde un archivo CSV, renombra las columnas y los ordena
    por fecha de más antiguo a más reciente.

    Parámetros:
    vix_csv_path (str): Ruta al archivo CSV que contiene los datos del VIX.

    Retorna:
    pd.DataFrame: DataFrame con los datos del VIX ordenados por fecha.
    """
    # Cargar el archivo CSV
    vix_data = pd.read_csv(vix_csv_path)

    # Renombrar las columnas
    vix_data.columns = ['Date', 'OPEN_VIX', 'HIGH_VIX', 'LOW_VIX', 'CLOSE_VIX']

    # Convertir la columna 'DATE_VIX' a tipo datetime
```

```
    vix_data['Date'] = pd.to_datetime(vix_data['Date'])

    # Ordenar los datos por la columna 'DATE_VIX' de más antiguo a más reciente
    vix_data = vix_data.sort_values(by='Date', ascending=True)

    # Resetear el índice después de ordenar
    vix_data = vix_data.reset_index(drop=True)

    # Paso 4: Filtrar los datos para que solo contengan fechas desde 2017
    vix_data = vix_data[vix_data['Date'] >= '2017-01-01']

    # Devolver el DataFrame ordenado
    return vix_data

# Ejemplo de uso

vix_data_sorted = load_and_sort_vix_data()
```

	Date	OPEN_VIX	HIGH_VIX	LOW_VIX	CLOSE_VIX
6802	2017-01-03	14.07	14.07	12.85	12.85
6803	2017-01-04	12.78	12.80	11.63	11.85
6804	2017-01-05	11.96	12.09	11.40	11.67
6805	2017-01-06	11.70	11.74	10.98	11.32
6806	2017-01-09	11.71	12.08	11.46	11.56

15. Añadimos las variables de tipos de interés y VIX a nuestro conjunto de datos global (Funciones e implementación)

```
def join_data_frame(df, df_aux):


    # Convertir las columnas de fecha a datetime
    df['timestamp_join'] = pd.to_datetime(df['timestamp_join'])
    df_aux= df_aux.reset_index()
    #df_aux['Date'] = pd.to_datetime(df_aux['Date'])

    # Expandir los datos del segundo dataframe
    expanded_df2 = pd.DataFrame({
        'timestamp_join': pd.date_range(start=df_aux['Date'].min(), end=df_aux['Date'].max(), fr
eq='H')
        }).merge(df_aux.rename(columns={'Date': 'timestamp_join'}), on='timestamp_join', how='le
ft')

    # Rellenar los valores del segundo dataframe
    expanded_df2.fillna(method='ffill', inplace=True)
    expanded_df2.set_index('timestamp_join', inplace=True)
    #df = df.drop('timestamp', axis = 1)
    # Unir ambos dataframes
    result = pd.merge(df, expanded_df2, on='timestamp_join', how='left')

    return result

df['timestamp_join'] = df.index
df = join_data_frame(df, df_tipos_rellenos)

#Una vez que ya juntamos los dos dataframes podemos ver com un summary los datos pérdidos
summary = summary_statistics(df)
#Intentamos rellenar esos datos pérdidos con los datos anteriores, en las variables de interest
rate
var = ['1 Mo', '3 Mo', '6 Mo', '1 Yr', '2 Yr', '3 Yr', '5 Yr', '7 Yr', '10 Yr',
```

```
       '20 Yr', '30 Yr']
df[var]= fill_missing_values(df[var])
#Una vez que ya juntamos los dos dataframes podemos ver com un summary los datos pérdidos
summary = summary_statistics(df)

df= join_data_frame(df, vix_data_sorted)

df['timestamp'] = pd.to_datetime(df['timestamp'])
df.set_index('timestamp', inplace=True)
df= df.drop(['timestamp_join'], axis = 1)
```

timestamp	open	high	low	close	volume	timestamp2	SMA_5	EMA_5	WMA
2017-08-17 04:00:00	4261.48	4313.62	4261.32	4308.83	47.181009	17395.166667	NaN	NaN	N
2017-08-17 05:00:00	4308.83	4328.69	4291.37	4315.32	23.234916	17395.208333	NaN	NaN	N
2017-08-17 06:00:00	4330.29	4345.45	4309.37	4324.35	7.229691	17395.250000	NaN	NaN	N
2017-08-17 07:00:00	4316.62	4349.99	4287.41	4349.99	4.443249	17395.291667	NaN	NaN	N
2017-08-17 08:00:00	4333.32	4377.85	4333.32	4360.69	0.972807	17395.333333	4331.836	4331.836	4341.0

5 rows × 313 columns

16. Recodificación de variables

```
'''
En primer lugar, eliminamos la variable de cierre, low y close ya que no tendremos información d
e esta
a tiempo real cuando vayamos a operar, solo tendremos el cierre anterior que coincidirá con
el de apertura en ese momento.

En segundo lugar, tendremos que trasladar la información de los indicadores técnicos una observa
ción, porque yo
en un instante real, solo tendré la información del instante interior y la siguiente instante te
ndrá información
actualizada de mi instante anterior, así constantemente.

Por último lo que haremos será recodificar las variables de que esten en formato precio para que
sean distancias
respecto a la variable open de apertura. De esta manera nuestros modelos podrán aprender de mejo
r manera
al estar aprendiendo de patrones y no de valores concretos.

'''

data.drop(columns=['close', 'high', 'low', 'index_x', 'index_y','HIGH_VIX', 'LOW_VIX', 'CLOSE_VI
X', 'timestamp2'], inplace=True)


# Obtener todas las columnas excepto 'open'
columnas_a_desplazar = [col for col in data.columns if col not in ['open', 'timestamp', 'OPEN_VI
X']]
```

```
data_desplazado = data.copy()
# Desplazar los valores una fila hacia abajo
data_desplazado[columnas_a_desplazar] = data_desplazado[columnas_a_desplazar].shift(1)

def recodificar_medias_moviles(df, periodos = [5, 7, 9, 14, 24, 48, 72]):
        # Cálculo de medias móviles
    for periodo in periodos:
        # SMA: Media Móvil Simple
        df[f'SMA_{periodo}'] = ((df['open'] - df[f'SMA_{periodo}']) / df[f'SMA_{periodo}'])*100

        df[f'SMMA_{periodo}'] = ((df['open'] - df[f'SMMA_{periodo}']) / df[f'SMMA_{periodo}'])*1
00
        # EMA: Media Móvil Exponencial
        df[f'EMA_{periodo}'] = ((df['open'] - df[f'EMA_{periodo}']) / df[f'EMA_{periodo}'])*100

        # WMA: Media Móvil Ponderada
        df[f'WMA_{periodo}'] = ((df['open'] - df[f'WMA_{periodo}']) / df[f'WMA_{periodo}'])*100
    return df

def recodificar_bollinger_bands(df, bollinger_periods = [5, 10, 14, 20, 30]):
    # Cálculo de las Bandas de Bollinger para cada período
    for period in bollinger_periods:

        # Agregar las bandas al DataFrame
        df[f'BB_upper_{period}'] = ((df['open'] - df[f'BB_upper_{period}']) / df[f'BB_upper_{per
iod}'])*100

        df[f'BB_middle_{period}'] = ((df['open'] - df[f'BB_middle_{period}']) / df[f'BB_middle_{
period}'])*100
        df[f'BB_lower_{period}'] = ((df['open'] - df[f'BB_lower_{period}']) / df[f'BB_lower_{per
iod}'])*100
    return df

def recodificar_parabolic_sar(df):
    # Cálculo del Parabolic SAR y detección de tendencias
    for i in [1, 2, 3]:
        df[f'SAR_{i}'] = ((df['open'] - df[f'SAR_{i}']) / df[f'SAR_{i}'])*100
    return df

def recodificar_multiple_fibonacci_retracements(df, windows=[24, 48, 120, 336, 720, 1440]):

    for window in windows:
        # Crear los máximos y mínimos para la ventana
        df[f'fib_high_{window}'] = ((df['open'] - df[f'fib_high_{window}']) / df[f'fib_high_{win
dow}'])*100
        df[f'fib_low_{window}'] = ((df['open'] - df[f'fib_low_{window}']) / df[f'fib_low_{window
}'])*100

