

**UNIVERSIDAD POLITÉCNICA DE MADRID**

**ESCUELA TÉCNICA SUPERIOR  
DE INGENIEROS DE TELECOMUNICACIÓN**



**GRADO EN INGENIERÍA DE  
TECNOLOGÍAS Y SERVICIOS DE  
TELECOMUNICACIÓN  
TRABAJO FIN DE GRADO**

**Desarrollo de un autómata para la gestión  
de operaciones en el mercado financiero  
en tiempo real**

**ÁLVARO RODRÍGUEZ GABALDÓN  
2025**



## **GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN**

### **TRABAJO FIN DE GRADO**

**Título:** Desarrollo de un autómata para la gestión de operaciones en el mercado financiero en tiempo real

**Autor:** D. Álvaro Rodríguez Gabaldón

**Tutor:** Dr. Eduardo López Gonzalo

**Ponente:** D. .....

**Departamento:** Departamento de Señales, Sistemas y Radiocomunicaciones.

### **MIEMBROS DEL TRIBUNAL**

**Presidente:** D. .....

**Vocal:** D. .....

**Secretario:** D. .....

**Suplente:** D. .....

Los miembros del tribunal arriba nombrados acuerdan otorgar la calificación de:

.....

Madrid, a ..... de ..... de 20...

**UNIVERSIDAD POLITÉCNICA DE MADRID**

**ESCUELA TÉCNICA SUPERIOR  
DE INGENIEROS DE TELECOMUNICACIÓN**



**GRADO EN INGENIERÍA DE  
TECNOLOGÍAS Y SERVICIOS DE  
TELECOMUNICACIÓN  
TRABAJO FIN DE GRADO**

**Desarrollo de un autómata para la gestión de  
operaciones en el mercado financiero en  
tiempo real**

**Álvaro Rodríguez Gabaldón  
2025**



## RESUMEN

El proyecto consiste básicamente en el desarrollo e implementación de un autómata para la gestión de operaciones financieras en tiempo real, lo que se conoce también como *trading* algorítmico. La implementación será llevada a cabo mediante el lenguaje de programación Python, y llevaremos a cabo estas operaciones en la Trader Workstation (TWS) o alternativamente Interactive Brokers Gateway, todo esto a través de la API de Interactive Brokers (IBKR). El sistema recopilará datos en tiempo real y los procesará para generar un Swing Chart a partir de un Bar Chart, que es la unidad básica de información con la que nosotros vamos a trabajar, detectando patrones como dobles máximos y mínimos. Basándose en estas señales, ejecutará órdenes de compra y venta con una estrategia de gestión de riesgo, utilizando órdenes de tipo stop y limit para optimizar beneficios y minimizar pérdidas. Además, el autómata actualizará continuamente los datos históricos, extrayendo nuevos minutos desde IBKR y estructurándolos en grupos para su análisis. Este enfoque busca mejorar la eficiencia operativa y la toma de decisiones en el *trading* algorítmico.

## SUMMARY

The project essentially consists of the development and implementation of an automaton for managing financial operations in real time, also known as algorithmic trading. The implementation will be carried out using the Python programming language, and trading operations will take place through Trader Workstation (TWS) or, alternatively, Interactive Brokers Gateway, all via the Interactive Brokers (IBKR) API. The system will collect real-time data and process it to generate a Swing Chart from a Bar Chart, which is the basic unit of information we will work with, detecting patterns such as double tops and double bottoms. Based on these signals, it will execute buy and sell orders using a risk management strategy, utilizing stop and limit orders to optimize gains and minimize losses. Additionally, the automaton will continuously update historical data by retrieving new minutes from IBKR and structuring them into groups for analysis. This approach aims to improve operational efficiency and decision-making in algorithmic trading.

## **PALABRAS CLAVE**

Trading algorítmico, predicción de mercados financieros, automatización, análisis en tiempo real, *backtesting*.

## **KEYWORDS**

Algorithmic trading, financial market forecasting, automation, real-time analysis, backtesting.

## ÍNDICE DEL CONTENIDO

<b>1. INTRODUCCIÓN Y OBJETIVOS.....</b>	<b>1</b>
1.1. Introducción .....	1
1.2. Objetivos .....	3
<b>2. DESARROLLO.....</b>	<b>5</b>
<b>3. RESULTADOS.....</b>	<b>33</b>
<b>4. CONCLUSIONES Y LÍNEAS FUTURAS.....</b>	<b>38</b>
4.1. Conclusiones .....	38
4.2. Líneas futuras.....	38
<b>5. BIBLIOGRAFÍA .....</b>	<b>40</b>
<b>ANEXO A: ASPECTOS ÉTICOS, ECONÓMICOS, SOCIALES Y AMBIENTALES .....</b>	<b>40</b>
A.1 impacto social .....	41
A.2 impacto económico .....	41
A.3 impacto medioambiental .....	42
A.4 responsabilidad ética y profesional .....	42
<b>ANEXO B: PRESUPUESTO ECONÓMICO .....</b>	<b>43</b>
<b>ANEXO C:</b> <b>PROCESADO.PY.....</b>	<b>44</b>
<b>ANEXO D:</b> <b>ESTRATEGIA_LIMIT.PY.....</b>	<b>50</b>
<b>ANEXO E:</b> <b>ESTRATEGIA_STOP.PY.....</b>	<b>55</b>
<b>ANEXO F:</b> <b>MAIN_SECUENCIAL.PY.....</b>	<b>61</b>
<b>ANEXO G:</b> <b>MAIN_MULTIPROCESSING.PY.....</b>	<b>67</b>
<b>ANEXO H:</b> <b>MAIN_VISUALCAHRT_SECUENCIAL.PY.....</b>	<b>72</b>
<b>ANEXO I:</b> <b>MAIN_VISUALCHART_MULTIPROCESSING.PY.....</b>	<b>78</b>
<b>ANEXO J:</b> <b>VARIABLES_POR_CONTRATOS.PY.....</b>	<b>83</b>

<b>ANEXO K:</b>	
<b>OBTENER_CURVAS.PY.....</b>	<b>84</b>
<b>ANEXO L:</b>	
<b>VARIABLES_BACKTEST.PY.....</b>	<b>94</b>
<b>ANEXO M:</b>	
<b>PINTAR_CURVAS.PY.....</b>	<b>95</b>

# 1. INTRODUCCIÓN Y OBJETIVOS

## 1.1. INTRODUCCIÓN

El *trading* financiero constituye una forma dinámica de interacción entre inversores y los distintos mercados, en la que se busca obtener rentabilidad mediante la compra y venta de activos. Para dar respuesta a las diferentes necesidades, perfiles y estrategias de los operadores, se han desarrollado diversas modalidades de trading. Entre ellas destacan el *trading* clásico, el *trading* algorítmico y el *trading* cuantitativo, cada uno con características propias y enfoques particulares [3].

El *trading* clásico, también conocido como *trading* discrecional, se basa fundamentalmente en la aplicación del análisis técnico y la interpretación directa de los gráficos y datos de mercado por parte del inversor. Las decisiones de compra o venta se toman en función de indicadores como las medias móviles, el volumen o ratios financieros, y requieren de la experiencia y criterio del operador. Este tipo de *trading* implica un componente subjetivo importante, ya que el juicio personal del trader influye en la valoración de las oportunidades y en la gestión de las posiciones abiertas.

El *trading* algorítmico consiste en el uso de algoritmos y programas informáticos que ejecutan estrategias predefinidas para analizar el mercado y realizar operaciones. Estos sistemas permiten procesar grandes volúmenes de datos y actuar de forma rápida y precisa, reduciendo la intervención humana y el riesgo de errores asociados al factor emocional. El *trading* algorítmico puede incorporar órdenes automáticas como las *stop loss* o *take profit*, gestionando las operaciones de forma continua incluso fuera del horario habitual de los operadores.

El *trading* cuantitativo se fundamenta en el uso de modelos matemáticos y estadísticos avanzados para identificar patrones y tendencias en los precios de los activos. Este enfoque emplea técnicas como el análisis de volatilidad, correlaciones o modelos predictivos basados en series temporales. A través del *trading* cuantitativo, los inversores buscan adoptar decisiones objetivas, basadas en datos y cálculos precisos, minimizando así el impacto de sesgos subjetivos.

El presente trabajo se enmarca en el ámbito del *trading* algorítmico, una modalidad de operativa financiera que se basa en el uso de algoritmos y programas informáticos para tomar decisiones de inversión y ejecutar operaciones en los mercados. Este tipo de *trading* se caracteriza por la reducción de la intervención humana directa, permitiendo la ejecución de estrategias de forma automática, rápida y consistente, lo que supone una ventaja competitiva frente a los métodos tradicionales. El sistema propuesto está orientado a operar con contratos de futuros, que son instrumentos financieros derivados. Un contrato de futuro es un acuerdo estandarizado entre dos partes para comprar o vender un activo subyacente, por ejemplo, materias primas, divisas o índices bursátiles, en una fecha futura determinada, a un precio pactado en el momento de la firma. Estos contratos se negocian en mercados organizados, lo que aporta transparencia y seguridad jurídica a las transacciones.

En el proyecto se trabaja con futuros negociados en mercados de contratación como **CME Globex**, **NYMEX**, **COMEX**, **CBOT** o **MEFFRV**, pertenecientes a algunas de las principales bolsas de futuros del mundo, como el **CME Group** o el **Grupo BME** (en el caso de MEFFRV). Estos contratos permiten a los operadores gestionar riesgos o especular sobre la evolución del precio de diversos activos. Los contratos utilizados en este proyecto abarcan índices bursátiles, materias primas, metales y divisas, lo que permite aplicar estrategias diversificadas.

A continuación, se describen los principales contratos utilizados:

- **ES (E-mini S&P 500)**: contrato de futuro sobre el índice S&P 500, uno de los índices bursátiles más representativos de la economía estadounidense y de los futuros más líquidos a nivel global.
- **NQ (E-mini Nasdaq 100)**: futuro sobre el índice Nasdaq 100, que agrupa a las principales empresas tecnológicas y de innovación de Estados Unidos.
- **YM (E-mini Dow Jones)**: contrato que replica el comportamiento del índice Dow Jones Industrial Average, compuesto por 30 grandes empresas estadounidenses.
- **RTY (E-mini Russell 2000)**: futuro sobre el índice Russell 2000, formado por empresas de pequeña capitalización de EE.UU., utilizado frecuentemente para estrategias que buscan exposición a este segmento del mercado.
- **CL (Crude Oil)**: contrato de futuro sobre el petróleo crudo, una de las materias primas más importantes del mundo, clave para la economía global.
- **NG (Natural Gas)**: futuro sobre el gas natural, energético estratégico con alta volatilidad y fuerte componente estacional.
- **HO (Heating Oil)**: contrato sobre el gasóleo de calefacción, ligado al sector energético y sujeto a variaciones estacionales y de oferta.
- **GC (Gold)**: futuro sobre el oro, activo refugio clásico en momentos de incertidumbre.
- **HG (Copper)**: contrato de futuro sobre el cobre, metal industrial que actúa como barómetro de la actividad económica mundial.
- **PA (Palladium)**: futuro sobre el paladio, metal usado principalmente en la industria automotriz y electrónica.
- **PL (Platinum)**: contrato sobre el platino, otro metal precioso relevante en la industria y la joyería.
- **EUR (Euro FX)**: futuro sobre el euro frente al dólar estadounidense, muy utilizado en cobertura de riesgos cambiarios y especulación.
- **GBP (British Pound FX)**: contrato sobre la libra esterlina frente al dólar estadounidense.
- **AUD (Australian Dollar FX)**: contrato de futuros sobre el dólar australiano, divisa asociada a economías basadas en recursos naturales.
- **CHF (Swiss Franc FX)**: futuro sobre el franco suizo frente al dólar estadounidense, tradicional activo refugio.
- **JPY (Japanese Yen FX)**: contrato sobre el yen japonés frente al dólar estadounidense.
- **IBEX35**: futuro sobre el índice de referencia de la Bolsa española, que recoge a las 35 empresas más representativas.
- **ZS (Soybean)**: contrato sobre la soja, una de las principales materias primas agrícolas a nivel mundial.
- **ZM (Soybean Meal)**: futuro sobre la harina de soja, subproducto clave en la industria de alimentación animal.
- **ZC (Corn)**: contrato sobre el maíz, producto agrícola fundamental en la alimentación y en la producción de biocombustibles.
- **ZW (Wheat)**: futuro sobre el trigo, otro de los principales cereales básicos de la economía mundial.

Contrato	Tamaño del contrato	Tick mínimo	Valor del tick	Margen/contrato (aprox.)
<b>ES (E-mini S&amp;P 500)</b>	\$50 × índice	0,25	\$12,50	\$10.000
<b>NQ (E-mini Nasdaq 100)</b>	\$20 × índice	0,25	\$5,00	\$40.000
<b>YM (E-mini Dow Jones)</b>	\$5 × índice	1	\$5,00	\$13.000
<b>RTY (E-mini Russell 2000)</b>	\$50 × índice	0,1	\$5,00	\$10.00
<b>CL (Crude Oil)</b>	1.000 barriles	0,01	\$10,00	\$20.000
<b>NG (Natural Gas)</b>	10.000 MMBtu	0,001	\$10,00	\$11.000
<b>HO (Heating Oil)</b>	42.000 galones	0,0001	\$4,20	\$26.000

<b>GC (Gold)</b>	100 onzas troy	0,1	\$10,00	\$20.000
<b>HG (Copper)</b>	25.000 libras	0,0005	\$12,50	\$13.000
<b>PA (Palladium)</b>	100 onzas troy	0,05	\$5,00	\$20.000
<b>PL (Platinum)</b>	50 onzas troy	0,1	\$5,00	\$6.000
<b>EUR (Euro FX)</b>	125.000 €	0,00005	\$6,25	\$5.000
<b>GBP (British Pound FX)</b>	62.500 £	0,00005	\$3,13	\$2.500
<b>AUD (Australian Dollar FX)</b>	100.000 A\$	0,0001	\$10,00	\$2.500
<b>CHF (Swiss Franc FX)</b>	125.000 CHF	0,0001	\$12,50	\$2.500
<b>JPY (Japanese Yen FX)</b>	12,5M JPY	0,0000005	\$6,25	\$4.000
<b>IBEX35</b>	10 € × índice	1	10 €	13.000 €
<b>ZS (Soybeans)</b>	5.000 bushels	0,25	\$12,50	\$3.500
<b>ZM (Soybean Meal)</b>	100 short tons	0,1	\$10,00	\$3.200
<b>ZC (Corn)</b>	5.000 bushels	0,25	\$12,50	\$2.200
<b>ZW (Wheat)</b>	5.000 bushels	0,25	\$12,50	\$3.300

Todos estos datos menos el margen se encuentran en [4] y para el IBEX35 en [5]. Respecto a este último, los márgenes pueden variar según el *broker*, la volatilidad del mercado o los ajustes del *exchange*. Queremos operar con alrededor de \$40.000 por símbolo de contrato de ahí que haya símbolos que operen con más contratos dependiendo del margen por contrato.