        # Calcular niveles de Fibonacci para esta ventana
        df[f'fib_0_{window}'] = ((df['open'] - df[f'fib_0_{window}']) / df[f'fib_0_{window}'])*1
00  # Nivel 0%
        df[f'fib_23.6_{window}'] = ((df['open'] - df[f'fib_23.6_{window}']) / df[f'fib_23.6_{win
dow}'])*100
        df[f'fib_38.2_{window}'] = ((df['open'] - df[f'fib_38.2_{window}']) / df[f'fib_38.2_{win
dow}'])*100
        df[f'fib_50_{window}'] = ((df['open'] - df[f'fib_50_{window}']) / df[f'fib_50_{window}']
)*100
        df[f'fib_61.8_{window}'] = ((df['open'] - df[f'fib_61.8_{window}']) / df[f'fib_61.8_{win
dow}'])*100
        df[f'fib_78.6_{window}'] = ((df['open'] - df[f'fib_78.6_{window}']) / df[f'fib_78.6_{win
dow}'])*100
        df[f'fib_100_{window}'] = ((df['open'] - df[f'fib_100_{window}']) / df[f'fib_100_{window
}'])*100

    return df

data_desplazado = recodificar_medias_moviles(data_desplazado)
data_desplazado = recodificar_bollinger_bands(data_desplazado)
```

```
data_desplazado = recodificar_parabolic_sar(data_desplazado)
data_desplazado = recodificar_multiple_fibonacci_retracements(data_desplazado)
```

17. Creación de la variable objetivo target_pct

```
# Crear la variable de target (porcentaje de variación en 6 horas)
data_desplazado['target_pct'] = (data_desplazado['open'].shift(-5) - data_desplazado['open']) /
data_desplazado['open'] * 100
```

18. Depuración post-creación de variables

```
'''
Al realizar el desplazamiento, tengo algunas observaciones donde pierdo la inform
ación de igual manera con las variables de las cuales dependen de muchos datos
pasados, es decir, van a ser valores nan porque no tenemos registros de esos
datos pasados. Por lo tanto, ajustaremos la base de datos para no recoger los 2
primeros meses de donde no teniamos información (1440 = 2 meses).

Ahora tengo que eliminar también las 106 últimas ya que de estas no tengo datos
sobre el VIX.
'''

data_depurado = data_desplazado[1440:(len(data_desplazado)-107)]

#Transformamos las variables categóricas a categorías
categorical_cols = ["SAR_trend_change_1", "SAR_trend_change_2", "SAR_trend_change_3",
                    "MACD_cross_signal", "STOCH_cross_bull_5_3", "STOCH_cross_bear_5_3",
                    "STOCH_cross_bull_5_5", "STOCH_cross_bear_5_5", "STOCH_cross_bull_5_9",
                    "STOCH_cross_bear_5_9", "STOCH_cross_bull_9_3", "STOCH_cross_bear_9_3",
                    "STOCH_cross_bull_9_5", "STOCH_cross_bear_9_5", "STOCH_cross_bull_9_9",
                    "STOCH_cross_bear_9_9", "STOCH_cross_bull_14_3", "STOCH_cross_bear_14_3",
                    "STOCH_cross_bull_14_5", "STOCH_cross_bear_14_5", "STOCH_cross_bull_14_9",
                    "STOCH_cross_bear_14_9"]

# Convertir las variables categóricas a tipo 'category'
data_depurado[categorical_cols] = data_depurado[categorical_cols].astype("category")

#Eliminamos también aquellas variables que tienen valores constantes:
data_depurado = data_depurado.loc[:, data_depurado.nunique() > 1]

data_depurado.to_csv("BaseDatosFactorial.csv")
```

19. Feuture engeniering

Leamos ahora los datos y recodificamos las variables necesarias como las categorías para convertirlas en factor en R:

```
library(caret)
 library(rpart)
 library(rpart.plot)
 library(pROC)
 library(rattle)
 library(vcd)
 library(partykit)
library(dplyr)
library("FactoMineR")
library("factoextra")
library(lubridate)
 library(randomForest)
 library(OOBCurve)
 # Dado que los modelos que se van a crear pueden aprovechar los beneficios de la
 # paralelización, lo ponemos (si da problemas en tu ordenador, puedes obviarlo)
  library(foreach)
 library(doParallel)
 cluster <- makeCluster(detectCores()- 1)
 registerDoParallel(cluster)
 library(pacman)

data<-read.csv("BaseDatosFactorial.csv")
```

Recodificamos las variables categóricas a factor.

```
categorical_cols <- c("SAR_trend_change_1", "SAR_trend_change_2", "SAR_trend_change_3",
                      "MACD_cross_signal", "STOCH_cross_bull_5_3", "STOCH_cross_bear_5_3",
                      "STOCH_cross_bull_5_5", "STOCH_cross_bear_5_5", "STOCH_cross_bull_5_9",
                      "STOCH_cross_bear_5_9", "STOCH_cross_bull_9_3", "STOCH_cross_bear_9_3",
                      "STOCH_cross_bull_9_5", "STOCH_cross_bear_9_5", "STOCH_cross_bull_9_9",
                      "STOCH_cross_bear_9_9", "STOCH_cross_bull_14_3", "STOCH_cross_bear_14_3",
                      "STOCH_cross_bull_14_5", "STOCH_cross_bear_14_5", "STOCH_cross_bull_14_9",
                      "STOCH_cross_bear_14_9")

data_recodificado <- data |>
  mutate(across(all_of(categorical_cols), ~ as.factor(as.numeric(. == "True", 1, 0))))

# Identificar variables con valores espec?ficos y convertirlas en factor
convertir_a_factor <- function(df) {
  for (col in names(df)) {
    valores_unicos <- unique(df[[col]])

    # Verificar si los valores únicos corresponden a (0,1), (-1,1) o (-1,0)
    if (all(valores_unicos %in% c(0, 1)) ||
        all(valores_unicos %in% c(-1, 1)) ||
        all(valores_unicos %in% c(-1, 0))) {

      # Convertir a factor si no es ya un factor
      if (!is.factor(df[[col]])) {
        df[[col]] <- as.factor(df[[col]])
      }
    }
  }
  return(df)
}

# Aplicar la función a tu dataframe
data_recodificado <- convertir_a_factor(data_recodificado)

# Verificar que las variables han sido convertidas
#str(data_recodificado)
data_recodificado <- data_recodificado |>
  mutate(across(where(is.factor), ~ as.numeric(as.character(.))))
```

Creamos primero un conjunto de datos con los patrones de velas, recodificados para que recojan los patrones que ocurren en k observaciones:

```
library(zoo)

# Definir los valores de k
k_values <- c(5, 9, 14, 20, 30)

# Lista de patrones de velas de sobrecompra
patterns <- c("timestamp", "CDL2CROWS", "CDL3BLACKCROWS", "CDL3INSIDE", "CDL3LINESTRIKE",
"CDL3WHITESOLDIERS", "CDLADVANCEBLOCK", "CDLBELTHOLD", "CDLBREAKAWAY",
"CDLCLOSINGMARUBOZU", "CDLDARKCLOUDCOVER", "CDLDOJI", "CDLDOJISTAR",
"CDLDRAGONFLYDOJI", "CDLENGULFING", "CDLEVENINGDOJISTAR", "CDLEVENINGSTAR",
"CDLGAPSIDESIDEWHITE", "CDLGRAVESTONEDOJI", "CDLHAMMER", "CDLHANGINGMAN",
"CDLHARAMI", "CDLHARAMICROSS", "CDLHIGHWAVE", "CDLHIKKAKE", "CDLHIKKAKEMOD",
"CDLHOMINGPIGEON", "CDLIDENTICAL3CROWS", "CDLINVERTEDHAMMER",
"CDLLADDERBOTTOM", "CDLLONGLEGGEDDOJI", "CDLLONGLINE", "CDLMARUBOZU",
"CDLMATCHINGLOW", "CDLMORNINGDOJISTAR", "CDLMORNINGSTAR", "CDLPIERCING",
"CDLRICKSHAWMAN", "CDLRISEFALL3METHODS", "CDLSEPARATINGLINES",
"CDLSHOOTINGSTAR", "CDLSHORTLINE", "CDLSPINNINGTOP", "CDLSTALLEDPATTERN",
"CDLSTICKSANDWICH", "CDLTAKURI", "CDLTASUKIGAP", "CDLTHRUSTING",
"CDLTRISTAR", "CDLUNIQUE3RIVER", "CDLXSIDEGAP3METHODS")



data_patrones <- data_recodificado[,patterns]

data_patrones <- data_patrones |>
```

```
  mutate(timestamp = as.POSIXct(data_patrones$timestamp, format = "%Y-%m-%d %H:%M:%S", tz = "UTC
")) |>
  arrange(timestamp)
tiempo <- data_patrones$timestamp
data_patrones$timestamp<-NULL
patterns <- patterns[-1]
# Contar apariciones en la ventana de k períodos
for (pattern in patterns) {
  for (k in k_values) {
    data_patrones[[paste0(pattern, "_", k)]] <- rollapply(
      data_patrones[[pattern]],
      width = k,
      FUN = function(x) sum(x, na.rm = TRUE),  # Cuenta cuántas veces aparece
      fill = 0,
      align = "right"
    )
  }
}
data_patrones <- data_patrones |>
  mutate(timestamp = as.POSIXct(tiempo, format = "%Y-%m-%d %H:%M:%S", tz = "UTC")) |>
  select(c(-patterns))
```

Sobrecompra (Overbought)