El proyecto se apoya en el diseño e implementación de un algoritmo que será responsable de tomar las decisiones de compra y venta de forma autónoma, siguiendo una lógica previamente establecida. Para ello, será necesario definir una serie de parámetros clave que determinarán el comportamiento de la estrategia. Estos parámetros podrán ajustarse y optimizarse en función de los resultados obtenidos durante las pruebas y simulaciones realizadas.

Además, este desarrollo permitirá profundizar en diversos aspectos técnicos relacionados con la conexión a plataformas de intermediación, el manejo de datos en tiempo real y el tratamiento de situaciones imprevistas como interrupciones de servicio o pérdida de conectividad. Se busca no solo crear una herramienta funcional, sino también establecer una base sólida sobre la que se pueda seguir construyendo y evolucionando el proyecto en el futuro, incluyendo la posible incorporación de técnicas más avanzadas como machine learning o redes neuronales para mejorar las decisiones de trading.

## 1.2. OBJETIVOS

El principal objetivo de este proyecto es desarrollar una estrategia automatizada que sea capaz de operar en los mercados financieros, gestionando órdenes sobre diferentes activos de forma eficiente y sin intervención manual directa. Esta estrategia buscará identificar oportunidades en el mercado, ejecutar las operaciones y realizar un seguimiento de su evolución hasta el cierre, cumpliendo con los criterios de gestión del riesgo y de ejecución definidos en el diseño del sistema.

Como objetivos secundarios, se destacan los siguientes aspectos fundamentales para garantizar el correcto funcionamiento y la robustez del sistema:

- **Mantener y actualizar adecuadamente los datos históricos de distintos productos financieros:** Para que el algoritmo pueda tomar decisiones informadas, es esencial disponer de un histórico fiable y bien estructurado. Este histórico permitirá no solo evaluar la estrategia

sobre datos pasados, sino también calcular indicadores y patrones que sirvan de base para la operativa en tiempo real.

- **Recibir y realizar peticiones correctamente con Trader Workstation (TWS) o IB Gateway:** El programa debe ser capaz de establecer y mantener una conexión estable con la plataforma de intermediación seleccionada, en este caso Interactive Brokers, a través de sus herramientas de conexión. Esto incluye la capacidad de enviar solicitudes de datos, recibir respuestas y gestionar el flujo de información de manera eficiente.
- **Colocar adecuadamente las órdenes y obtener información sobre su ejecución:** Parte esencial del sistema es la correcta transmisión de las órdenes al mercado y el seguimiento de su estado, asegurando que se ejecuten según las condiciones especificadas y recogiendo toda la información relacionada (hora de ejecución, precio de entrada y salida, volúmenes, etc.).
- **Gestionar las pérdidas de conexión, asegurando la reconexión y el mantenimiento de la operativa:** Los sistemas de trading automatizado deben estar preparados para gestionar situaciones en las que se pierda la conexión con el bróker o la plataforma de datos. El sistema deberá ser capaz de detectar estas incidencias, reintentar la conexión de forma automática y, en la medida de lo posible, reconstruir el estado de las operaciones activas y pendientes, minimizando el impacto en la operativa.
- **Implementar un entorno de pruebas analizando la actuación de la estrategia a través de los datos históricos:** Antes de poner en marcha el sistema en un entorno real de mercado, es imprescindible validar su funcionamiento mediante simulaciones y backtesting. Este entorno de pruebas permitirá ajustar los parámetros de la estrategia, evaluar su rentabilidad potencial y analizar su comportamiento en distintos escenarios de mercado.
- **Establecer unas bases para un posible futuro desarrollo de este proyecto:** Más allá de cumplir con los objetivos iniciales, este trabajo pretende sentar los cimientos para futuras mejoras y ampliaciones. Entre ellas se podría contemplar la inclusión de nuevas técnicas de análisis, la diversificación hacia otros productos financieros (como opciones o CFDs), el diseño de un módulo de gestión de carteras o la integración con tecnologías avanzadas de inteligencia artificial para la toma de decisiones.

En conjunto, estos objetivos buscan no solo la creación de una herramienta funcional, sino también un sistema robusto, escalable y preparado para evolucionar con las necesidades del operador y del entorno financiero.

## 2. DESARROLLO

### 2.1. ELEMENTOS BÁSICOS DEL ANÁLISIS

Cada uno de estos contratos cotiza en puntos de índice o directamente en el valor del subyacente del activo, cada uno de estos puntos tiene asociado un valor monetario, pues bien teniendo esto en cuenta, la unidad básica de información con la que se va a trabajar serán “barras” correspondientes a cada minuto en los cuales cotiza dicho contrato, estas barras están caracterizadas por el precio con el que abrió dicho minuto (open), el máximo valor alcanzado (high), el mínimo (low), el precio con el que cerró (close) y el volumen (volume), que no trataremos en esta estrategia. Estas barras se agrupan en unos archivos .min que tendremos localizados en nuestro directorio de trabajo, los cuales actualizaremos constantemente. Cada símbolo de contrato con los que trabajaremos tendrá su archivo .min en el que estarán almacenados el histórico de minutos.

### 2.2. LOS CONTRATOS Y LA FORMA DE DEFINIRLOS

Inicialmente definiremos en nuestro script **main.py** la variable CONTRATOS que contendrá la información relevante para poder definirlos.

```
CONTRATOS = [
    {'symbol': 'ES', 'expiry': '202506', 'exchange': 'CME', 'client_id': 1},
    {'symbol': 'NQ', 'expiry': '202506', 'exchange': 'CME', 'client_id': 2},
    {'symbol': 'CL', 'expiry': '202507', 'exchange': 'NYMEX', 'client_id': 3},
    {'symbol': 'GC', 'expiry': '202506', 'exchange': 'COMEX', 'client_id': 4},
    {'symbol': 'IBEX35', 'expiry': '202506', 'exchange': 'MEFFRV', 'client_id': 5},
    {'symbol': 'HG', 'expiry': '202506', 'exchange': 'COMEX', 'client_id': 6},
    {'symbol': 'NG', 'expiry': '202506', 'exchange': 'NYMEX', 'client_id': 7},
    {'symbol': 'HO', 'expiry': '202506', 'exchange': 'NYMEX', 'client_id': 8},
    {'symbol': 'YM', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 9},
    {'symbol': 'RTY', 'expiry': '202506', 'exchange': 'CME', 'client_id': 10},
    {'symbol': 'JPY', 'expiry': '202506', 'exchange': 'CME', 'client_id': 11},
    {'symbol': 'EUR', 'expiry': '202506', 'exchange': 'CME', 'client_id': 12},
    {'symbol': 'GBP', 'expiry': '202506', 'exchange': 'CME', 'client_id': 13},
    {'symbol': 'AUD', 'expiry': '202506', 'exchange': 'CME', 'client_id': 14},
    {'symbol': 'CHF', 'expiry': '202506', 'exchange': 'CME', 'client_id': 15},
    {'symbol': 'PA', 'expiry': '202506', 'exchange': 'NYMEX', 'client_id': 16},
    {'symbol': 'PL', 'expiry': '202507', 'exchange': 'NYMEX', 'client_id': 17},
    {'symbol': 'ZS', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 18},
    {'symbol': 'ZM', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 19},
    {'symbol': 'ZC', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 20},
    {'symbol': 'ZW', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 21},
]
```

Figura 2.1

El símbolo, fecha de expiración, exchange y client\_id correspondiente a cada uno.

A continuación desarrollaremos la función `configurar_contrato(symbol)`:

```

def configurar_contrato(symbol):
    # Diccionario completo con configuración por contrato
    config = {
        'ES': {'umbral': 2.5, 'margen': 2000, 'contratos_orden': 2, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago'},
        'NQ': {'umbral': 10.0, 'margen': 4000, 'contratos_orden': 1, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago'},
        'YM': {'umbral': 20.0, 'margen': 13000, 'contratos_orden': 3, 'variacion_minima': 1.0, 'R': 3, 'num_decimales': None, 'zona_horaria': 'America/Chicago'},
        'RTY': {'umbral': 1.0, 'margen': 10000, 'contratos_orden': 4, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/Chicago'},
        'JPY': {'umbral': 1.0, 'margen': 4000, 'contratos_orden': 10, 'variacion_minima': 0.00001, 'R': 3, 'num_decimales': 6, 'zona_horaria': 'America/Chicago'},
        'EUR': {'umbral': 0.0002, 'margen': 5000, 'contratos_orden': 8, 'variacion_minima': 0.0005, 'R': 3, 'num_decimales': 5, 'zona_horaria': 'America/Chicago'},
        'GBP': {'umbral': 0.0002, 'margen': 2500, 'contratos_orden': 16, 'variacion_minima': 0.0001, 'R': 3, 'num_decimales': 4, 'zona_horaria': 'America/Chicago'},
        'AUD': {'umbral': 0.0002, 'margen': 2500, 'contratos_orden': 16, 'variacion_minima': 0.00005, 'R': 3, 'num_decimales': 5, 'zona_horaria': 'America/Chicago'},
        'CHF': {'umbral': 0.0002, 'margen': 5000, 'contratos_orden': 8, 'variacion_minima': 0.00005, 'R': 3, 'num_decimales': 5, 'zona_horaria': 'America/Chicago'},
        'CL': {'umbral': 0.03, 'margen': 2000, 'contratos_orden': 2, 'variacion_minima': 0.01, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/New_York'},
        'NG': {'umbral': 0.002, 'margen': 11000, 'contratos_orden': 3, 'variacion_minima': 0.001, 'R': 3, 'num_decimales': 3, 'zona_horaria': 'America/New_York'},
        'HO': {'umbral': 0.0002, 'margen': 26000, 'contratos_orden': 1, 'variacion_minima': 0.0001, 'R': 3, 'num_decimales': 4, 'zona_horaria': 'America/New_York'},
        'GC': {'umbral': 15.0, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/New_York'},
        'PA': {'umbral': 5.0, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.5, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/New_York'},
        'PL': {'umbral': 5.0, 'margen': 6000, 'contratos_orden': 6, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/New_York'},
        'HG': {'umbral': 0.02, 'margen': 13000, 'contratos_orden': 3, 'variacion_minima': 0.0005, 'R': 3, 'num_decimales': 4, 'zona_horaria': 'America/New_York'},
        'ZS': {'umbral': 5.0, 'margen': 3500, 'contratos_orden': 10, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago'},
        'ZM': {'umbral': 1.0, 'margen': 3200, 'contratos_orden': 11, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/Chicago'},
        'ZC': {'umbral': 2.0, 'margen': 2200, 'contratos_orden': 18, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago'},
        'ZW': {'umbral': 2.0, 'margen': 3300, 'contratos_orden': 11, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago'},
        'IBEX35': {'umbral': 30.0, 'margen': 13000, 'contratos_orden': 3, 'variacion_minima': 1.0, 'R': 3, 'num_decimales': None, 'zona_horaria': 'Europe/Madrid'}
    }

    # Obtenemos la configuración o valores por defecto si el símbolo no existe
    contrato = config.get(symbol, [
        'umbral': 1.0,
        'margen': 10000,
        'contratos_orden': 1,
        'variacion_minima': 0.01, # Valor por defecto para variación mínima
        'R': 3,
        'num_decimales': 2, # Valor por defecto para número de decimales
        'zona_horaria': 'America/New_York' # Valor por defecto para zona horaria
    ])

    # Devolvemos todos los parámetros necesarios
    return (
        contrato['umbral'],
        contrato['contratos_orden'],
        contrato['variacion_minima'],
        contrato['R'],
        contrato['num_decimales'],
        contrato['zona_horaria']
    )
}

```

Figura 2.2

Esta función nos devuelve datos de cada contrato que se utilizarán para interactuar con Interactive Brokers y nuestra estrategia. Cada operación involucrará un número determinado de contratos por cada futuro (contratos\_orden), el margen es el saldo por contrato que debes tener disponible para poder operar, en caso de que esto no se cumpla IB te lo notificará y la orden será cancelada automáticamente, en el momento que vuelvas a tener saldo suficiente volverás a operar con normalidad. También tendremos el script `variables_por_contrato.py`

```

variables_por_contrato = defaultdict(lambda: {
    'orden_activa': {
        'enviada': False,
        'entry_order': None,
        'tipo': None,
        'tp_order': None,
        'sl_order': None,
        'take_profit_price': None,
        'stop_loss_price': None,
        'en_espera': False,
    },
    'quantity': 0,
    'umbral': 0.0,
    'variacion_minima': 0.0,
    'R': 0.0,
    'num_decimales': 0,
    'zona_horaria': None,
})

def obtener_variables(identificador):
    return variables_por_contrato[identificador]

```

Figura 2.3

De esta manera en el main:

```
contratos_ib = [Future(c['symbol'], c['expiry'], c['exchange']) for c in CONTRATOS]
vars_contratos = {c['symbol']: variables_por_contrato(c['symbol']) for c in CONTRATOS}

for config, contract in zip(CONTRATOS, contratos_ib):
    symbol = config['symbol']
    umbral, cantidad, variacion_minima, r, num_decimales, zona_horaria = configurar_contrato(config['symbol'])
    vars_contratos[symbol].update({'umbral': umbral, 'quantity': cantidad, 'variacion_minima': variacion_minima, 'R': r, 'num_decimales': num_decimales, 'zona_horaria': zona_horaria})
    file_path = f"{symbol}.min"
```

Figura 2.4

Definiremos e inicializaremos los distintos contratos.