```
data_recodificado <- data_recodificado |>
  mutate(timestamp = as.POSIXct(data_recodificado$timestamp, format = "%Y-%m-%d %H:%M:%S", tz =
"UTC")) |>
  arrange(timestamp)
#Estas variables luego las eliminaremos para liberar espacio
variables_interes <- c("RSI_Close_5_Overbought", "RSI_TypicalPrice_5_Overbought",
            "STOCH_overbought_5_3", "STOCH_overbought_5_5", "STOCH_overbought_5_9",
            "RSI_Close_9_Overbought", "RSI_TypicalPrice_9_Overbought",
            "STOCH_overbought_9_3", "STOCH_overbought_9_5", "STOCH_overbought_9_9",
            "RSI_Close_14_Overbought", "RSI_TypicalPrice_14_Overbought",
            "STOCH_overbought_14_3", "STOCH_overbought_14_5", "STOCH_overbought_14_9",
            "RSI_Close_20_Overbought", "RSI_TypicalPrice_20_Overbought",
            "RSI_Close_30_Overbought", "RSI_TypicalPrice_30_Overbought",
            "RSI_Close_5_Oversold", "RSI_TypicalPrice_5_Oversold",
            "STOCH_oversold_5_3", "STOCH_oversold_5_5", "STOCH_oversold_5_9",
            "RSI_Close_9_Oversold", "RSI_TypicalPrice_9_Oversold",
            "STOCH_oversold_9_3", "STOCH_oversold_9_5", "STOCH_oversold_9_9",
            "RSI_Close_14_Oversold", "RSI_TypicalPrice_14_Oversold",
            "STOCH_oversold_14_3", "STOCH_oversold_14_5", "STOCH_oversold_14_9",
            "RSI_Close_20_Oversold", "RSI_TypicalPrice_20_Oversold",
            "RSI_Close_30_Oversold", "RSI_TypicalPrice_30_Oversold", "timestamp")
data_feuture <- left_join(data_patrones, data_recodificado[, variables_interes], by = "timestamp
")
data_feuture <- data_feuture |>
  mutate(Overbought_5 = RSI_Close_5_Overbought + RSI_TypicalPrice_5_Overbought +
                   STOCH_overbought_5_3 + STOCH_overbought_5_5 + STOCH_overbought_5_9 -
                     CDL2CROWS_5 - CDL3BLACKCROWS_5 -CDL3LINESTRIKE_5 - CDLDARKCLOUDCOVER_5
         - CDLEVENINGDOJISTAR_5- CDLEVENINGSTAR_5 - CDLGRAVESTONEDOJI_5 - CDLHANGINGMAN_5 - CDLI
DENTICAL3CROWS_5 - CDLSHOOTINGSTAR_5,

    Overbought_9 = RSI_Close_9_Overbought + RSI_TypicalPrice_9_Overbought +
                   STOCH_overbought_9_3 + STOCH_overbought_9_5 + STOCH_overbought_9_9 -
                     CDL2CROWS_9 - CDL3BLACKCROWS_9 -CDL3LINESTRIKE_9 -
      CDLDARKCLOUDCOVER_9 -CDLEVENINGDOJISTAR_9- CDLEVENINGSTAR_9 - CDLGRAVESTONEDOJI_9 - CDLHAN
GINGMAN_9 - CDLIDENTICAL3CROWS_9 - CDLSHOOTINGSTAR_9,

    Overbought_14 = RSI_Close_14_Overbought + RSI_TypicalPrice_14_Overbought +
                   STOCH_overbought_14_3 + STOCH_overbought_14_5 + STOCH_overbought_14_9 -
                     CDL2CROWS_14 - CDL3BLACKCROWS_14 -CDL3LINESTRIKE_14 -
      CDLDARKCLOUDCOVER_14 - CDLEVENINGDOJISTAR_14- CDLEVENINGSTAR_14 - CDLGRAVESTONEDOJI_14 - C
DLHANGINGMAN_14 - CDLIDENTICAL3CROWS_14 - CDLSHOOTINGSTAR_14,

    Overbought_20 = RSI_Close_20_Overbought + RSI_TypicalPrice_20_Overbought -
                     CDL2CROWS_20 - CDL3BLACKCROWS_20 -CDL3LINESTRIKE_20 -
      CDLDARKCLOUDCOVER_20 - CDLEVENINGDOJISTAR_20- CDLEVENINGSTAR_20 - CDLGRAVESTONEDOJI_20 - C
DLHANGINGMAN_20 - CDLIDENTICAL3CROWS_20 - CDLSHOOTINGSTAR_20,
```

```
    Overbought_30 = RSI_Close_30_Overbought + RSI_TypicalPrice_30_Overbought -
                    CDL2CROWS_30 - CDL3BLACKCROWS_30 -CDL3LINESTRIKE_30 -
      CDLDARKCLOUDCOVER_30 - CDLEVENINGDOJISTAR_30- CDLEVENINGSTAR_30 - CDLGRAVESTONEDOJI_30 - C
DLHANGINGMAN_30 - CDLIDENTICAL3CROWS_30 - CDLSHOOTINGSTAR_30
  )
```

Sobreventa (Oversold)

```
data_feuture <- data_feuture |>
  mutate(Oversold_5= RSI_Close_5_Oversold + RSI_TypicalPrice_5_Oversold +
                   STOCH_oversold_5_3 + STOCH_oversold_5_5 + STOCH_oversold_5_9 + CDLHAMMER_5
         + CDLINVERTEDHAMMER_5 + CDLLADDERBOTTOM_5 + CDLMATCHINGLOW_5 + CDLMORNINGDOJISTAR_5 + C
DLMORNINGSTAR_5 + CDLPIERCING_5 + CDLTAKURI_5 + CDLUNIQUE3RIVER_5,

    Oversold_9 = RSI_Close_9_Oversold + RSI_TypicalPrice_9_Oversold +
                   STOCH_oversold_9_3 + STOCH_oversold_9_5 + STOCH_oversold_9_9 + CDLHAMMER_9
    + CDLINVERTEDHAMMER_9 + CDLLADDERBOTTOM_9 + CDLMATCHINGLOW_9 + CDLMORNINGDOJISTAR_9 + CDLMOR
NINGSTAR_9 + CDLPIERCING_9 + CDLTAKURI_9 + CDLUNIQUE3RIVER_9,

    Oversold_14 = RSI_Close_14_Oversold + RSI_TypicalPrice_14_Oversold+ STOCH_oversold_14_3
    + STOCH_oversold_14_5 + STOCH_oversold_14_9 + CDLHAMMER_14 + CDLINVERTEDHAMMER_14
    + CDLLADDERBOTTOM_14 + CDLMATCHINGLOW_14 + CDLMORNINGDOJISTAR_14 + CDLMORNINGSTAR_14
    + CDLPIERCING_14 + CDLTAKURI_14 + CDLUNIQUE3RIVER_14,

    Oversold_20 = RSI_Close_20_Oversold + RSI_TypicalPrice_20_Oversold + CDLHAMMER_20 + CDLINVER
TEDHAMMER_20
    + CDLLADDERBOTTOM_20 + CDLMATCHINGLOW_20 + CDLMORNINGDOJISTAR_20 + CDLMORNINGSTAR_20 +
      CDLPIERCING_20 + CDLTAKURI_20 + CDLUNIQUE3RIVER_20
                    ,

    Oversold_30 = RSI_Close_30_Oversold + RSI_TypicalPrice_30_Oversold + CDLHAMMER_30 + CDLINVER
TEDHAMMER_30 +
      CDLLADDERBOTTOM_30 + CDLMATCHINGLOW_30 + CDLMORNINGDOJISTAR_30 + CDLMORNINGSTAR_30 +
      CDLPIERCING_30 + CDLTAKURI_30 + CDLUNIQUE3RIVER_30)
```

Eliminamos las variables que no sean de interes para mejorar el procesado:

```
variables_interes <- variables_interes[-length(variables_interes)]
data_feuture <- data_feuture |>
  select(c(-variables_interes))
```

Cambio alcista (Change_trend_bull)