### 2.3. ACTUALIZACIÓN DE LOS DATOS HISTÓRICOS

Como paso siguiente hacemos un script llamado **procesado.py** en el que definiremos todas las funciones encargadas de manipular los archivos .min. La primera en nuestra cadena de procesos será `actualizar_archivo_min(ib, file_path, contract, zona_horaria)`:

```
def actualizar_archivo_min(ib, file_path, contract, zona_horaria):
    """Actualiza el archivo .min con nuevos datos."""

    def leer_min(file_path):
        """Lee el último registro del archivo .min y devuelve la fecha más reciente."""
        try:
            with open(file_path, "rb") as f:
                f.seek(-32, 2)
                data = f.read(32)
                if len(data) < 32:
                    return None

                fecha, tiempo, *_ = struct.unpack("ii f f f f ii", data)
                ano = fecha // 500
                aux = fecha % 500
                mes = aux // 32
                dia = aux % 32
                hour = tiempo // 3600
                minute = (tiempo % 3600) // 60
                return datetime(ano, mes, dia, hour, minute, second=59, tzinfo=ZoneInfo(zona_horaria)) # <- Añade second=59
        except FileNotFoundError:
            return None

    def escribir_min(file_path, barras):
        """Escribe nuevas barras en el archivo .min evitando duplicados."""
        with open(file_path, "ab") as f:
            for bar in barras:
                dt = bar.date
                fecha = dt.year * 500 + dt.month * 32 + dt.day
                tiempo = dt.hour * 3600 + dt.minute * 60

                data = struct.pack("ii f f f f ii", fecha, tiempo, bar.open, bar.high, bar.low, bar.close, int(bar.volume), 0)
                f.write(data)

    ultima_fecha = leer_min(file_path)
    print(f"Última fecha en el archivo: {ultima_fecha}")
    if ultima_fecha is None:
        print(f"Error al leer el archivo {file_path}")
        return

    ahora_utc = datetime.now(timezone.utc)
    diferencia_segundos = int((ahora_utc - ultima_fecha.astimezone(timezone.utc)).total_seconds())
    dias = diferencia_segundos // 86400
    segundos_restantes = diferencia_segundos % 86400
    barras_totales = []
```

```

def filtrar_nuevas_barras(barras):
    return [bar for bar in barras if bar.date >= ultima_fecha]

# **Solicitar datos históricos**
if dias > 0:
    bars = ib.reqHistoricalData(
        contract,
        endTime='',
        durationStr=f"{dias} D",
        barSizeSetting='1 min',
        whatToShow='TRADES',
        useRTH=False,
        formatDate=1
    )
    if bars:
        barras_totales.extend(filtrar_nuevas_barras(bars))

if segundos_restantes > 0:
    bars = ib.reqHistoricalData(
        contract,
        endTime='',
        durationStr=f'{segundos_restantes} S',
        barSizeSetting='1 min',
        whatToShow='TRADES',
        useRTH=False,
        formatDate=1
    )
    if bars:
        barras_totales.extend(filtrar_nuevas_barras(bars))

if barras_totales:
    escribir_min(file_path, barras_totales)
    borrar_ultimas_n_lineas_bin(file_path, 1)
    print(f"Se han agregado {len(barras_totales)} registros al archivo .min.")
else:
    print("No hay nuevos registros para agregar.")

```

Figura 2.5

Como parámetros pasaremos una instancia de Interactive Brokers para poder hacer pedida de nuevas barras, la ruta del .min, el contrato y su zona horaria correspondiente. Lo primero será poder saber el último dato del .min para poder pedir a IB las nuevas barras necesarias, a continuación definimos los métodos leer\_min(file\_path) y escribir\_min(file\_path, barras), leemos el último registro del .min y podemos escribir todas las nuevas barras que queramos respectivamente con estas funciones. Lo siguiente será averiguar la diferencia de tiempo entre el instante actual y el último minuto almacenado en el archivo:

```

ahora_utc = datetime.now(timezone.utc)
diferencia_segundos = int((ahora_utc - ultima_fecha.astimezone(timezone.utc)).total_seconds())
dias = diferencia_segundos // 86400
segundos_restantes = diferencia_segundos % 86400
barras_totales = []

```

Figura 2.6

ultima\_fecha será el último minuto almacenado en el .min en la hora local del Exchange, por eso usamos el método astimezone para pasarlo a horario utc y que la comparación con ahora\_utc sea correcta, obtenemos esta diferencia en segundos, hay un problema y es que IB tiene un límite de

cuantos segundos puedes pedir en una única llamada, por lo que dividimos todo esto en días y segundos. Esto es lo que dice la API de IB [1]:

`durationStr ( str )` – Time span of all the bars. Examples: '60 S', '30 D', '13 W', '6 M', '10 Y'.

Figura2.7

Puesto que nuestros .min al principio van a tener registros relativamente recientes nos bastará con días y lo que sobre segundos. Luego utilizaremos solo segundos. Definiremos la `barras_totales` que es lo que escribiremos en el .min finalmente. También definimos `filtrar_nuevas_barras(barras)` para asegurarnos de no tener duplicados. Luego hacemos la pedida de los minutos nuevos. Este es el resumen de todos los parámetros de la pedida [1]:

- `contract ( Contract )` – Contract of interest.
- `endDateTime ( Union [ datetime , date , str , None ] )` – Can be set to " to indicate the current time, or it can be given as a datetime.date or datetime.datetime, or it can be given as a string in 'yyyyMMdd HH:mm:ss' format. If no timezone is given then the TWS login timezone is used.
- `durationStr ( str )` – Time span of all the bars. Examples: '60 S', '30 D', '13 W', '6 M', '10 Y'.
- `barSizeSetting ( str )` – Time period of one bar. Must be one of: '1 secs', '5 secs', '10 secs', '15 secs', '30 secs', '1 min', '2 mins', '3 mins', '5 mins', '10 mins', '15 mins', '20 mins', '30 mins', '1 hour', '2 hours', '3 hours', '4 hours', '8 hours', '1 day', '1 week', '1 month'.
- `whatToShow ( str )` – Specifies the source for constructing bars. Must be one of: 'TRADES', 'MIDPOINT', 'BID', 'ASK', 'BID\_ASK', 'ADJUSTED\_LAST', 'HISTORICAL\_VOLATILITY', 'OPTION\_IMPLIED\_VOLATILITY', 'REBATE\_RATE', 'FEE\_RATE', 'YIELD\_BID', 'YIELD\_ASK', 'YIELD\_BID\_ASK', 'YIELD\_LAST'. For 'SCHEDULE' use `reqHistoricalSchedule()`.
- `useRTH ( bool )` – If True then only show data from within Regular Trading Hours, if False then show all data.
- `formatDate ( int )` – For an intraday request setting to 2 will cause the returned date fields to be timezone-aware datetime.datetime with UTC timezone, instead of local timezone as used by TWS.
- `keepUpToDate ( bool )` – If True then a realtime subscription is started to keep the bars updated; `endDateTime` must be set empty ("") then.
- `chartOptions ( List [ TagValue ] )` – Unknown.
- `timeout ( float )` – Timeout in seconds after which to cancel the request and return an empty bar series. Set to `0` to wait indefinitely.

Figura 2.7

Si `barras_totales` no está vacío añadimos en el .min y borramos el último minuto puesto que a la hora de pedir el último minuto aún está en formación y no se ha cerrado, es decir, los valores de high, low, close aún no están cerrados. Para esto nos servimos de la función `borrar_ultimas_n_lineas_min(file_path, n_lineas)`, que lo que hace es borrar los últimos "n\_lineas" registros:

```

def borrar_ultimas_n_lineas_min(nombre_archivo, n_lineas):
    TAM_REGISTRO = struct.calcsize("ii f f f f ii") # 32 bytes por registro
    """
    Borra las últimas `n_lineas` de un archivo .min manteniendo la estructura binaria.

    Parámetros:
        nombre_archivo (str): Ruta del archivo .min.
        n_lineas (int): Número de registros a eliminar (los más recientes).
    """
    try:
        with open(nombre_archivo, "rb") as f:
            datos = f.read() # Leer todo el archivo en memoria

        total_registros = len(datos) // TAM_REGISTRO # Calcular cuántos registros hay

        if total_registros == 0:
            print("El archivo está vacío, no hay nada que borrar.")
            return

        if n_lineas >= total_registros:
            print("Intentaste borrar más registros de los que existen. Se eliminará todo el archivo.")
            os.remove(nombre_archivo) # Borra completamente el archivo
            return

        # Obtener solo la parte inicial (excluyendo los últimos `n_lineas` registros)
        datos_restantes = datos[: (total_registros - n_lineas) * TAM_REGISTRO]

        # Sobrescribir el archivo con los datos restantes
        with open(nombre_archivo, "wb") as f:
            f.write(datos_restantes)

        #print(f"Se eliminaron las últimas {n_lineas} líneas de {nombre_archivo} correctamente.")

    except Exception as e:
        print(f"Error al borrar líneas: {e}")

```

Figura 2.8

## 2.4. EL SWING CHART

Hasta ahora lo que almacenamos en el .min es lo que se conoce como el “bar chart”, pero sobre lo que nosotros vamos a basar nuestra estrategia es en el “swing chart” [2]. El swing chart se construye identificando los puntos clave donde la tendencia cambia de dirección, filtrando el “ruido” del bar chart. Comienza determinando la dirección inicial (alcista o bajista) según si el primer día cierra arriba o abajo de su apertura. Luego, recorre cada barra comparando máximos (highs) y mínimos (lows) con los anteriores: si rompe el último swing alto en una tendencia bajista, marca un nuevo swing alcista; si rompe el último swing bajo en una tendencia alcista, señala un swing bajista. Los outside days (días con máximos más altos y mínimos más bajos) se tratan como posibles cambios de volatilidad, mientras que los inside days (rangos más estrechos) se ignoran por defecto. El resultado es una serie de líneas que conectan únicamente los máximos y mínimos significativos, simplificando la visualización de la tendencia real. Esta es la función que creamos en **procesado.py**:

```

def swing1(hi, lo, cl, op, dhi, dlo):
    dias = len(hi)
    swings = []
    trasw = []
    calsw = []

    if op[0] < cl[0]:
        swings.extend([lo[0], hi[0]])
        trasw.extend([1, 1])
        calsw.extend([dlo[0], dhi[0]])
        direccion = 1
    else:
        swings.extend([hi[0], lo[0]])
        trasw.extend([1, 1])
        calsw.extend([dhi[0], dlo[0]])
        direccion = -1

    indiceswing = 1

    for i in range(dias - 1):
        if (hi[i + 1] > hi[i]) and (lo[i + 1] < lo[i]): # outside day
            if dlo[i + 1] < dhi[i + 1]:
                if (direccion == 1 and lo[i + 1] < swings[indiceswing]):
                    direccion = -1
                    swings.append(lo[i + 1])
                    trasw.append(i + 1)
                    calsw.append(dlo[i + 1])
                    indiceswing += 1
                elif (direccion == -1 and lo[i + 1] < swings[indiceswing]):
                    swings[indiceswing] = lo[i + 1]
                    trasw[indiceswing] = i + 1
                    calsw[indiceswing] = dlo[i + 1]

                    direccion = 1
                    swings.append(hi[i + 1])
                    trasw.append(i + 1)
                    calsw.append(dhi[i + 1])
                    indiceswing += 1
                else:
                    if (direccion == -1 and hi[i + 1] > swings[indiceswing]):
                        direccion = 1
                        swings.append(hi[i + 1])
                        trasw.append(i + 1)
                        calsw.append(dhi[i + 1])
                        indiceswing += 1

                    elif (direccion == 1 and hi[i + 1] > swings[indiceswing]):
                        swings[indiceswing] = hi[i + 1]
                        trasw[indiceswing] = i + 1
                        calsw[indiceswing] = dhi[i + 1]

                        direccion = -1
                        swings.append(lo[i + 1])
                        trasw.append(i + 1)
                        calsw.append(dlo[i + 1])
                        indiceswing += 1
                    elif hi[i + 1] > hi[i]:
                        if (direccion == -1 and hi[i + 1] > swings[indiceswing]):
                            direccion = 1
                            swings.append(hi[i + 1])
                            trasw.append(i + 1)
                            calsw.append(dhi[i + 1])
                            indiceswing += 1
                        elif (direccion == 1 and hi[i + 1] > swings[indiceswing]):
                            swings[indiceswing] = hi[i + 1]
                            trasw[indiceswing] = i + 1
                            calsw[indiceswing] = dhi[i + 1]
                    elif lo[i + 1] < lo[i]:
                        if (direccion == 1 and lo[i + 1] < swings[indiceswing]):
                            direccion = -1
                            swings.append(lo[i + 1])
                            trasw.append(i + 1)
                            calsw.append(dlo[i + 1])
                            indiceswing += 1
                        elif (direccion == -1 and lo[i + 1] < swings[indiceswing]):
                            swings[indiceswing] = lo[i + 1]
                            trasw[indiceswing] = i + 1
                            calsw[indiceswing] = dlo[i + 1]

    return np.array(swings), np.array(calsw), np.array(trasw)

```

Figura 2.9

A continuación explicamos las entradas de esta función. Para poder obtener este swing hay que manipular el bar chart, para ello hay que obtener todos los datos del .min de la misma manera que hacíamos a la hora de actualizar los datos históricos, solo que en esta ocasión tenemos que sacar todos los registros. Definimos entonces la función leer\_min(file\_path, zona\_horaria) :

```
def leer_min(file_path, zona_horaria):
    TAM_REGISTRO = struct.calcsize("ii f f f f ii") # 32 bytes
    caldayi = []
    tradayi = []
    hora = []
    hii = []
    loi = []
    cli = []
    opi = []
    voli = []

    with open(file_path, "rb") as f:
        f.seek(0, 0) # Asegurar que leemos desde el principio real

        while True:
            data = f.read(TAM_REGISTRO)
            if len(data) < TAM_REGISTRO:
                break # Evita errores si el archivo termina antes

            # Desempaquetar datos
            fecha, time, op, hi, lo, cl, vol, _ = struct.unpack("ii f f f f ii", data)

            # Convertir la fecha al formato datetime
            año = fecha // 500
            aux = fecha % 500
            mes = aux // 32
            dia = aux % 32
            hour = time // 3600
            minute = (time % 3600) // 60

            fecha_datetime = datetime(año, mes, dia, hour, minute, tzinfo=ZoneInfo(zona_horaria))
            caldayi.append(fecha_datetime)
            tradayi.append(len(caldayi))
            opi.append(op)
            hii.append(hi)
            loi.append(lo)
            cli.append(cl)
            voli.append(vol)

    return caldayi, tradayi, hii, loi, cli, opi, voli
```

Figura 2.10

caldayi es una lista que contendrá todas las fechas de las barras, hii los máximos, loi mínimos, opi las entradas en cada minuto, cli la salida. voli y tradayi no los vamos a utilizar.