```
variables_interes1 <- c("SAR_trend_change_1", "SAR_trend_change_2", "SAR_trend_change_3",
                        "MACD_cross_signal", "STOCH_cross_bull_5_3", "STOCH_cross_bear_5_3",
                        "STOCH_cross_bull_5_5", "STOCH_cross_bear_5_5", "STOCH_cross_bull_5_9",
                        "STOCH_cross_bear_5_9", "STOCH_cross_bull_9_3", "STOCH_cross_bear_9_3",
                        "STOCH_cross_bull_9_5", "STOCH_cross_bear_9_5", "STOCH_cross_bull_9_9",
                        "STOCH_cross_bear_9_9", "STOCH_cross_bull_14_3", "STOCH_cross_bear_14_3",
                        "STOCH_cross_bull_14_5", "STOCH_cross_bear_14_5", "STOCH_cross_bull_14_9",
                        "STOCH_cross_bear_14_9", "cruce_alcista_dmi_7",  "cruce_alcista_dmi_14",
                        "cruce_alcista_dmi_21", "cruce_bajista_dmi_7", "cruce_bajista_dmi_14",
                        "cruce_bajista_dmi_21", "timestamp")
data_feuture <- data_feuture |>
  mutate(timestamp = as.POSIXct(tiempo, format = "%Y-%m-%d %H:%M:%S", tz = "UTC"))
data_feuture <- left_join(data_feuture, data_recodificado[, variables_interes1], by = "timestamp
")

data_feuture <- data_feuture |>
  mutate(change_trend_bull_5 = SAR_trend_change_1 + STOCH_cross_bull_5_3 + STOCH_cross_bull_5_5+
           STOCH_cross_bull_5_9 + MACD_cross_signal + cruce_alcista_dmi_7 +
           CDLHAMMER_5 + CDLINVERTEDHAMMER_5 + CDLMORNINGDOJISTAR_5 +
           CDLMORNINGSTAR_5 + CDLPIERCING_5 + CDLTAKURI_5 + CDLUNIQUE3RIVER_5 + CDLLADDERBOTTOM_
5 + CDLMATCHINGLOW_5,
         change_trend_bull_9= SAR_trend_change_1 + SAR_trend_change_2 +
           STOCH_cross_bull_9_3 + STOCH_cross_bull_9_5+ STOCH_cross_bull_9_9 +
           MACD_cross_signal + cruce_alcista_dmi_7 + CDLHAMMER_9 +
           CDLINVERTEDHAMMER_9 + CDLMORNINGDOJISTAR_9 + CDLMORNINGSTAR_9 +
           CDLPIERCING_9 + CDLTAKURI_9 + CDLUNIQUE3RIVER_9 + CDLLADDERBOTTOM_9 +
           CDLMATCHINGLOW_9,
```

```
        change_trend_bull_14 = SAR_trend_change_2 + STOCH_cross_bull_14_3 +
          STOCH_cross_bull_14_5+ STOCH_cross_bull_14_9 + MACD_cross_signal +
          cruce_alcista_dmi_14 + CDLHAMMER_14 + CDLINVERTEDHAMMER_14 +
          CDLMORNINGDOJISTAR_14 + CDLMORNINGSTAR_14 + CDLPIERCING_14 +
          CDLTAKURI_14 + CDLUNIQUE3RIVER_14 + CDLLADDERBOTTOM_14 +
          CDLMATCHINGLOW_14,
        change_trend_bull_20 = SAR_trend_change_3 +  MACD_cross_signal +
          cruce_alcista_dmi_21 + CDLHAMMER_20 + CDLINVERTEDHAMMER_20 +
          CDLMORNINGDOJISTAR_20 + CDLMORNINGSTAR_20 + CDLPIERCING_20 +
          CDLTAKURI_20 + CDLUNIQUE3RIVER_20 + CDLLADDERBOTTOM_20 +
          CDLMATCHINGLOW_20,
        change_trend_bull_30 = SAR_trend_change_3 +  MACD_cross_signal +
          cruce_alcista_dmi_21 + CDLHAMMER_30 + CDLINVERTEDHAMMER_30 +
          CDLMORNINGDOJISTAR_30 + CDLMORNINGSTAR_30 + CDLPIERCING_30 +
          CDLTAKURI_30 + CDLUNIQUE3RIVER_30 + CDLLADDERBOTTOM_30 +
          CDLMATCHINGLOW_30
        )
```

Cambio bajista (Change_trend_bear)

```
data_feuture <- data_feuture |>
  mutate(change_trend_bear_5 = SAR_trend_change_1 + STOCH_cross_bear_5_3 +
          STOCH_cross_bear_5_5+ STOCH_cross_bear_5_9 + MACD_cross_signal +
          cruce_bajista_dmi_7 - CDL2CROWS_5 - CDL3BLACKCROWS_5 -
          CDLDARKCLOUDCOVER_5 - CDLEVENINGSTAR_5 - CDLEVENINGDOJISTAR_5 -
          CDLGRAVESTONEDOJI_5 - CDLHANGINGMAN_5 - CDLIDENTICAL3CROWS_5 -
          CDLSHOOTINGSTAR_5,
        change_trend_bear_9= SAR_trend_change_1 + STOCH_cross_bear_9_3 +
          STOCH_cross_bear_9_5+ STOCH_cross_bear_9_9 + MACD_cross_signal +
          cruce_bajista_dmi_7 - CDL2CROWS_9 - CDL3BLACKCROWS_9 -
          CDLDARKCLOUDCOVER_9 - CDLEVENINGSTAR_9 - CDLEVENINGDOJISTAR_9 -
          CDLGRAVESTONEDOJI_9 - CDLHANGINGMAN_9 - CDLIDENTICAL3CROWS_9 -
          CDLSHOOTINGSTAR_9,
        change_trend_bear_14 = SAR_trend_change_2 + STOCH_cross_bear_14_3 +
          STOCH_cross_bear_14_5+ STOCH_cross_bear_14_9 + MACD_cross_signal +
          cruce_bajista_dmi_14 - CDL2CROWS_14 - CDL3BLACKCROWS_14 -
          CDLDARKCLOUDCOVER_14 - CDLEVENINGSTAR_14 - CDLEVENINGDOJISTAR_14 -
          CDLGRAVESTONEDOJI_14 - CDLHANGINGMAN_14 - CDLIDENTICAL3CROWS_14 -
          CDLSHOOTINGSTAR_14,
        change_trend_bear_20 = SAR_trend_change_3 +  MACD_cross_signal +
          cruce_bajista_dmi_21 - CDL2CROWS_20 - CDL3BLACKCROWS_20 -
          CDLDARKCLOUDCOVER_20 - CDLEVENINGSTAR_20 - CDLEVENINGDOJISTAR_20 -
          CDLGRAVESTONEDOJI_20 - CDLHANGINGMAN_20 - CDLIDENTICAL3CROWS_20 -
          CDLSHOOTINGSTAR_20,
        change_trend_bear_30 = SAR_trend_change_3 +  MACD_cross_signal +
          cruce_bajista_dmi_21 + CDL2CROWS_30 - CDL3BLACKCROWS_30 -
          CDLDARKCLOUDCOVER_30 - CDLEVENINGSTAR_30 - CDLEVENINGDOJISTAR_30 -
          CDLGRAVESTONEDOJI_30 - CDLHANGINGMAN_30 - CDLIDENTICAL3CROWS_30 -
          CDLSHOOTINGSTAR_30)
```

Eliminamos las variables que no sean de interes para mejorar el procesado:

```
variables_interes1 <- variables_interes1[-length(variables_interes1)]
data_feuture <- data_feuture |>
  select(c(-variables_interes1))
```

Continuación alcista (trend_cont_bull)

```
variables_interes2 <- c("MACD_above_signal", "trend_strong_adx_7",
                        "trend_strong_adx_14", "trend_strong_adx_21", "confirmacion_tendencia_7"
, "confirmacion_tendencia_14",
                        "confirmacion_tendencia_21","timestamp")

data_feuture <- left_join(data_feuture, data_recodificado[, variables_interes2],
                          by = "timestamp")

data_feuture <- data_feuture |>
  mutate(trend_cont_bull_7 = MACD_above_signal + trend_strong_adx_7 +
          confirmacion_tendencia_7 + CDL3WHITESOLDIERS_5 + CDLBELTHOLD_5 +
          CDLCLOSINGMARUBOZU_5 + CDLRISEFALL3METHODS_5,
        trend_cont_bull_14 = MACD_above_signal + trend_strong_adx_14 +
```

```
          confirmacion_tendencia_14 + CDL3WHITESOLDIERS_14 + CDLBELTHOLD_14 +
          CDLCLOSINGMARUBOZU_14 + CDLRISEFALL3METHODS_14,
        trend_cont_bull_21 = MACD_above_signal + trend_strong_adx_21 +
          confirmacion_tendencia_21 + CDL3WHITESOLDIERS_20 + CDLBELTHOLD_20 +
          CDLCLOSINGMARUBOZU_20 + CDLRISEFALL3METHODS_20)
```

Continuación bajista (trend_cont_bear)