Ahora la idea es tener una especie de ventana que ataÑe a los últimos registros del .min, esta se irá deslizando a medida que el .min se actualiza, esta ventana se va a dividir en n\_grupos cada grupo contendrá n\_min\_barras, este último valor será uno de los parámetros que vamos a considerar a la hora de optimizar la curva de ganancias acumuladas. De cada grupo calcularemos el mayor high, menor low, open de la barra más antigua, close de la más reciente fecha del máximo de cada grupo (dhi) y mínimo (dlo). Hacemos la función dividir\_min\_en\_grupos\_para\_swings(caldayi, hii, loi, cli, opi, voli, n\_min\_barras, n\_grupos, inicio\_relativo), inicio\_relativo es desde qué punto del .min partimos para formar los grupos siempre desde este hasta los registros más antiguos, en nuestro caso siempre desde el final del .min, es decir el registro más reciente (len(caldayi)-1) :

```

def dividir_min_en_grupos_para_swings(caldayi, hii, loi, cli, opi, voli, n_min_barras, n_grupos, inicio_relativo):
    """
        Divide los datos en grupos hacia atrás y calcula métricas de swing simultáneamente.

    Devuelve:
    - grupos: Lista de diccionarios con los datos crudos de cada grupo
    - hi_swings: Array con los máximos high de cada grupo
    - lo_swings: Array con los mínimos low de cada grupo
    - op_swings: Array con los opens más antiguos (primer elemento)
    - cl_swings: Array con los closes más recientes (último elemento)
    - dhi_swings: Array con las fechas donde ocurrieron los máximos
    - dlo_swings: Array con las fechas donde ocurrieron los mínimos
    """

    num_barras = len(caldayi)
    total_min_deseados = n_min_barras * n_grupos

    # Validaciones
    if num_barras < total_min_deseados:
        raise ValueError("No hay suficientes datos para formar los grupos especificados.")

    if inicio_relativo is None:
        inicio_relativo = num_barras

    if inicio_relativo < total_min_deseados:
        raise ValueError(f"El inicio_relativo ({inicio_relativo}) es insuficiente para {n_grupos} grupos de {n_min_barras} mins.")

    # Inicializamos arrays para los resultados
    hi_swings = np.empty(n_grupos)
    lo_swings = np.empty(n_grupos)
    op_swings = np.empty(n_grupos)
    cl_swings = np.empty(n_grupos)
    dhi_swings = np.empty(n_grupos, dtype=caldayi.dtype)
    dlo_swings = np.empty(n_grupos, dtype=caldayi.dtype)

    fin_segmento = inicio_relativo + 1
    inicio_segmento = fin_segmento - total_min_deseados

    for i in range(n_grupos):
        # Calculamos los índices del grupo actual

        grupo_inicio = i * n_min_barras
        grupo_fin = grupo_inicio + n_min_barras
        # Extraemos el grupo
        hi_grupo = hii[inicio_segmento + grupo_inicio : inicio_segmento + grupo_fin]
        loi_grupo = loi[inicio_segmento + grupo_inicio : inicio_segmento + grupo_fin]

        # Calculamos métricas
        max_hi_idx = np.argmax(hi_grupo)
        min_loi_idx = np.argmin(loi_grupo)

        # Almacenamos resultados
        hi_swings[i] = hi_grupo[max_hi_idx]
        lo_swings[i] = loi_grupo[min_loi_idx]
        op_swings[i] = opi[inicio_segmento + grupo_inicio] # Primer open
        cl_swings[i] = cli[inicio_segmento + grupo_fin - 1] # Último close
        dhi_swings[i] = caldayi[inicio_segmento + grupo_inicio + max_hi_idx]
        dlo_swings[i] = caldayi[inicio_segmento + grupo_inicio + min_loi_idx]

    return hi_swings, lo_swings, op_swings, cl_swings, dhi_swings, dlo_swings

```

Figura 2.11

Y todo esto será la entrada para la función de swings.

## 2.5.LA ESTRATEGIA

A continuación vamos a explicar la estrategia: procesar\_doble\_top\_bottom(swingsy, calswy, umbral, ib, contract, orden\_activa, high\_barra\_actual, low\_barra\_actual, client\_id, cantidad, variacion\_minima, R\_multiplier, num\_decimales, zona\_horaria). La idea es detectar en el swing chart las situaciones de doble top/bottom, comparando los últimos swings de precio almacenados en el array swingsy. Si el último swing es más bajo que el anterior (swingsy[-1] < swingsy[-

2]), busca un posible double bottom verificando si la diferencia entre el mínimo actual y un mínimo anterior (saltando swings intermedios con  $(-2 * k) - 1$ ) está dentro de un umbral predefinido; si se cumple, marca el patrón. Para un posible double top ( $\text{swingsy}[-1] > \text{swingsy}[-2]$ ), realiza la misma lógica pero comparando máximos. En ambos casos, cuando se detecta el patrón, registra el precio (`swing_detectado`) y la hora (`tiempo_deteccion`), deteniendo la búsqueda si la diferencia supera el umbral (evitando falsos positivos). Comparamos el último swing ( $\text{swingsy}[-1]$ ) con swings previos no consecutivos (saltando de 2 en 2 para evitar swings intermedios, "for `k` in range(`1, (lsw - 1) // 2 + 1`)").

```
if lsw >= 3:
    tipo = None
    swing_detectado = None
    tiempo_deteccion = None

    if swingsy[-1] < swingsy[-2]: # posible double bottom
        for k in range(1, (lsw - 1) // 2 + 1):
            dif = swingsy[-1] - swingsy[(-2 * k)-1]
            if abs(dif) < umbral:
                tipo = 'double_bottom'
                swing_detectado = swingsy[-1] # Precio del swing
                tiempo_deteccion = calswy[-1] # Hora de detección (datetime)
                break
            elif dif >= umbral:
                break

    elif swingsy[-1] > swingsy[-2]: # posible double top
        for k in range(1, (lsw - 1) // 2 + 1):
            dif = swingsy[-1] - swingsy[(-2 * k)-1]
            if abs(dif) < umbral:
                tipo = 'double_top'
                swing_detectado = swingsy[-1]
                tiempo_deteccion = calswy[-1]
                break
            elif dif <= -umbral:
                break
```

Ahora tenemos que verificar que no hayan pasado más de 5 minutos desde la detección, para ello pasamos como parámetro a esta función la zona horaria correspondiente. Luego, comprueba si el precio actual (tomando el high actual para double bottom o el low para double top) está dentro de un umbral permitido respecto al swing detectado, evitando entradas en zonas de riesgo. Si ambas condiciones se cumplen, confirma la señal como válida.

```
if tipo and swing_detectado and tiempo_deteccion:
    # 1. Verificar que no hayan pasado más de 5 minutos desde la detección
    tiempo_actual_utc = datetime.now(timezone.utc)
    tiempo_actual_local = tiempo_actual_utc.astimezone(ZoneInfo(zona_horaria)) # Ajustar a la zona horaria deseada
    tiempo_transcurrido = tiempo_actual_local - tiempo_deteccion

    if tiempo_transcurrido > tiempo_max_espera:
        print(f"⚠ {tipo} detectado hace {tiempo_transcurrido}. Demasiado tarde para entrar.")
        return

    # 2. Verificar que el precio actual está dentro del rango permitido
    precio_actual = high_barra_actual if tipo == 'double_bottom' else low_barra_actual
    distancia_al_swing = abs(precio_actual - swing_detectado)

    if distancia_al_swing > umbral:
        print(f"⚠ {tipo} detectado en {swing_detectado}, pero precio actual ({precio_actual}) está fuera de rango.")
        return

    # --- LÓGICA DE ENTRADA (código existente, ahora con validaciones) ---
    print(f"✅ {tipo.replace('_', ' ')} válido. Swing: {swing_detectado}, Precio actual: {precio_actual}")
```

Tras esto vamos a explicar los dos tipos de orden vamos a colocar y dónde. Stop Order es una orden condicional que se convierte en una orden de mercado (ejecutada al precio disponible) cuando se alcanza un precio de activación (stop price), usada principalmente para limitar pérdidas (stop loss)

o entrar en tendencias (stop entry). Limit Order es una orden que solo se ejecuta al precio especificado o mejor, permitiendo control exacto del precio de entrada/salida pero sin garantía de ejecución si el mercado no alcanza ese nivel. Mientras la stop order prioriza la activación sobre el precio final, la limit order prioriza el precio exacto sobre la ejecución inmediata. Si estamos en situación de doble máximo entramos con una `LimitOrder` de compra cuyo precio (`entry_limit_price`) será un tick por debajo del low de la barra actual (`low_barra_actual - variacion_minima`), esta orden se ejecutará si el precio del activo es igual o inferior a este precio de entrada. En el código utilizamos la función `round` para redondear al número de decimales pasado como parámetro (`num_decimales`) porque para algunos contratos, IB puede poner problemas si en la colocación de la orden en el mercado se incluyen más decimales de lo necesario. También definiríamos los precios de salida de las órdenes que completarán nuestra operación, solo se ejecutará una. El `take_profit_price` se colocará un tick por encima del swing detectado y el `stop_loss_price` una cantidad por debajo del `entry_limit_price` (`entry_limit_price - (R_multiplier * (take_profit_price - entry_limit_price))`), `R_multiplier` es otro parámetro que le pasaremos a la función. Los parámetros de definición de la orden son:

- `action`: Indica si es compra (BUY) o venta (SELL).
- `totalQuantity`: Cantidad de acciones o contratos a operar. Que en nuestro caso es igual al parámetro que le pasamos a la función “cantidad”
- `lmtPrice`: Precio límite (la orden solo se ejecuta a este precio o mejor).
- `tif='GTD'`: Time in Force (vigencia de la orden), en este caso Good Till Date (GTD), que expira en una fecha/hora específica (`goodTillDate`).
- `goodTillDate=order_timer`: Fecha/hora de caducidad de la orden (ej: 20240630 23:59:59).
- `outsideRth=True`: Permite que la orden se ejecute fuera del horario regular (extended hours).
- `transmit=False`: Retrasa el envío al mercado (útil para órdenes complejas que requieren ajustes antes de activarse).
- `orderId`: ID único generado para identificar la orden en el sistema.

Esta orden de entrada tendrá un tiempo límite de vida de 5 minutos, pasado ese tiempo si esta sigue activa se cancela automáticamente por parte de IB (`order_timer = (datetime.now() + timedelta(minutes=5)).strftime("%H:%M.%S")`)

```
if tipo == 'double_top':
    action = 'BUY'
    quantity = cantidad
    entry_limit_price = round(low_barra_actual - variacion_minima, num_decimales) # ej: calculado a partir del swing
    take_profit_price = round(swingsy[-1] + variacion_minima, num_decimales)
    stop_loss_price = round(entry_limit_price - (R_multiplier * (take_profit_price - entry_limit_price)), num_decimales)
    # Orden de entrada
    orden_entrada = LimitOrder(
        action=action,
        totalQuantity=quantity,
        lmtPrice=entry_limit_price,
        tif='GTD',
        goodTillDate=order_timer,
        outsideRth=True,
        transmit=False,
        orderId=ib.client.getReqId()
    )
```

En el caso de doble mínimo entraremos con una `LimitOrder` de venta un tick por encima del high actual, esta se ejecutará si el precio del activo es igual o mayor que el precio de entrada de nuestra orden, el take profit será un tick por debajo del swing actual y el stop loss (`R_multiplier * (take_profit_price - entry_limit_price)`) por encima del `entry_limit_price`.

```

elif tipo == 'double_bottom':
    action = 'SELL'
    quantity = cantidad
    entry_limit_price = round(high_barra_actual + variacion_minima, num_decimales)
    take_profit_price = round(swingsy[-1] - variacion_minima, num_decimales)
    stop_loss_price = round(entry_limit_price + (R_multiplier * (entry_limit_price - take_profit_price)), num_decimales)
    # Orden de entrada
    orden_entrada = LimitOrder(
        action=action,
        totalQuantity=quantity,
        lmtPrice=entry_limit_price,
        tif='GTD',
        goodTillDate=order_timer,
        outsideRth=True,
        transmit=False,
        orderId=ib.client.getReqId()
    )

```

Ahora hay que definir las órdenes take profit y stop loss que son las responsables de cerrar la operación.

```

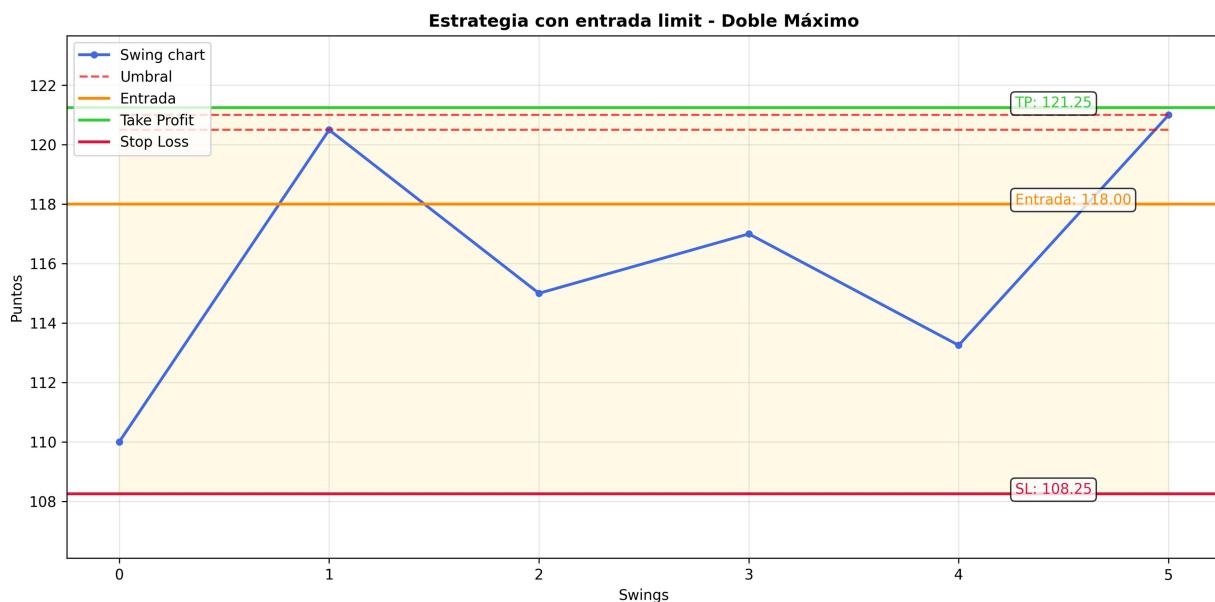
tp = LimitOrder(
    action='SELL' if action == 'BUY' else 'BUY',
    totalQuantity=quantity,
    lmtPrice=take_profit_price, # Precio take-profit
    outsideRth=True,
    parentId=orden_entrada.orderId,
    transmit=False,
    orderId=ib.client.getReqId()
)

sl = StopOrder(
    action='SELL' if action == 'BUY' else 'BUY',
    totalQuantity=quantity,
    stopPrice=stop_loss_price, # Precio stop-loss
    outsideRth=True,
    parentId=orden_entrada.orderId,
    transmit=True, # Última orden del bracket debe ser True
    orderId=ib.client.getReqId()
)

```

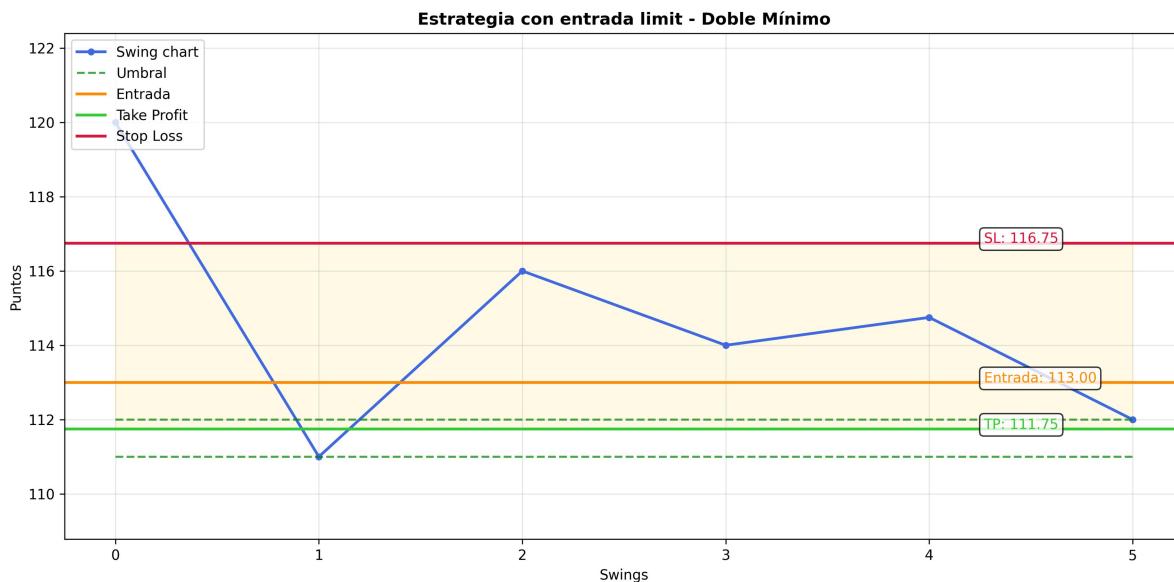
Observamos que estas ya no tienen el timer impuesto, se terminarán ejecutando sí o sí, también la inclusión de “parentId” que será igual al id de la orden de entrada ya que esto hará que estas tres órdenes actúen a modo de conjunto como una “bracket order”, permitiendo un comportamiento tal que al ejecutarse la orden de entrada, automáticamente el sistema por sí mismo coloque ambas y al ejecutarse una de las dos la otra se anule automáticamente. Mención también al parámetro “transmit”, este controla el envío de órdenes al mercado: cuando es “False” (como en la orden de entrada y el take profit), la orden se prepara pero no se envía inmediatamente, permitiendo agruparla con otras, cuando es “True” (como en el stop loss), activa el envío de todas las órdenes pendientes vinculadas al mismo ‘parentId’. Esto asegura que la entrada, el take profit y el stop loss lleguen juntos al mercado, evitando ejecuciones parciales y garantizando que la operación esté siempre protegida. En nuestra estrategia, el stop loss (con ‘transmit=True’) actúa como "disparador" para enviar el paquete completo, coordinando la gestión de riesgo de forma automática.

Si estamos en la situación de doble máximo el take profit será una `LimitOrder` de venta y por lo tanto se ejecutará cuando el precio del activo se encuentre por encima o sea igual que `take_profit_price`. Para el stop loss, que es una `StopOrder`, se ejecutará si el precio del activo sea igual o esté por debajo del `stop_loss_price`. En el caso del doble mínimo el take profit será una `LimitOrder` de compra y por lo tanto se ejecutará cuando el precio del activo se encuentre por debajo o sea igual que `take_profit_price`. Para el stop loss, que es una `StopOrder`, se ejecutará si el precio del activo sea igual o esté por encima del `stop_loss_price`. A continuación unos ejemplos de doble máximo y mínimo:



En el momento que se detecta condición de doble máximo en el swing chart (vemos como la diferencia entre los máximos es menor que el umbral impuesto) colocamos la orden de entrada un tick por debajo del low de la barra más reciente, es decir `low_actual - variación_minima`, si tenemos que `low_actual` es 118,25 y `variación_minima` es 0,25, la orden de entrada estará en 118, luego el take profit un tick por encima del swing más reciente, si este está a 121 tenemos 121,25 y para terminar si tenemos `R_multiplier = 3`; `entrada - (3 * (take profit - entrada)) = 108,25`.

Si tenemos doble mínimo:



El high actual es 112,75 por lo que si la variación mínima es 0,25 entramos con 113, el take profit un tick por debajo del swing actual, 111,75 y stop loss se calcula como:  $113 + (3 * (113 - 111,75)) = 116,75$ .

Existe otra variante en la que la orden de entrada es una `StopOrder`:

```

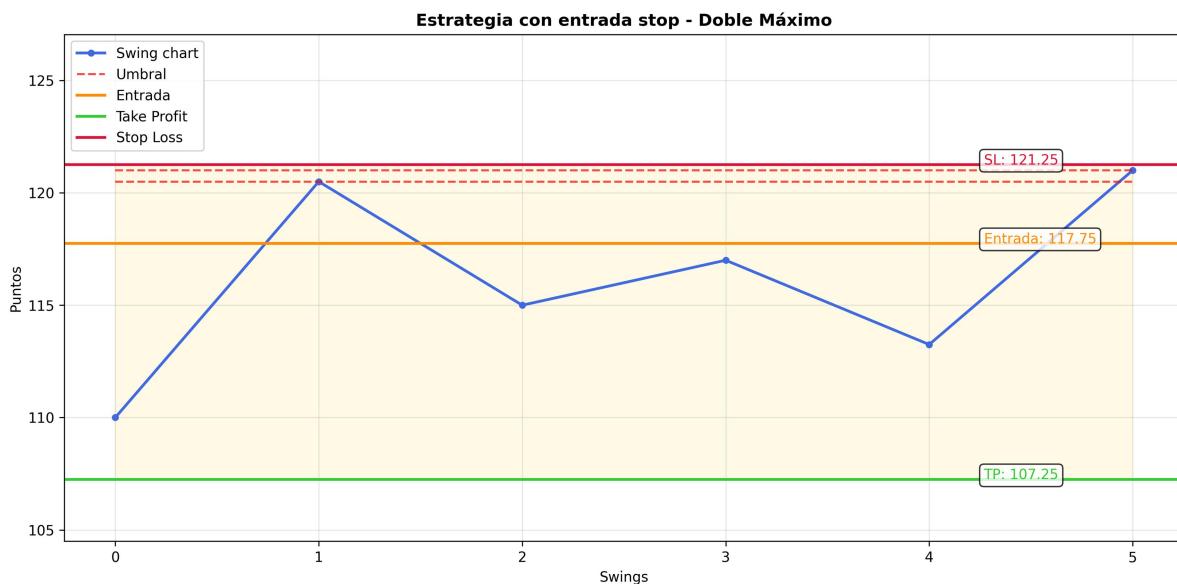
order_timer = (datetime.now() + timedelta(minutes=5)).strftime("%H:%M:%S")

if tipo == 'double_top':
    action = 'SELL'
    quantity = cantidad
    entry_stop_price = round(low_barra_actual - variacion_minima, num_decimales) # ej: calculado a partir del swing
    stop_loss_price = round(swingsy[-1] + variacion_minima, num_decimales)
    take_profit_price = round(entry_stop_price - (R_multiplier * (stop_loss_price - entry_stop_price)), num_decimales)
    # Orden de entrada
    entry_order = StopOrder(
        action=action,
        totalQuantity=quantity,
        lmtPrice=entry_stop_price,
        tif='GTD',
        goodTillDate=order_timer,
        outsideRth=True,
        transmit=False,
        orderId=ib.client.getReqId()
    )

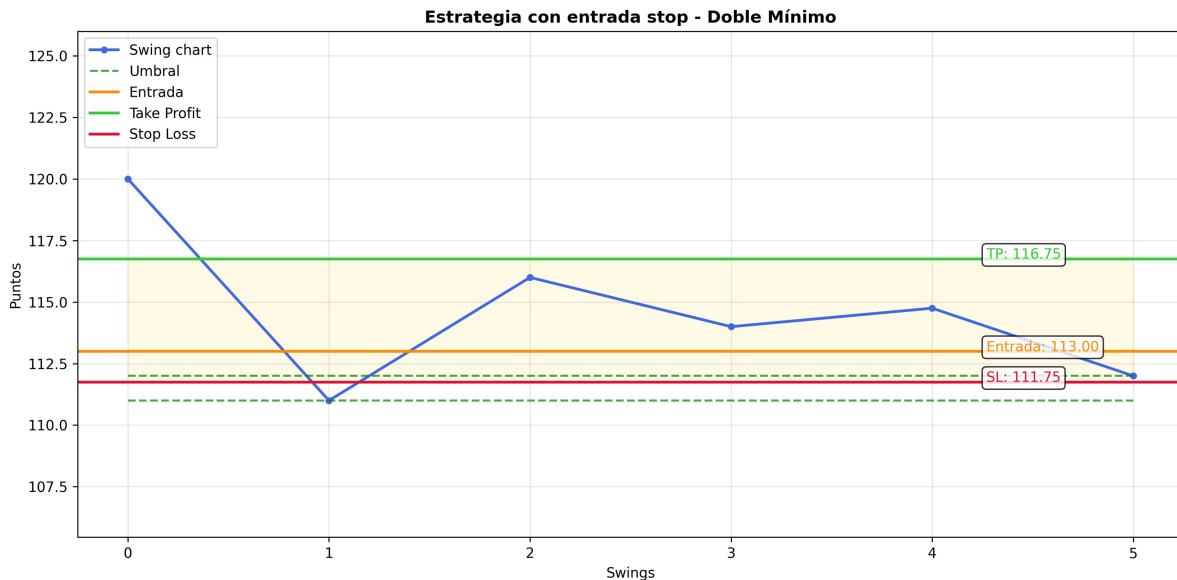
elif tipo == 'double_bottom':
    action = 'BUY'
    quantity = cantidad
    entry_stop_price = round(high_barra_actual + variacion_minima, num_decimales)
    stop_loss_price = round(swingsy[-1] - variacion_minima, num_decimales)
    take_profit_price = round(entry_stop_price + (R_multiplier * (entry_stop_price - stop_loss_price)), num_decimales)
    # Orden de entrada
    entry_order = StopOrder(
        action=action,
        totalQuantity=quantity,
        lmtPrice=entry_stop_price,
        tif='GTD',
        goodTillDate=order_timer,
        outsideRth=True,
        transmit=False,
        orderId=ib.client.getReqId()
    )

```

Si estamos en doble máximo entramos con una stop de venta, se ejecutará cuando el precio sea igual o menor que `entry_stop_price`, cerramos la posición con un `take profit límite de compra` si el precio es igual o menor que `take_profit_price` acabando con ganancias o si este es mayor o igual que `stop_loss_price` saliendo con pérdidas. Si es doble mínimo entramos con un stop de compra que se ejecutará si el precio es igual o mayor que el `entry_stop_price`, continuamos con un `take profit límite de venta` saliendo con ganancias si el precio iguala o supera el `take_profit_price` o con pérdidas si este es igual o inferior a `stop_loss_price`.



Si el low de la barra actual es 118 y la variación mínima 0,25 entramos con 117,75. SL se coloca un tick por encima del swing actual (121) y TP, siendo  $R\_multiplier 3: 117,75 - (3 * (121,25 - 117,75)) = 107,25$ .