```
data_feuture <- data_feuture |>
  mutate(trend_cont_bear_7 = MACD_above_signal + trend_strong_adx_7 +
          confirmacion_tendencia_7 - CDL3LINESTRIKE_5 - CDLSTALLEDPATTERN_5 -
          CDLTHRUSTING_5,
        trend_cont_bear_14 = MACD_above_signal + trend_strong_adx_14 +
          confirmacion_tendencia_14 - CDL3LINESTRIKE_14 -
          CDLSTALLEDPATTERN_14 - CDLTHRUSTING_14,
        trend_cont_bear_21 = MACD_above_signal + trend_strong_adx_21 +
          confirmacion_tendencia_21- CDL3LINESTRIKE_20 -
          CDLSTALLEDPATTERN_20 - CDLTHRUSTING_20)
```

Eliminamos las variables que no sean de interes para mejorar el procesado:

```
variables_interes2 <- variables_interes2[-length(variables_interes2)]
data_feuture <- data_feuture |>
  select(c(-variables_interes2))
```

Nos quedamos con las variables nuevas y las añadimos a data_recodificado:

```
data_recodificado <- data_recodificado |>
  select(c(-patterns, -variables_interes, -variables_interes1, -variables_interes2))

variables_nuevas <- c("Overbought_5", "Overbought_9","Overbought_14","Overbought_20","Overbought
_30",
                      "Oversold_5","Oversold_9","Oversold_14","Oversold_20",
                      "Oversold_30", "change_trend_bull_5", "change_trend_bull_9",
                      "change_trend_bull_14", "change_trend_bull_20",
                      "change_trend_bull_30", "change_trend_bear_5",
                      "change_trend_bear_9", "change_trend_bear_14",
                      "change_trend_bear_20",  "change_trend_bear_30",
                      "trend_cont_bull_7", "trend_cont_bull_14",
                      "trend_cont_bull_21", "trend_cont_bear_7",
                      "trend_cont_bear_14", "trend_cont_bear_21", "timestamp")

data_recodificado <- left_join(data_recodificado, data_feuture[, variables_nuevas])

# Guardas el data_recodificado en un archivo CSV
write.csv(data_recodificado, "data_recodificado.csv", row.names = FALSE)
```

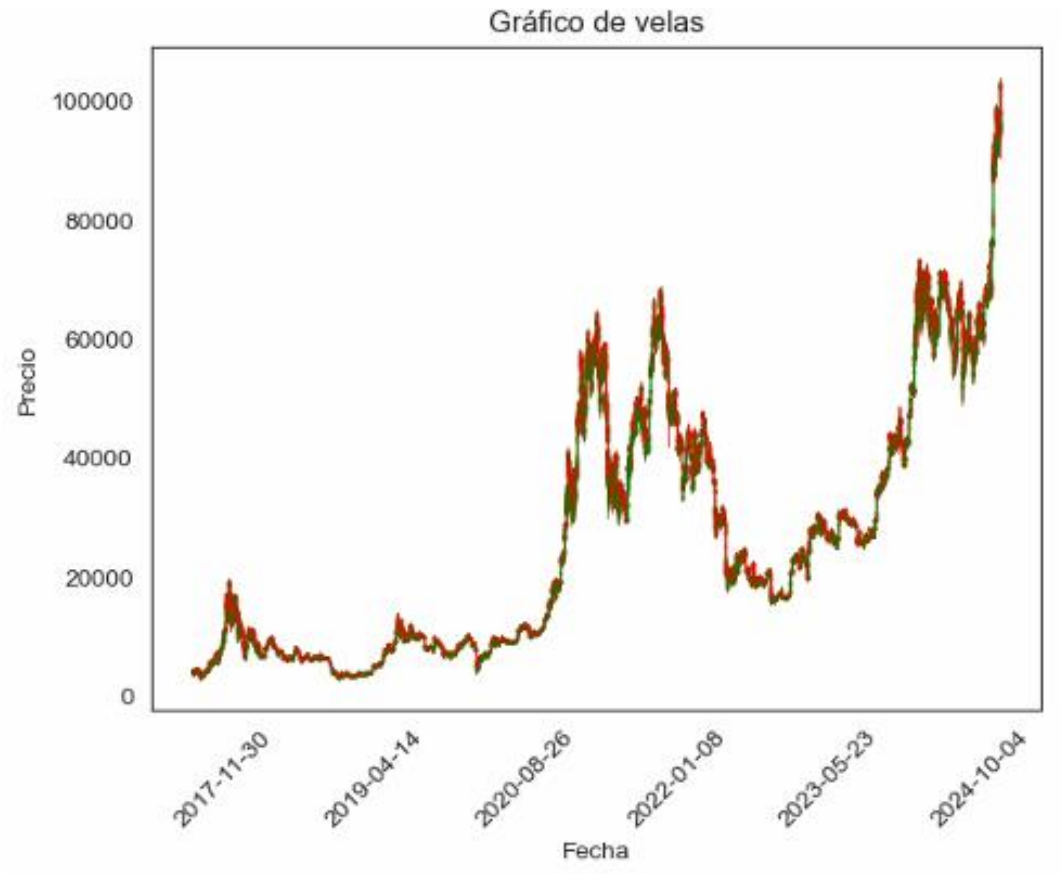
ANEXO 2. Gráficos variables (Ejemplos).

1. Cargamos librerias

```
library(reticulate)

Warning: package 'reticulate' was built under R version 4.4.3

reticulate::py_config()

python:         C:/Users/34609/AppData/Local/R/cache/R/reticulate/uv/cache/archive-v0/ujyEGGlHuu
jAt99QnBX8j/Scripts/python.exe
libpython:      C:/Users/34609/AppData/Local/R/cache/R/reticulate/uv/python/cpython-3.11.12-wind
ows-x86_64-none/python311.dll
pythonhome:     C:/Users/34609/AppData/Local/R/cache/R/reticulate/uv/cache/archive-v0/ujyEGGlHuu
jAt99QnBX8j
virtualenv:     C:/Users/34609/AppData/Local/R/cache/R/reticulate/uv/cache/archive-v0/ujyEGGlHuu
jAt99QnBX8j/Scripts/activate_this.py
version:        3.11.12 (main, Apr  9 2025, 04:03:34) [MSC v.1943 64 bit (AMD64)]
Architecture:   64bit
numpy:          C:/Users/34609/AppData/Local/R/cache/R/reticulate/uv/cache/archive-v0/ujyEGGlHuu
jAt99QnBX8j/Lib/site-packages/numpy
numpy_version:  2.2.6

NOTE: Python version was forced by VIRTUAL_ENV

import ccxt
 import pandas as pd
 from datetime import datetime, timedelta
 import time
 import talib as ta
 import matplotlib.pyplot as plt
 from mplfinance.original_flavor import candlestick_ohlc
 import matplotlib.dates as mdates
 import math
 import requests
 import yfinance as yf
 from bs4 import BeautifulSoup         # for web scraping and parsing HTML
 from fake_useragent import UserAgent  # provides a fake User-Agent header for web s
 import seaborn as sns
 sns.set_style("white")
 import numpy as np
 import matplotlib as mpl
 from matplotlib.ticker import FuncFormatter
 from sklearn.preprocessing import StandardScaler
 # Plotly
 from plotly.offline import init_notebook_mode, iplot, plot
 import plotly as py
 init_notebook_mode(connected=True)
 import plotly.graph_objs as go
 from scipy.optimize import minimize
 import mplfinance as mpf
 from matplotlib.lines import Line2D
 import ccxt
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier, export_text, DecisionTreeRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, precision_s
core, recall_score, f1_score, roc_auc_score, roc_curve, auc
from sklearn.metrics import make_scorer, mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.decomposition import FactorAnalysis
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

2. Gráfico de la evolución del activo históricamente mediante velas japonesas (Open, High, Low, Close)

```
#Gráfico de velas
def plot_candlestick(data):
    fig, ax = plt.subplots()

    # Crear una copia del DataFrame para evitar modificar el original
    data_copy = data.copy()

    # Si el índice contiene el 'timestamp', conviértelo a una columna
    if data_copy.index.name == 'timestamp':  # Comprueba si el índice tiene ese nombre
        data_copy.reset_index(inplace=True)

    # Convertir 'timestamp' a formato de fecha
    data_copy['timestamp'] = pd.to_datetime(data_copy['timestamp'], unit='ms')

    # Convertir 'timestamp' a mdates.date2num
    data_copy['timestamp'] = data_copy['timestamp'].apply(mdates.date2num)

    # Establecer el formato del eje de fecha en días
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))

    # Dibujar velas alcistas y bajistas con colores personalizados
    candlestick_ohlc(ax, data_copy[['timestamp', 'open', 'high', 'low', 'close']].values, width=
0.6,
                     colorup='g', colordown='r')


    ax.set_xlabel('Fecha')
    ax.set_ylabel('Precio')
    ax.set_title('Gráfico de velas')
    plt.xticks(rotation=45)
    plt.show()
```



El gráfico mostrado representa el precio de Bitcoin en temporalidad de una hora, utilizando velas japonesas para visualizar los movimientos del mercado. Las velas verdes indican impulsos alcistas, mientras que las velas rojas representan retrocesos bajistas.