En doble mínimo entramos un tick por encima del high de la barra actual (112,75), el SL un tick por debajo del swing actual (112) y TP como  $113 + (3 * (113 - 111,75)) = 116,75$ .

Todos estos códigos pueden verse en su totalidad en los anexos D y E.

Después de definir las órdenes lo siguiente es actualizar unos estados que nos servirán para gestionar todo esto, así como mandar las órdenes definidas anteriormente, recordemos que tenemos esto:

```
variables_por_contrato = defaultdict(lambda: {
    'orden_activa': {
        'enviada': False,
        'entry_order': None,
        'tipo': None,
        'tp_order': None,
        'sl_order': None,
        'take_profit_price': None,
        'stop_loss_price': None,
        'en_espera': False,
    },
    'quantity': 0,
    'umbral': 0.0,
    'variacion_minima': 0.0,
    'R': 0.0,
    'num_decimales': 0,
    'zona_horaria': None,
})
})
```

El parámetro orden activa se lo pasamos a nuestra función e inicializamos nuestros estados:

```
orden_activa.update({
    'enviada': True,
    'entry_order': ib.placeOrder(contract, orden_entrada),
    'tipo': tipo,
    'tp_order': ib.placeOrder(contract, tp),
    'sl_order': ib.placeOrder(contract, sl),
    'take_profit_price': take_profit_price,
    'stop_loss_price': stop_loss_price,
    'en_espera': False
})
```

También mandamos las órdenes al sistema gracias a la instancia de IB que pasamos a la función, aparte de mandarlas guardamos en las variables correspondientes los objetos Trade [1] correspondientes a las órdenes enviadas, es importante este matiz, lo que hicimos antes a la hora de definirlas era un objeto Order, una vez que son mandadas al sistema a través de placeOrder pasan a ser Trades, estos objetos tienen entre sus características el objeto Order previo a ser mandado, esto es importante a la hora de hacer futuras consultas para saber el estado de las órdenes enviadas, muchas de estas consultas no se pueden hacer directamente al objeto Order por lo que es importante tener esto en cuenta.

A partir de aquí, para la próxima vez que entremos a esta función, hay que monitorear las órdenes enviadas, verificamos si hay efectivamente órdenes enviadas, tras esto verificamos posibles estados de nuestras órdenes:

```
# Si ya hay una orden enviada, verificamos si fue ejecutada o hay que cancelarla
if orden_activa['enviada']:
    if orden_activa['tp_order'] and orden_activa['tp_order'].orderStatus.status == 'Filled':
        print("Take Profit ejecutado.")
        orden_activa.update({'enviada': False, 'entry_order': None, 'tipo': None, 'tp_order': None, 'sl_order': None, 'take_profit_price': None, 'stop_loss_price': None, 'en_espera': False})

    elif orden_activa['sl_order'] and orden_activa['sl_order'].orderStatus.status == 'Filled':
        print("Stop Loss ejecutado.")
        orden_activa.update({'enviada': False, 'entry_order': None, 'tipo': None, 'tp_order': None, 'sl_order': None, 'take_profit_price': None, 'stop_loss_price': None, 'en_espera': False})

    elif orden_activa['entry_order'].orderStatus.status == 'Filled':
        print("Entrada ejecutada. Esperando a que se ejecuten TP o SL...")
        return

    elif orden_activa['entry_order'].orderStatus.status in ('Cancelled', 'Inactive'):
        print("Orden cancelada o inactiva. Se puede buscar nueva entrada.")
        orden_activa.update({'enviada': False, 'entry_order': None, 'tipo': None, 'tp_order': None, 'sl_order': None, 'take_profit_price': None, 'stop_loss_price': None, 'en_espera': False})
```

Estas comprobaciones sirven para saber si la orden de entrada se ejecutó, en caso de cerrarse la operación volvemos a resetear estados y volvemos a buscar patrones, si no seguimos esperando hasta que termine de cerrarse o la orden de entrada haya expirado por el timer de 5 minutos. Si la orden de entrada aún no se ejecutó y sigue activa verificamos si se sigue cumpliendo la condición correspondiente de doble top/bottom, de ser así anulamos, reseteamos y vuelta a buscar nuevos

patrones, en caso contrario dejamos que la orden siga colocada y verificamos en cada nueva iteración hasta saber si finalmente se ejecutó o expiró. En cualquier caso reseteamos y volvemos a empezar.

```

else:
    try:
        # Verificación basada en el estado de la orden
        if (orden_activa['entry_order'].isActive() == True):

            if orden_activa['en_espera'] == True:
                print("Orden en espera, no se cancela.")
                return

            if orden_activa['tipo'] == 'double_top':
                if swingsy[-1] > swingsy[-2]: # possible double top
                    for k in range(1, (lsw - 1) // 2 + 1):
                        dif = swingsy[-1] - swingsy[(-2 * k)-1]
                        if abs(dif) < umbral:
                            print("Cancelando orden anterior que no se ejecutó...")
                            ib.cancelOrder(orden_activa['entry_order'].order)
                            orden_activa.update({
                                'enviada': False,
                                'entry_order': None,
                                'tipo': None,
                                'tp_order': None,
                                'sl_order': None,
                                'take_profit_price': None,
                                'stop_loss_price': None,
                                'en_espera': False
                            })
                            break
                elif dif <= -umbral:
                    orden_activa['en_espera'] = True
                    return
                else:
                    orden_activa['en_espera'] = True
                    return
            else:
                orden_activa['en_espera'] = True
                return

```

```
else: # possible double bottom
    if swingsy[-1] < swingsy[-2]: # possible double bottom
        for k in range(1, (lsw - 1) // 2 + 1):
            dif = swingsy[-1] - swingsy[(-2 * k)-1]
            if abs(dif) < umbral:
                print("Cancelando orden anterior que no se ejecutó...")
                ib.cancelOrder(orden_activa['entry_order'].order)
                orden_activa.update({
                    'enviada': False,
                    'entry_order': None,
                    'tipo': None,
                    'tp_order': None,
                    'sl_order': None,
                    'take_profit_price': None,
                    'stop_loss_price': None,
                    'en_espera': False
                })
                break
            elif dif >= umbral:
                orden_activa['en_espera'] = True
                return
        else:
            orden_activa['en_espera'] = True
            return

    else:
        orden_activa['en_espera'] = True
        return

except Exception as e:
    print(f"Error durante el proceso de cancelación: {str(e)}")
    # No es crítico si falla la cancelación
```

## 2.6. CONEXIÓN, RECONEXIÓN Y LA GESTIÓN DEL PROGRAMA

A continuación comentaremos el archivo `main_secuencial.py`, en el que se van a gestionar todas las funcionalidades de los ficheros mencionados anteriormente así como la conexión, desconexión y reconexión con Interactive Brokers. Despues de definir los contratos sigue la función `main()`:

```

def main():
    max_reintentos = 20 # Máximo de intentos de reconexión
    reintentos = 0
    primeraConexion = True
    ib = None

    while reintentos < max_reintentos: # Bucle externo para controlar reintentos
        try:

            except KeyboardInterrupt:
                print("\n🔴 Interrupción por usuario")
                break

            except Exception as e:
                reintentos += 1
                print(f"🔴 {config['symbol']}: Error crítico (intento {reintentos}/{max_reintentos}) - {str(e)}")
                time.sleep(10)

        # --- Limpieza Final ---
        if ib and ib.isConnected():
            ib.disconnect()
        print("💡 Programa terminado")

    if __name__ == '__main__':
        print("""
=====
🚀 BOT MULTICONTRATO (ÚNICA INSTANCIA IB)
=====
""")
        main()

```

Inicializamos una instancia de IB, luego tenemos un bucle para controlar la reconexión, al salir de este bucle gestionamos la salida del programa y la desconexión comprobando si la instancia no está vacía y si existe conexión. Dentro del bucle tenemos un bloque try donde estará toda nuestra lógica de administración de tareas, le siguen dos except, uno para cerrar el programa y otro para capturar eventos de desconexión, esperamos 10 segundos entre cada reinicio de conexión. El número de intentos podemos ponerlo en lo que queramos. Dentro del try tenemos:

```

if ib is None or not ib.isConnected():
    ib = IB()
    ib.connect('127.0.0.1', 7497, clientId=1)
    print("✅ Conexión IB establecida")
    reintentos = 0

if primeraConexion:
    # Prepara todos los contratos
    contratos_ib = [Future(c['symbol'], c['expiry'], c['exchange']) for c in CONTRATOS]
    vars_contratos = {c['symbol']: variables_por_contrato(c['symbol']) for c in CONTRATOS}

    for config, contract in zip(CONTRATOS, contratos_ib):
        symbol = config['symbol']
        umbral, cantidad, variacion_minima, r, num_decimales, zona_horaria = configurar_contrato(config['symbol'])
        vars_contratos[symbol].update({'umbral': umbral, 'cantidad': cantidad, 'variacion_minima': variacion_minima, 'R': r,
                                       'num_decimales': num_decimales, 'zona_horaria': zona_horaria})
        file_path = f'{symbol}.min'

    primeraConexion = False

```

Tenemos un primer if en el que nos conectamos y ponemos reintentos a 0, después un segundo if en el que entraremos solo la primera vez para definir todos los contratos y todas las variables asociadas a ellos, seguido de estos tenemos un else en el que entraremos cada vez que se produzca una reconexión.

```

else:
    for config, contract in zip(Contratos, contratos_ib):
        symbol = config['symbol']
        vars_contrato = vars_contratos[symbol]

        # RECONSTRUCCIÓN POST-RECONEXIÓN

        print("Entrando en reconstrucción post-reconexión") # Debug
        if vars_contrato['orden_activa']['entry_order'] and hasattr(vars_contrato['orden_activa']['entry_order'].order, 'permId'):
            vars_contrato['orden_activa']['entry_order'] = recuperar_trade_por_id(ib, vars_contrato['orden_activa']['entry_order'].order.permId)

        if vars_contrato['orden_activa']['tp_order'] and hasattr(vars_contrato['orden_activa']['tp_order'].order, 'permId'):
            vars_contrato['orden_activa']['tp_order'] = recuperar_trade_por_id(ib, vars_contrato['orden_activa']['tp_order'].order.permId)

        if vars_contrato['orden_activa']['sl_order'] and hasattr(vars_contrato['orden_activa']['sl_order'].order, 'permId'):
            vars_contrato['orden_activa']['sl_order'] = recuperar_trade_por_id(ib, vars_contrato['orden_activa']['sl_order'].order.permId)

```

El objetivo de este es volver a recuperar la referencia de los Trades almacenados en orden\_activa. Cuando se pierde la conexión se pierde también la referencia a este objeto por lo que es imposible volver a saber el estado de estos, por eso hay que recuperarlos, para ello hacemos uso de la función recuperar\_trade\_por\_id(ib, order\_id) que devuelve el objeto buscado, para ello hacemos una petición de todos los trades de la sesión mediante la instancia de IB y luego filtramos hasta encontrar el trade cuya orden tenga el “permId” buscado.

```

def recuperar_trade_por_id(ib, order_id: int) -> Trade:

    for trade in ib.trades():
        if trade.order.permId == order_id:
            return trade

    return None

```

A continuación tenemos el bucle principal el cual se ejecutará continuamente para actualizar los ficheros .min, detectar patrones y colocar órdenes.

```

while True:
    for config, contract in zip(CONTRATOS, contratos_ib):
        # Verificar conexión antes de cada operación
        if not ib.isConnected():
            raise ConnectionError("Conexión perdida con IB")

        try:
            # Procesamiento por contrato
            symbol = config['symbol']
            file_path = f"{symbol}.min"

            # 1. Actualizar datos
            actualizar_archivo_min(ib, file_path, contract, vars_contratos[symbol]['zona_horaria'])

            caldayi, _, hii, loi, cli, opa, voli = leer_min(file_path, vars_contratos[symbol]['zona_horaria'])
            caldayi = np.array(caldayi)
            hii = np.array(hii)
            loi = np.array(loi)
            cli = np.array(cli)
            opa = np.array(opa)
            voli = np.array(voli)
            hi_agrupacion, lo_agrupacion, _, _, _, _ = dividir_min_en_grupos_para_swings(caldayi, hii, loi, cli, opa, voli, 3, 1, len(caldayi)-1)

            # 2. Calcular en swings
            hi, lo, cl, op, dhi, dlo = dividir_min_en_grupos_para_swings(caldayi, hii, loi, cli, opa, voli, 2, 80, len(caldayi)-1)
            swings, tiempos, _ = swing1(hi, lo, cl, op, dhi, dlo)

            # 3. Ejecutar estrategia
            procesar_doble_top_bottom(
                swings, tiempos, vars_contratos[symbol]['umbral'], ib, contract,
                vars_contratos[symbol]['orden_activa'], hi_agrupacion[0], lo_agrupacion[0],
                config['client_id'], vars_contratos[symbol]['quantity'], vars_contratos[symbol]['variacion_minima'],
                vars_contratos[symbol]['R'], vars_contratos[symbol]['num_decimales'], vars_contratos[symbol]['zona_horaria'],
            )
        except ConnectionError as e:
            print(f"⚠ {config['symbol']}: Error de conexión - {str(e)}")
            if ib.isConnected():
                ib.disconnect()
            time.sleep(10)
            break # Sale del bucle interno para reintentar conexión

        except Exception as e:
            print(f"⚠ Error en {symbol}: {str(e)}")

```

Lo primero que nos encontramos es un bucle for para acceder a todos los contratos y todas las variables asociadas a ellos, tras esto comprobamos con cada iteración si hay alguna desconexión de ser así el bloque except correspondiente capturará el evento de desconexión, esperará 10 segundos y se saldrá del bucle for, seguidamente volverá al bucle for y si se sigue desconectado, al estar fuera del bloque try interno esta excepción caerá en el except Exception exterior donde se incrementará reintentos en una unidad y así hasta que se vuelva a conectar, saltando como comentamos antes al bloque else para volver a tener la referencia de los trades y así volver a empezar. Como vemos el flujo normal consiste en actualizar el .min, leer este y sacar las listas necesarias con los días, highs, lows, close..., seguidamente convertirlos de listas a Numpy arrays para así operar en dividir\_min\_en\_grupos\_para\_swing de forma más eficiente en las operaciones con los vectores. Observamos que calculamos el high y low actual como un grupo de los 3 últimos registros y estos serán respectivamente el mayor y menor high y low del grupo, para posteriormente pasárselos a la función de estrategia, este número (swing corto) de los 3 últimos registros será otro parámetro a la hora de estudiar las formas de optimizar las ganancias, en el apartado de resultados comentaremos más sobre esto, por lo que puede cambiar en función de maximizar beneficios. En la imagen en concreto vemos que a la hora de calcular el swing chart tendremos en cuenta los 80 grupos de 2 minutos más recientes del .min, los minutos de cada grupo será otro parámetro a estudiar (swing largo) al igual que el número de grupos.

Esta es la versión secuencial en la que se tiene una única instancia de IB y vamos trabajando con cada contrato uno tras otro, también se implementó una versión multiprocesamiento (**main\_multiprocessing.py**) donde tendremos una instancia IB por cada contrato y así trabajar de forma paralela con cada uno de ellos, en esta versión se dará uso de la característica

clientId de cada contrato para establecer las respectivas conexiones, existe un máximo de 32 conexiones API simultáneas por cuenta por lo que en nuestro caso al ser 21 contratos no habrá problema, existe el inconveniente de que se consume más RAM que un hilo, pero de tener disponible una mayor capacidad esta versión es más potente. Este script está diseñado para automatizar el proceso de trading en múltiples contratos de futuros usando la API de Interactive Brokers a través de la biblioteca `ib_insync` de la misma manera que en el secuencial. Su propósito es gestionar en paralelo varios contratos, cada uno con su propia configuración específica, realizando tareas como conexión al broker, recolección de datos, análisis de patrones y ejecución de estrategias de *trading* en tiempo real. Esto permite operar de manera simultánea distintos instrumentos financieros de forma eficiente y escalable.

En primer lugar, importamos varias librerías esenciales: `time` para gestionar pausas y tiempos de espera, `multiprocessing.Process` para ejecutar varios procesos en paralelo (cada uno controlando un contrato independiente), `ib_insync` para interactuar con la plataforma de Interactive Brokers, y módulos propios como `variables_por_contrato`, `procesado` y `estrategia_limit`, que contienen funciones y lógicas específicas para el manejo de los datos y estrategias de trading.

El script define una lista llamada `CONTRATOS`, donde se especifican los contratos de futuros que se desean operar. Cada elemento de la lista es un diccionario con detalles como el símbolo del contrato, fecha de expiración, bolsa de negociación y un `client_id` único para identificar la conexión al broker. Esto permite que cada contrato se procese en su propio cliente de TWS o IB Gateway, evitando conflictos en las conexiones.

Luego, hay una función llamada `configurar_contrato` que devuelve los parámetros específicos de configuración para un símbolo dado. Estos parámetros incluyen valores como el umbral de movimiento del precio que activa la estrategia, el margen necesario, la cantidad de contratos por orden, la variación mínima de precio a considerar, el múltiplo de riesgo/beneficio ( $R$ ), el número de decimales para redondeos, y la zona horaria para manejar correctamente los datos temporales. Esto permite que cada contrato se adapte a sus propias características del mercado.

Tras esto tenemos la función principal llamada `ejecutar_contrato(config)` que se encarga de gestionar la conexión con Interactive Brokers (IBKR) y ejecutar la lógica de trading para un contrato determinado. La función recibe un diccionario `config` que contiene información específica de cada contrato. Dentro de la función, se establece un bucle externo que controla los intentos de reconexión con un máximo definido por `max_reintentos`. Esto asegura que, si el bot pierde la conexión o se encuentra con un error crítico, intente reconectarse antes de abandonar el proceso.

Al inicio de cada intento, el bot verifica si necesita conectarse. Si no hay una conexión activa, crea una instancia de `IB()`, se conecta al servidor local de IBKR (en el puerto 7497, que es el estándar para TWS en modo paper trading), y reinicia el contador de reintentos. Luego, el bot configura el contrato (en este caso un futuro) utilizando los parámetros de `config`. También prepara el nombre del archivo en el que se guardarán los datos minuto a minuto del contrato (`file_path`).

La primera vez que el bot logra conectarse, inicializa todas las variables específicas de ese contrato (como el umbral de activación, cantidad a operar, variación mínima, tamaño de riesgo  $R$ , cantidad de decimales y zona horaria). Esto se hace a través de llamadas a funciones como `configurar_contrato()` y se almacenan en un diccionario `vars_contrato`. En caso de reconexión (cuando no es la primera vez), el bot intenta reconstruir el estado de las órdenes activas recuperando los Trade mediante sus IDs permanentes (`permId`) con la función `recuperar_trade_por_id`. Así, si el bot se reinicia tras una desconexión, intenta no perder el rastro de las órdenes enviadas anteriormente.

Dentro del bucle interno de trading, el bot realiza el ciclo operativo: primero verifica que la conexión siga activa y, si no lo está, lanza un error para activar la lógica de reconexión. Luego, actualiza el archivo de datos minuto a minuto usando `actualizar_archivo_min()`, y carga estos datos en arrays de Numpy para trabajar con ellos. A partir de los datos, forma agrupaciones para análisis de swings. Esta información se pasa a la función `swing1()` que genera el swing chart con sus tiempos correspondientes.

Con estos swings calculados, el bot llama a `procesar_doble_top_bottom()` que implementa la estrategia: busca patrones de doble máximo o doble mínimo y gestiona las órdenes activas: entry, take profit (tp) y stop loss (sl). El ciclo de trading espera 50 segundos entre cada iteración (con `ib.sleep(50)`), de modo que se ajusta bien al flujo de datos de un minuto. Si ocurre un error de conexión durante este bucle, el bot limpia la conexión (desconecta si aún está conectada), espera unos segundos y rompe el bucle interno para intentar reconectar.

En la parte exterior de la función, el bot está preparado para manejar errores críticos (como fallas al conectar) con un sistema de reintentos: si ocurre un error, incrementa el contador y espera antes de volver a intentar. Si se supera el número máximo de reintentos, el bot desconecta y termina. Además, el bot permite interrumpir su ejecución de forma segura con `KeyboardInterrupt` (por ejemplo, presionando `Ctrl+C`), para que el usuario pueda detener el proceso cuando lo desee.

Finalmente, en el bloque `if __name__ == '__main__' :`, el bot lanza un proceso independiente para cada contrato especificado en la lista `CONTRATOS`. Esto se hace mediante la creación de procesos con `multiprocessing.Process`. El programa principal espera que todos los procesos terminen usando `join()`, y si recibe una señal de interrupción, finaliza todos los procesos de forma limpia con `terminate()`. Al final, imprime un mensaje de despedida confirmando que todos los procesos se han cerrado.

En la creación de procesos, se inicializa una lista `procesos` y, para cada configuración en `CONTRATOS`, se crea un nuevo proceso (no un hilo) con `Process(target=ejecutar_contrato, args=(config,))`, donde `target` especifica la función a ejecutar (`ejecutar_contrato`) y `args` pasa la configuración del contrato como argumento; luego, con `p.start()` se inicia el proceso y con `procesos.append(p)` se almacena en la lista para su gestión posterior. En el manejo de interrupciones, se usa un bloque `try` para esperar a que todos los procesos terminen (`p.join()`), capturando `KeyboardInterrupt` (con `Ctrl+C`) para detenerlos de forma controlada (`p.terminate()`), mientras que el bloque `finally` asegura que siempre se ejecute el mensaje final ("Todos los procesos terminados").

```
import time
from multiprocessing import Process
from ib_insync import *
import variables_por_contrato
from procesado import *
from estrategia_limit import procesar_doble_top_bottom

CONTRATOS = [
    {'symbol': 'ES', 'expiry': '202506', 'exchange': 'CME', 'client_id': 1},
    {'symbol': 'NQ', 'expiry': '202506', 'exchange': 'CME', 'client_id': 2},
    {'symbol': 'CL', 'expiry': '202507', 'exchange': 'NYMEX', 'client_id': 3},
    {'symbol': 'GC', 'expiry': '202506', 'exchange': 'COMEX', 'client_id': 4},
    {'symbol': 'IBEX35', 'expiry': '202506', 'exchange': 'MEFFRV', 'client_id': 5},
    {'symbol': 'HG', 'expiry': '202506', 'exchange': 'COMEX', 'client_id': 6},
]
```

```

        {'symbol': 'NG', 'expiry': '202506', 'exchange': 'NYMEX', 'client_id': 7},
        {'symbol': 'HO', 'expiry': '202506', 'exchange': 'NYMEX', 'client_id': 8},
        {'symbol': 'YM', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 9},
        {'symbol': 'RTY', 'expiry': '202506', 'exchange': 'CME', 'client_id': 10},
        {'symbol': 'JPY', 'expiry': '202506', 'exchange': 'CME', 'client_id': 11},
        {'symbol': 'EUR', 'expiry': '202506', 'exchange': 'CME', 'client_id': 12},
        {'symbol': 'GBP', 'expiry': '202506', 'exchange': 'CME', 'client_id': 13},
        {'symbol': 'AUD', 'expiry': '202506', 'exchange': 'CME', 'client_id': 14},
        {'symbol': 'CHF', 'expiry': '202506', 'exchange': 'CME', 'client_id': 15},
        {'symbol': 'PA', 'expiry': '202506', 'exchange': 'NYMEX', 'client_id': 16},
        {'symbol': 'PL', 'expiry': '202507', 'exchange': 'NYMEX', 'client_id': 17},
        {'symbol': 'ZS', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 18},
        {'symbol': 'ZM', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 19},
        {'symbol': 'ZC', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 20},
        {'symbol': 'ZW', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 21},
    ]
}

def configurar_contrato(symbol):
    # Diccionario completo con configuración por contrato
    config = {
        'ES': {'umbral': 2.5, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.25, 'R': 3,
        'num_decimales': 2, 'zona_horaria': 'America/Chicago'},
        'NQ': {'umbral': 10.0, 'margen': 40000, 'contratos_orden': 1, 'variacion_minima': 0.25, 'R': 3,
        'num_decimales': 2, 'zona_horaria': 'America/Chicago'},
        'YM': {'umbral': 20.0, 'margen': 13000, 'contratos_orden': 3, 'variacion_minima': 1.0, 'R': 3,
        'num_decimales': None, 'zona_horaria': 'America/Chicago'},
        'RTY': {'umbral': 1.0, 'margen': 10000, 'contratos_orden': 4, 'variacion_minima': 0.1, 'R': 3,
        'num_decimales': 1, 'zona_horaria': 'America/Chicago'},
        'JPY': {'umbral': 1.0, 'margen': 4000, 'contratos_orden': 10, 'variacion_minima': 0.000001,
        'R': 3, 'num_decimales': 6, 'zona_horaria': 'America/Chicago'},
        'EUR': {'umbral': 0.0002, 'margen': 5000, 'contratos_orden': 8, 'variacion_minima': 0.00005,
        'R': 3, 'num_decimales': 5, 'zona_horaria': 'America/Chicago'},
        'GBP': {'umbral': 0.0002, 'margen': 2500, 'contratos_orden': 16, 'variacion_minima': 0.0001,
        'R': 3, 'num_decimales': 4, 'zona_horaria': 'America/Chicago'},
        'AUD': {'umbral': 0.0002, 'margen': 2500, 'contratos_orden': 16, 'variacion_minima': 0.00005,
        'R': 3, 'num_decimales': 5, 'zona_horaria': 'America/Chicago'},
        'CHF': {'umbral': 0.0002, 'margen': 5000, 'contratos_orden': 8, 'variacion_minima': 0.00005,
        'R': 3, 'num_decimales': 5, 'zona_horaria': 'America/Chicago'},
        'CL': {'umbral': 0.03, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.01, 'R': 3,
        'num_decimales': 2, 'zona_horaria': 'America/New_York'},
        'NG': {'umbral': 0.002, 'margen': 11000, 'contratos_orden': 3, 'variacion_minima': 0.001, 'R': 3,
        'num_decimales': 3, 'zona_horaria': 'America/New_York'},
        'HO': {'umbral': 0.0002, 'margen': 26000, 'contratos_orden': 1, 'variacion_minima': 0.0001,
        'R': 3, 'num_decimales': 4, 'zona_horaria': 'America/New_York'},
        'GC': {'umbral': 15.0, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.1, 'R': 3,
        'num_decimales': 1, 'zona_horaria': 'America/New_York'},
    }
}

```

```

        'PA': {'umbral': 5.0, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.5, 'R': 3,
'num_decimales': 1, 'zona_horaria': 'America/New_York'},
        'PL': {'umbral': 5.0, 'margen': 6000, 'contratos_orden': 6, 'variacion_minima': 0.1, 'R': 3,
'num_decimales': 1, 'zona_horaria': 'America/New_York'},
        'HG': {'umbral': 0.02, 'margen': 13000, 'contratos_orden': 3, 'variacion_minima': 0.0005, 'R':
3, 'num_decimales': 4, 'zona_horaria': 'America/New_York'},
        'ZS': {'umbral': 5.0, 'margen': 3500, 'contratos_orden': 10, 'variacion_minima': 0.25, 'R': 3,
'num_decimales': 2, 'zona_horaria': 'America/Chicago'},
        'ZM': {'umbral': 1.0, 'margen': 3200, 'contratos_orden': 11, 'variacion_minima': 0.1, 'R': 3,
'num_decimales': 1, 'zona_horaria': 'America/Chicago'},
        'ZC': {'umbral': 2.0, 'margen': 2200, 'contratos_orden': 18, 'variacion_minima': 0.25, 'R': 3,
'num_decimales': 2, 'zona_horaria': 'America/Chicago'},
        'ZW': {'umbral': 2.0, 'margen': 3300, 'contratos_orden': 11, 'variacion_minima': 0.25, 'R': 3,
'num_decimales': 2, 'zona_horaria': 'America/Chicago'},
        'IBEX35': {'umbral': 30.0, 'margen': 13000, 'contratos_orden': 3, 'variacion_minima': 1.0,
'R': 3, 'num_decimales': None, 'zona_horaria': 'Europe/Madrid'},
        #'SI': {'umbral': 0.5, 'margen': 25000, 'contratos_orden': 1, 'variacion_minima': 0.005} #

Agregado Silver
}

# Obtenemos la configuración o valores por defecto si el símbolo no existe
contrato = config.get(symbol, {
    'umbral': 1.0,
    'margen': 10000,
    'contratos_orden': 1,
    'variacion_minima': 0.01, # Valor por defecto para variación mínima
    'R': 3,
    'num_decimales': 2, # Valor por defecto para número de decimales
    'zona_horaria': 'America/New_York' # Valor por defecto para zona horaria
})

# Devolvemos todos los parámetros necesarios
return (
    contrato['umbral'],
    contrato['contratos_orden'],
    contrato['variacion_minima'],
    contrato['R'],
    contrato['num_decimales'],
    contrato['zona_horaria']
)

def recuperar_trade_por_id(ib, order_id: int) -> Trade:

    for trade in ib.trades():
        if trade.order.permId == order_id:
            return trade

    return None