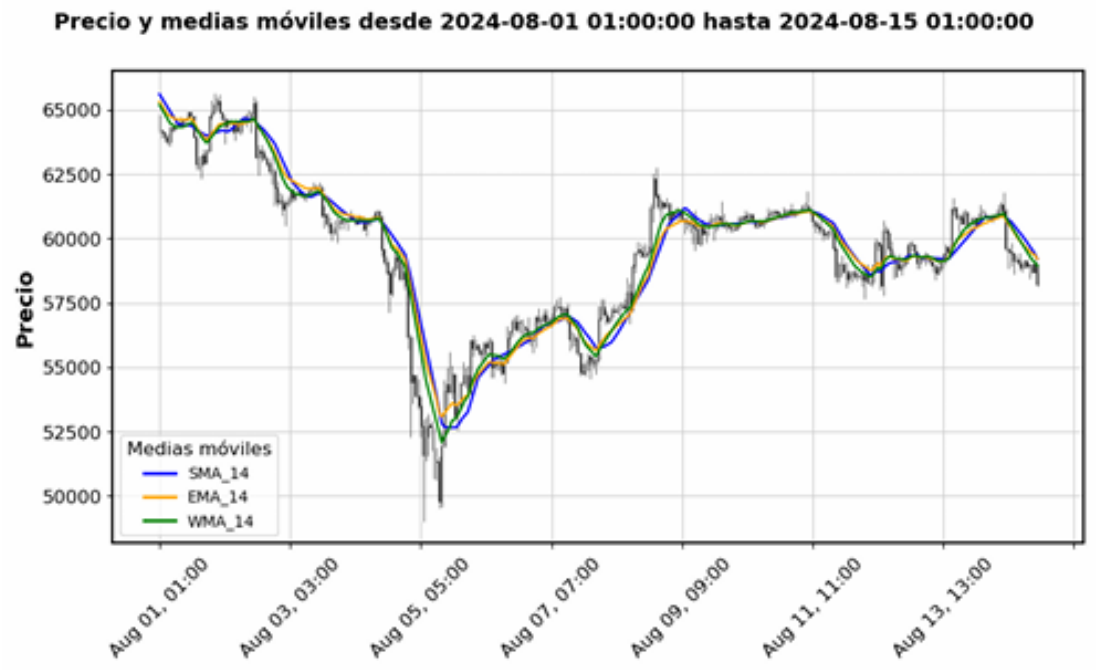
A lo largo del período analizado, se puede observar una tendencia alcista general, con un crecimiento notable del precio desde niveles muy bajos hasta superar los 100,000. No obstante, incluso dentro de esta tendencia positiva, se identifican múltiples fases de retroceso y corrección, lo cual es característico del comportamiento fractal de los mercados financieros: es decir, dentro de una tendencia principal pueden coexistir múltiples subtenencias en diferentes direcciones.

Esta estructura fractal implica que tanto los impulsos como los retrocesos se repiten en diferentes escalas temporales, y que una tendencia alcista a gran escala puede contener retrocesos bajistas temporales, que también pueden presentar su propia dinámica de impulsos y correcciones a menor escala. Este comportamiento debe ser considerado especialmente al trabajar con modelos predictivos o estrategias de trading basadas en patrones de velas.

3. Indicadores técnicos Medias Móviles. Ej: period = 14

```
#Función para representar gráficamente el precio junto a media móviles.
import pandas as pd
import matplotlib.pyplot as plt
import mplfinance as mpf

from matplotlib.lines import Line2D

def plot_candles_with_moving_averages(df, start_date, end_date):
    """
    Genera un gráfico de velas con medias móviles para un periodo de tiempo especificado.

    Parámetros:
    - df: DataFrame con índice datetime y columnas ['open', 'high', 'low', 'close'] y medias móv
iles (SMA, EMA, WMA).
    - start_date: Fecha de inicio del periodo (formato 'YYYY-MM-DD HH:MM:SS').
    - end_date: Fecha de fin del periodo (formato 'YYYY-MM-DD HH:MM:SS').
    """
    # Verificar si el índice es de tipo datetime
    if not pd.api.types.is_datetime64_any_dtype(df.index):
        print("El índice del DataFrame no está en formato datetime. Convirtiendo...")
        df.index = pd.to_datetime(df.index)

    # Filtrar el DataFrame por el rango de fechas
    df_filtered = df[(df.index >= start_date) & (df.index <= end_date)]

    # Verificar si hay datos en el rango de fechas
    if df_filtered.empty:
        print("No hay datos en el rango de fechas especificado.")
        return

    # Crear un diccionario para las medias móviles disponibles
    moving_averages = {
        'SMA_14': df_filtered['SMA_14'] if 'SMA_14' in df_filtered.columns else None,
        'EMA_14': df_filtered['EMA_14'] if 'EMA_14' in df_filtered.columns else None,
        'WMA_14': df_filtered['WMA_14'] if 'WMA_14' in df_filtered.columns else None
    }

    # Colores para las medias móviles
    colors = {'SMA_14': 'blue', 'EMA_14': 'orange', 'WMA_14': 'green'}

    # Crear un estilo para mplfinance
    style = mpf.make_mpf_style(base_mpf_style='classic', gridstyle='-', y_on_right=False)

    # Crear las líneas de medias móviles
    apds = [mpf.make_addplot(moving_averages[key], color=colors[key], linestyle='-', width=1.5)
            for key in moving_averages if moving_averages[key] is not None]

    # Graficar con mplfinance
    fig, axlist = mpf.plot(
```

```
        df_filtered,
        type='candle',
        style=style,
        addplot=apds,
        ylabel='Precio',
        ylabel_lower='Volumen',
        volume=False,
        figsize=(12, 6),
        returnfig=True
    )

    # Título del gráfico
    fig.suptitle(
        f"Precio y medias móviles desde {start_date} hasta {end_date}",
        fontsize=14,
        fontweight='bold',
        y=0.97
    )

    # Crear líneas personalizadas para la leyenda
    legend_elements = [
        Line2D([0], [0], color=colors[label], lw=2, label=label)
        for label in moving_averages if moving_averages[label] is not None
    ]

    # Añadir la leyenda en la esquina inferior izquierda
    ax = axlist[0]
    ax.legend(
        handles=legend_elements,
        loc='lower left',
        frameon=True,
        fontsize=10,
        title="Medias móviles",
        title_fontsize=12,
        facecolor='white'
    )

    # Ajustar el diseño para evitar superposiciones
    fig.subplots_adjust(top=0.85)
    plt.show()




plot_candles_with_moving_averages(df, "2024-08-01 01:00:00", "2024-08-15 01:00:00")
```



Este gráfico muestra el comportamiento del precio de Bitcoin junto con tres tipos de medias móviles:

- SMA (Simple Moving Average) en azul,
- EMA (Exponential Moving Average) en naranja,
- WMA (Weighted Moving Average) en verde,

Todas calculadas con una ventana de 14 periodos (una hora por vela) entre el 1 y el 15 de agosto de 2024.

Estas medias móviles son indicadores técnicos fundamentales para analizar la dirección de la tendencia. En este ejemplo observamos lo siguiente:

- En fases bajistas, como entre el 1 y el 4 de agosto, el precio se sitúa por debajo de las medias móviles, confirmando la debilidad del mercado.
- En fases alcistas, como del 4 al 8 de agosto, el precio se coloca por encima de las medias móviles, indicando impulso comprador.
- Los cruces entre medias (por ejemplo, cuando una media más rápida como la EMA cruza hacia arriba una media más lenta como la SMA) suelen anticipar posibles cambios de tendencia. Estos cruces son señales utilizadas comúnmente en estrategias de trading técnico.

Este análisis servirá de base para estudiar otros indicadores técnicos en el mismo rango temporal, como el RSI, MACD o ADX, que nos permitirán complementar la interpretación del comportamiento del precio y validar o reforzar estas señales de cambio de tendencia.

4. Indicadores técnicos MACD. parámetro = 14

El MACD es un indicador que combina señales de tendencia y de momento. Se compone de dos líneas: la línea MACD (calculada como la diferencia entre dos medias móviles exponenciales, típicamente de 12 y 26 periodos) y la línea de señal (media exponencial de 9 periodos de la línea MACD). Además, se representa un histograma que muestra la diferencia entre ambas líneas.

- Cuando la línea MACD cruza por encima de la línea de señal, se interpreta como una posible señal de compra.
- Cuando la línea MACD cruza por debajo de la línea de señal, se interpreta como una posible señal de venta.
- El histograma permite anticipar estos cruces al reflejar cambios en la aceleración del precio.

En el gráfico se observa, por ejemplo, un cruce alcista del MACD a partir del 7 de agosto, lo que anticipa un cambio de tendencia al alza en el precio del activo.

5. Indicadores técnicos RSI. parámetro = 14

El RSI es un oscilador que mide la fuerza relativa del precio, oscilando entre 0 y 100. Su interpretación habitual se basa en dos umbrales:

- Valores por encima de 70 indican una situación de sobrecompra, lo que puede preceder una corrección o retroceso en el precio.
- Valores por debajo de 30 indican una situación de sobreventa, lo que puede anticipar un rebote al alza.

Durante las caídas del 1 al 4 de agosto, el RSI se aproxima al umbral de 30, indicando agotamiento bajista. Posteriormente, a medida que el RSI sube por encima de dicho umbral, se anticipa un posible cambio de tendencia hacia una fase más alcista.

6. Indicadores técnicos Bandas de Bollinger

Las Bandas de Bollinger son un indicador de volatilidad que consta de tres componentes:

- Una media móvil simple (generalmente de 20 periodos).
- Una banda superior (media + 2 desviaciones estándar).
- Una banda inferior (media - 2 desviaciones estándar).

Este indicador permite identificar:

- Zonas de sobreventa cuando el precio toca o atraviesa la banda inferior.
- Zonas de sobrecompra cuando el precio se aproxima o supera la banda superior.
- Fases de contracción de las bandas, que suelen preceder a movimientos bruscos en el precio.
- Fases de expansión, que confirman periodos de alta volatilidad.

En el gráfico puede observarse una expansión de las bandas entre el 6 y el 9 de agosto, coincidiendo con un fuerte movimiento alcista. Asimismo, el precio tocó la banda inferior durante la caída previa, anticipando el posterior rebote.

7. Patrón de reversión bajista: CDLEVENINGSTAR (Evening Star)

```
import pandas as pd
import mplfinance as mpf
```

```
def plot_pattern_candles(df, pattern_column, start_date, end_date, candles_before=30, candles_af
ter=30):
    """
    Grafica las velas previas y posteriores a la detección de un patrón específico.

    Parámetros:
    - df: DataFrame con datos de velas (debe incluir 'timestamp', 'open', 'high', 'low', 'close'
, y columna del patrón).
    - pattern_column: Nombre de la columna que indica la detección del patrón (valores distintos
de 0 indican un patrón detectado).
    - start_date: Fecha de inicio del período de análisis (formato 'YYYY-MM-DD HH:MM:SS').
    - end_date: Fecha de fin del período de análisis (formato 'YYYY-MM-DD HH:MM:SS').
    - candles_before: Número de velas anteriores al patrón a mostrar (por defecto 10).
    - candles_after: Número de velas posteriores al patrón a mostrar (por defecto 10).
    """
    # Filtrar el DataFrame por el rango de fechas
    df_filtered = df[(df.index >= start_date) & (df.index <= end_date)]

    if df_filtered.empty:
        print("No hay datos en el rango especificado.")
        return

    # Identificar índices donde se detecta el patrón
    pattern_indices = df_filtered[df_filtered[pattern_column] != 0].index

    if pattern_indices.empty:
        print("No se detectaron patrones en el rango especificado.")
        return

    # Convertir a Timedelta para hacer operaciones con el índice de tipo Timestamp
    start_idx = df.index.get_loc(pattern_indices[0]) - candles_before
    end_idx = df.index.get_loc(pattern_indices[0]) + candles_after

    # Asegurarse de que no sobrepasemos los límites del DataFrame
    start_idx = max(start_idx, 0)
    end_idx = min(end_idx, len(df) - 1)

    # Obtener el DataFrame de velas previas y posteriores
    df_subset = df.iloc[start_idx:end_idx + 1]

    # Obtener la fecha de la vela donde se detectó el patrón aleatorio
    detected_pattern_date = pattern_indices[0]

    # Crear el gráfico
    title = f"Patrón {pattern_column} detectado en {detected_pattern_date}"
    mpf.plot(
        df_subset,
        type='candle',
        style='yahoo',
        title=title,
        ylabel='Precio',
        ylabel_lower='Volumen',
        volume=False,
        figsize=(10, 6),
    )

 plot_pattern_candles(df, pattern_column='CDLEVENINGSTAR', start_date='2024-06-21 00:00:00', end
_date = '2024-06-25 00:00:00'
```

8. Patrón de reversión alcista: CDLHAMMER (Hammer)

```
 plot_pattern_candles(df, pattern_column='CDLHAMMER', start_date='2024-11-01 00:00:00', end_date
= '2024-11-04 00:00:00'
```

9. Patrón de continuación: CDLRISEFALL3METHODS (Rising/Falling Three Methods)

```
 plot_pattern_candles(df, pattern_column='CDLRISEFALL3METHODS', start_date='2023-01-31 00:00:00'
, end_date = '2023-02-15 00:00:00'
```

10. Patrón de indecisión en el mercado: CDLDOJI (Doji)

```
 plot_pattern_candles(df, pattern_column='CDLDOJI', start_date='2024-11-01 13:00:00', end_date = '2024-11-04 00:00:00'
```

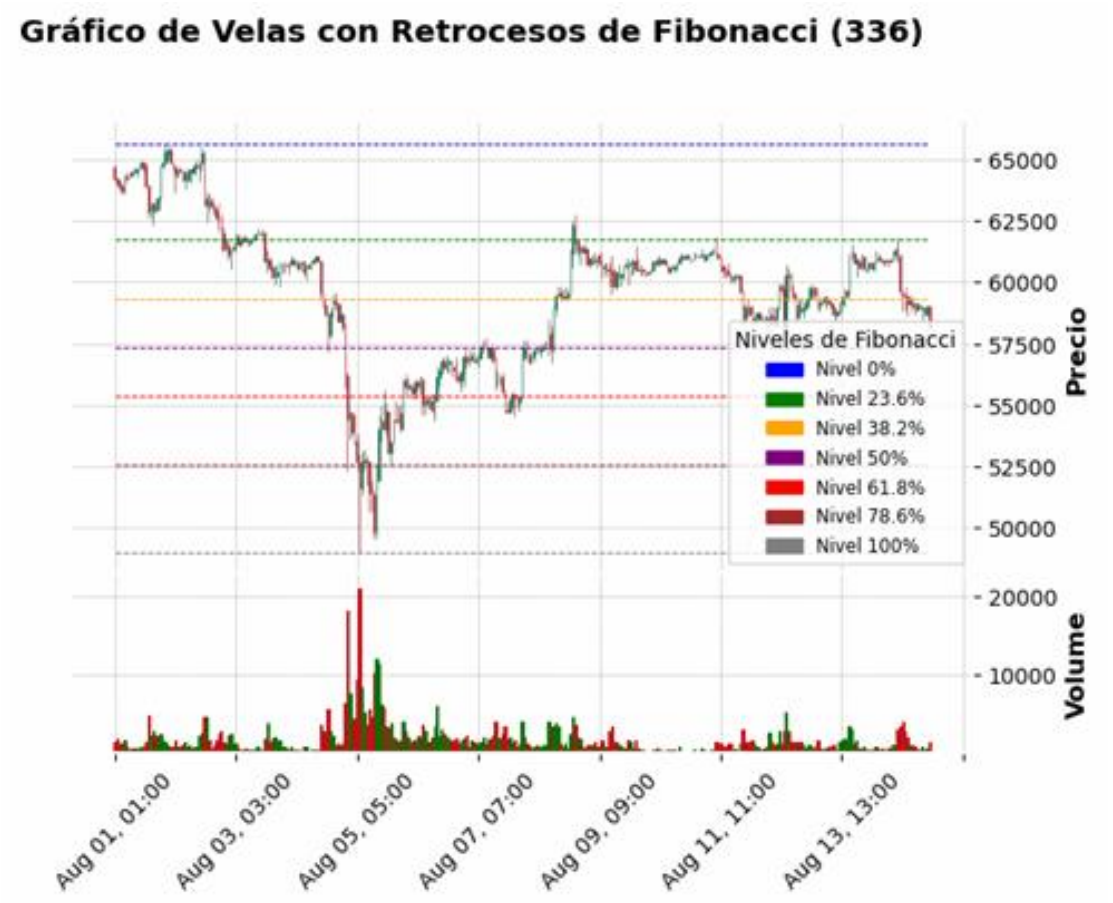
11. Patrón de fuerza/Gaps: CDLTASUKIGAP (Tasuki Gap)

```
 plot_pattern_candles(df, pattern_column='CDLTASUKIGAP', start_date='2021-05-15 13:00:00', end_date = '2021-05-25 00:00:00'
```

12. Retrocesos de Fibonacci y Volumen: Parámetro 336

```
def plot_fibonacci_with_candlesticks(df, start_date, end_date, fib_window):
    '''
    Grafica retrocesos de Fibonacci junto con un gráfico de velas japonesas para un periodo de t
iempo específico.

    Args:
    df (pd.DataFrame): DataFrame con columnas ['timestamp', 'open', 'high', 'low', 'close', 'vol
ume'] y niveles Fibonacci.
    start_time (str): Fecha y hora de inicio en formato 'YYYY-MM-DD HH:MM:SS'.
    end_time (str): Fecha y hora de fin en formato 'YYYY-MM-DD HH:MM:SS'.
    fib_window (int): Ventana de retroceso Fibonacci a graficar (debe coincidir con las calculad
as en `add_multiple_fibonacci_retracements`).

    Returns:
    None: Muestra el gráfico.
    '''
    # Filtra el DataFrame para el rango de tiempo especificado
    # Filtrar el DataFrame por el rango de fechas
    df_filtered = df[(df.index >= start_date) & (df.index <= end_date)]

    if df_filtered.empty:
        raise ValueError("No hay datos en el rango de tiempo especificado.")

    # Niveles de Fibonacci para la ventana específica
    fib_levels = [
        (f'fib_0_{fib_window}', 'blue', 'Nivel 0%'),
        (f'fib_23.6_{fib_window}', 'green', 'Nivel 23.6%'),
        (f'fib_38.2_{fib_window}', 'orange', 'Nivel 38.2%'),
        (f'fib_50_{fib_window}', 'purple', 'Nivel 50%'),
        (f'fib_61.8_{fib_window}', 'red', 'Nivel 61.8%'),
        (f'fib_78.6_{fib_window}', 'brown', 'Nivel 78.6%'),
        (f'fib_100_{fib_window}', 'gray', 'Nivel 100%'),
    ]

    # Configuración del gráfico de velas
    df_filtered['Date'] = pd.to_datetime(df_filtered['timestamp'])
    df_filtered.set_index('Date', inplace=True)

    # Prepara el DataFrame para mplfinance
    ohlc = df_filtered[['open', 'high', 'low', 'close', 'volume']]

    # Crear una lista de líneas para los niveles de Fibonacci
    fib_lines = []
    legend_patches = []  # Para construir la leyenda
    for level, color, label in fib_levels:
```

```
        if level in df_filtered.columns:
            fib_value = df_filtered[level].iloc[-1]  # Último valor de Fibonacci en el período
            fib_lines.append(
                mpf.make_addplot([fib_value] * len(df_filtered), color=color, linestyle='--', wi
dth=0.7)
            )
            # Añadir a la leyenda
            legend_patches.append(mpatches.Patch(color=color, label=label))

    # Crear el gráfico con niveles de Fibonacci
    fig, ax = mpf.plot(
        ohlc,
        type='candle',
        style='charles',
        title=f"Gráfico de Velas con Retrocesos de Fibonacci ({fib_window})",
        ylabel="Precio",
        addplot=fib_lines,  # Añade las líneas de Fibonacci al gráfico
        volume=True,
        returnfig=True  # Necesario para modificar el gráfico después
    )

    # Añadir la leyenda
    ax[0].legend(handles=legend_patches, loc='lower right', fontsize='small', title="Niveles de
Fibonacci")


# Ejemplo de uso
start_date = '2018-08-17 04:00:00'
end_date = '2018-08-26 14:00:00'
start_date = '2024-08-01 01:00:00'
end_date = '2024-08-15 01:00:00'
fib_window = 336  # Ventana de retrocesos a graficar

plot_fibonacci_with_candlesticks(df, start_date, end_date, fib_window)
```



El gráfico presentado muestra la evolución del precio de Bitcoin en velas de 1 hora, acompañado por los retrocesos de Fibonacci calculados sobre un intervalo de 336 períodos (equivalente a 14 días). Este tipo de herramienta técnica es utilizada para identificar posibles niveles de soporte y resistencia basados en la secuencia de Fibonacci, una técnica común en el análisis técnico.

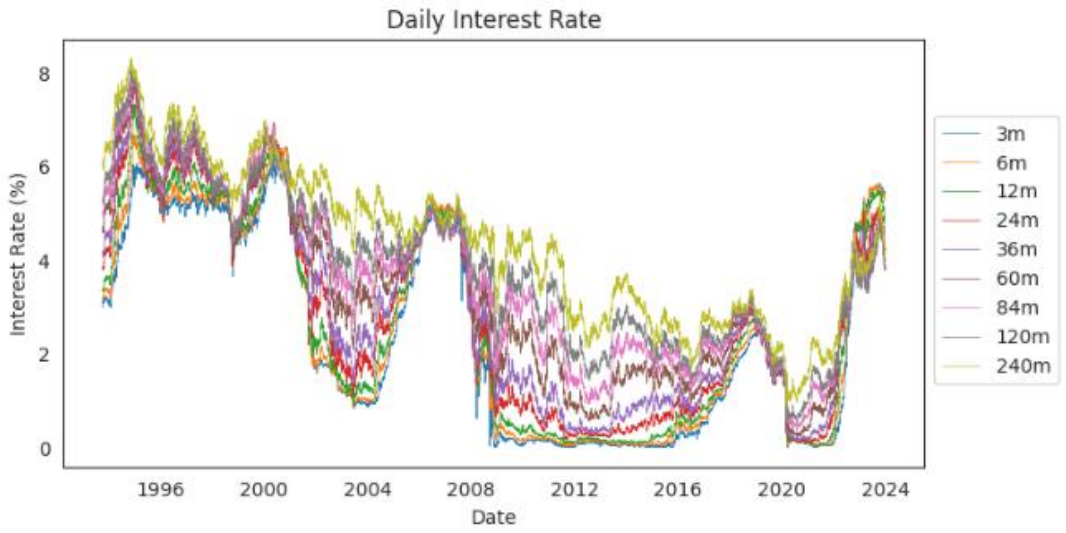
En este caso, los retrocesos se calculan entre un máximo y un mínimo significativos en el rango temporal seleccionado. Las líneas horizontales representan los niveles de retroceso estándar: 0%, 23.6%, 38.2%, 50%, 61.8%, 78.6% y 100%. Estos niveles ayudan a los analistas a anticipar posibles zonas donde el precio puede encontrar obstáculos durante una corrección o un impulso.

Como se observa en el gráfico, tras una tendencia bajista inicial, el precio de Bitcoin rebota desde el mínimo y comienza a subir, encontrando resistencia cerca del nivel del 61.8% y consolidando en la zona del 38.2%-50%. Estos niveles actúan como “zonas de interés”, donde los participantes del mercado suelen tomar decisiones basadas en la confluencia de señales técnicas.

Este tipo de análisis es útil para establecer puntos de entrada o salida, así como para gestionar el riesgo. Sin embargo, conviene recordar que los retrocesos de Fibonacci no garantizan movimientos futuros, sino que funcionan mejor como herramienta complementaria dentro de un enfoque técnico más amplio.

13. Evolución de los tipos de interés

```
def plot_rates(df):
 # Configurar la columna 'Date' como índice para facilitar el trazado
 df.set_index('Date', inplace=True)
 # Crear el gráfico
 plt.figure(figsize=(12, 6))
 for column in df.columns:
 plt.plot(df.index, df[column], label=column)
 # Personalizar el gráfico
 plt.title("Daily Interest Rate", fontsize=14)
 plt.xlabel("Date", fontsize=12)
 plt.ylabel("Interest Rate (%)", fontsize=12)
 plt.legend(title="Maturity", loc="upper right", fontsize=10)
 plt.grid(True, linestyle='--', alpha=0.6)
 # Mostrar el gráfico
 plt.tight_layout()
 plt.show()
 plot_rates(df_tipos_rellenos)
```



14. Evolución VIX(Sp500)

```
 def plot_vix(df):
 # Configurar la columna 'Date' como índice para facilitar el trazado
```

```
# Crear el gráfico
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['CLOSE_VIX'], label='CLOSE_VIX')
# Personalizar el gráfico
plt.title("Evolución VIX", fontsize=14)
plt.xlabel("Date", fontsize=12)
plt.ylabel("Interest Rate (%)", fontsize=12)
plt.legend(title="Maturity", loc="upper right", fontsize=10)
plt.grid(True, linestyle='--', alpha=0.6)
# Mostrar el gráfico
plt.tight_layout()
plt.show()
plot_vix(df)
```