```

```

def ejecutar_contrato(config):
    max_reintentos = 20 # Máximo de intentos de reconexión
    reintentos = 0
    primeraConexion = True
    ib = None

    while reintentos < max_reintentos: # Bucle externo para controlar reintentos
        try:
            # --- Conexión Inicial ---

            if ib is None or not ib.isConnected():
                print(f"➔ {config['symbol']}: Conectando a IB...")
                ib = IB()
                ib.connect('127.0.0.1', 7497, clientId=config['client_id'])
                print(f"☒{config['symbol']}: Conectado")
                reintentos = 0

            # --- Configuración del contrato (siempre se ejecuta) ---
            contract = Future(config['symbol'], config['expiry'], config['exchange'])
            file_path = f"{config['symbol']}.min"

            # --- Gestión del estado ---
            if primeraConexion:
                # Configuración inicial para primera conexión
                vars_contrato = variables_por_contrato.obtener_variables(config['symbol'])
                umbral, cantidad, variacion_minima, r, num_decimales, zona_horaria =
                    configurar_contrato(config['symbol'])
                vars_contrato.update({
                    'umbral': umbral,
                    'cantidad': cantidad,
                    'variacion_minima': variacion_minima,
                    'R': r,
                    'num_decimales': num_decimales,
                    'zona_horaria': zona_horaria
                })
                primeraConexion = False
            else:
                # RECONSTRUCCIÓN POST-RECONEXIÓN
                print("Entrando en reconstrucción post-reconexión") # Debug
                if vars_contrato['orden_activa']['entry_order'] and
hasattr(vars_contrato['orden_activa']['entry_order'].order, 'permId'):
                    vars_contrato['orden_activa']['entry_order'] = recuperar_trade_por_id(ib,
vars_contrato['orden_activa']['entry_order'].order.permId)

```

```

        if vars_contrato['orden_activa']['tp_order'] and
hasattr(vars_contrato['orden_activa']['tp_order'].order, 'permId'):
            vars_contrato['orden_activa']['tp_order'] = recuperar_trade_por_id(ib,
vars_contrato['orden_activa']['tp_order'].order.permId)

        if vars_contrato['orden_activa']['sl_order'] and
hasattr(vars_contrato['orden_activa']['sl_order'].order, 'permId'):
            vars_contrato['orden_activa']['sl_order'] = recuperar_trade_por_id(ib,
vars_contrato['orden_activa']['sl_order'].order.permId)

    # --- Bucle principal de trading ---
    while True:
        try:
            # Verificar conexión antes de cada operación
            if not ib.isConnected():
                raise ConnectionError("Conexión perdida con IB")

            # 1. Actualizar datos
            actualizar_archivo_min(ib, file_path, contract, vars_contrato['zona_horaria'])

            caldayi, _, hii, loi, cli, opi, voli = leer_min(file_path,
vars_contrato['zona_horaria'])
            caldayi = np.array(caldai)
            hii = np.array(hii)
            loi = np.array(loi)
            cli = np.array(cli)
            opi = np.array(opi)
            voli = np.array(voli)
            hi_agrupacion, lo_agrupacion, _, _, _, _ =
dividir_min_en_grupos_para_swings(caldai, hii, loi, cli, opi, voli, 3, 1, len(caldai)-1)

            # 2. Calcular en swings
            hi, lo, cl, op, dhi, dlo = dividir_min_en_grupos_para_swings(caldai, hii, loi,
cli, opi, voli, 2, 80, len(caldai)-1)
            swings, tiempos, _ = swing1(hi, lo, cl, op, dhi, dlo)

            # 3. Ejecutar estrategia
            procesar_doble_top_bottom(
                swings, tiempos, vars_contrato['umbral'], ib, contract,
                vars_contrato["orden_activa"], hi_agrupacion[0], lo_agrupacion[0],
                config['client_id'], vars_contrato['quantity'],
                vars_contrato['variacion_minima'], vars_contrato['R'], vars_contrato['num_decimales'],
                vars_contrato['zona_horaria'],
            )

            ib.sleep(50)

```

```

        except ConnectionError as e:
            print(f"⚠ {config['symbol']}]: Error de conexión - {str(e)}")
            if ib.isConnected():
                ib.disconnect()
            time.sleep(10)
            break # Sale del bucle interno para reintentar conexión

        except Exception as e:
            print(f"⚠ {config['symbol']}]: Error inesperado - {str(e)}")
            time.sleep(10)

    except KeyboardInterrupt:
        break # Permite salir con Ctrl+C

    except Exception as e:
        reintentos += 1
        print(f"⚠ {config['symbol']}]: Error crítico (intento {reintentos}/{max_reintentos}) - {str(e)}")
        time.sleep(10)

# --- Limpieza final ---
if ib and ib.isConnected():
    ib.disconnect()
print(f"⚠ {config['symbol']}]: Desconectado")

if __name__ == '__main__':
    print("""
=====
    Iniciando Bot Multi-Contrato
=====
""")
    procesos = []
    for config in CONTRATOS:
        p = Process(target=ejecutar_contrato, args=(config,))
        p.start()
        procesos.append(p)

    try:
        for p in procesos:
            p.join() # Espera a que todos los procesos terminen
    except KeyboardInterrupt:
        print("\n⚠ Recibida señal de interrupción. Deteniendo procesos...")
        for p in procesos:
            p.terminate() # Detención limpia
    finally:
        print("⚠ Todos los procesos terminados")

```

También tenemos versiones que trabajan con la plataforma Visual Chart, en vez de actualizar los .min la plataforma ya lo hace por nosotros y lo único que tenemos que hacer es leer, por lo que es una opción también interesante ya que nos ahorraremos la parte de la pedida de datos a IB que puede dar lugar errores. Lo único que hicimos fue añadir un atributo con la ruta específica de cada archivo y luego enlazar esto con la variable local ‘file\_path’.

```
config = {
    'ES': {'umbral': 2.5, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'ES.min'},
    'NQ': {'umbral': 10.0, 'margen': 40000, 'contratos_orden': 1, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'NQ.min'},
    'YM': {'umbral': 20.0, 'margen': 13000, 'contratos_orden': 3, 'variacion_minima': 1.0, 'R': 3, 'num_decimales': None, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'YM.min'},
    'RTY': {'umbral': 1.0, 'margen': 10000, 'contratos_orden': 4, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'RTY.min'},
    'JPY': {'umbral': 1.0, 'margen': 4000, 'contratos_orden': 10, 'variacion_minima': 0.000001, 'R': 3, 'num_decimales': 6, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'JPY.min'},
    'EUR': {'umbral': 0.0002, 'margen': 5000, 'contratos_orden': 8, 'variacion_minima': 0.00005, 'R': 3, 'num_decimales': 5, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'EUR.min'},
    'GBP': {'umbral': 0.0002, 'margen': 2500, 'contratos_orden': 16, 'variacion_minima': 0.0001, 'R': 3, 'num_decimales': 4, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'GBP.min'},
    'AUD': {'umbral': 0.0002, 'margen': 2500, 'contratos_orden': 16, 'variacion_minima': 0.00005, 'R': 3, 'num_decimales': 5, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'AUD.min'},
    'CHF': {'umbral': 0.0002, 'margen': 5000, 'contratos_orden': 8, 'variacion_minima': 0.00005, 'R': 3, 'num_decimales': 5, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'CHF.min'},
    'CL': {'umbral': 0.03, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.01, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/New_York', 'ruta_archivo': 'CL.min'},
    'NG': {'umbral': 0.002, 'margen': 11000, 'contratos_orden': 3, 'variacion_minima': 0.001, 'R': 3, 'num_decimales': 3, 'zona_horaria': 'America/New_York', 'ruta_archivo': 'NG.min'},
    'HO': {'umbral': 0.0002, 'margen': 26000, 'contratos_orden': 1, 'variacion_minima': 0.0001, 'R': 3, 'num_decimales': 4, 'zona_horaria': 'America/New_York', 'ruta_archivo': 'HO.min'},
    'GC': {'umbral': 15.0, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/New_York', 'ruta_archivo': 'GC.min'},
    'PA': {'umbral': 5.0, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.5, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/New_York', 'ruta_archivo': 'PA.min'},
    'PL': {'umbral': 5.0, 'margen': 6000, 'contratos_orden': 6, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/New_York', 'ruta_archivo': 'PL.min'},
    'HG': {'umbral': 0.02, 'margen': 13000, 'contratos_orden': 3, 'variacion_minima': 0.0005, 'R': 3, 'num_decimales': 4, 'zona_horaria': 'America/New_York', 'ruta_archivo': 'HG.min'},
    'ZS': {'umbral': 5.0, 'margen': 3500, 'contratos_orden': 10, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'ZS.min'},
    'ZM': {'umbral': 1.0, 'margen': 3200, 'contratos_orden': 11, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'ZM.min'},
    'ZC': {'umbral': 2.0, 'margen': 2200, 'contratos_orden': 18, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'ZC.min'},
    'ZW': {'umbral': 2.0, 'margen': 3300, 'contratos_orden': 11, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'ZW.min'},
    'IBEX35': {'umbral': 30.0, 'margen': 13000, 'contratos_orden': 3, 'variacion_minima': 1.0, 'R': 3, 'num_decimales': None, 'zona_horaria': 'Europe/Madrid', 'ruta_archivo': 'IBEX35.min'}
}
```

También actualizamos **variables\_por\_contrato.py** con este nuevo atributo, simplemente hay que ajustarlo con la ruta específica en cada ordenador.

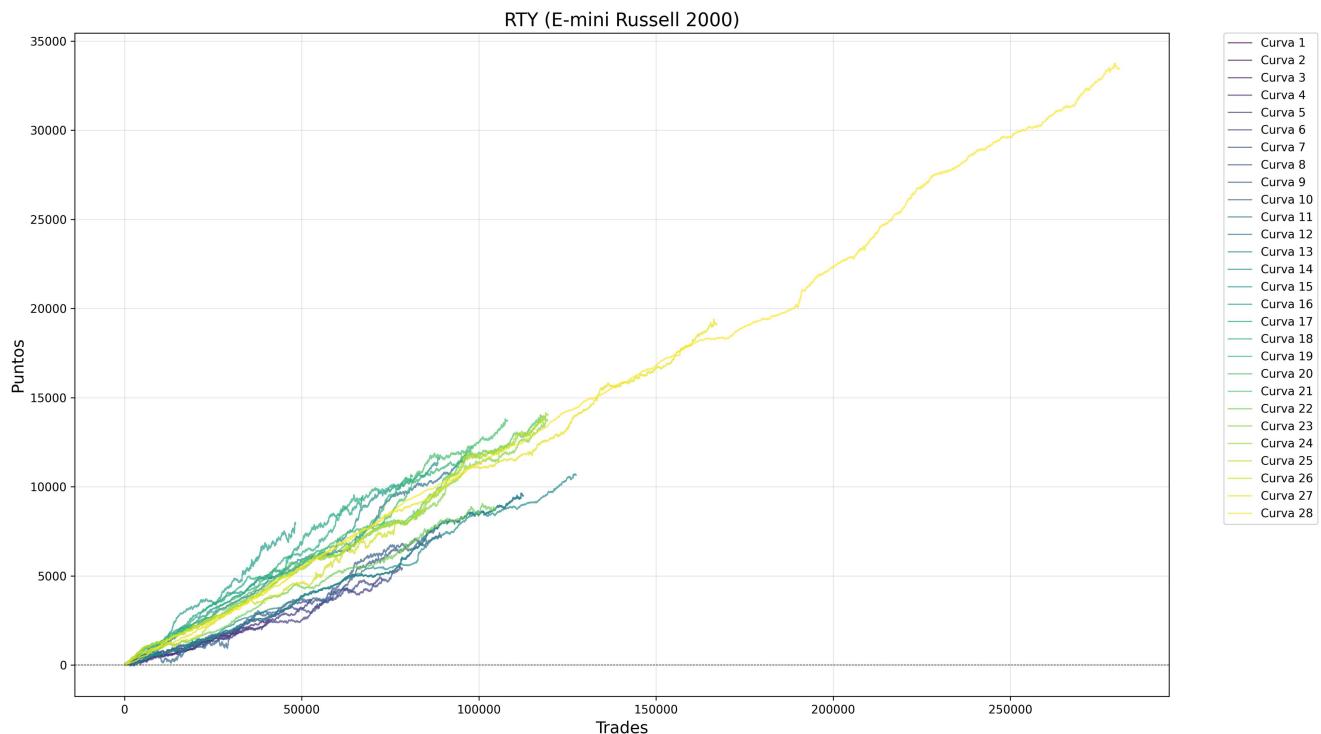
### 3. RESULTADOS

A continuación expondremos los resultados obtenidos para los futuros RTY (E-mini Russell 2.000), CL (Crude Oil Futures), NQ (E-mini Nasdaq-100 futures), ES (E-mini S&P 500) para ello analizaremos cómo nos hubiese ido en caso de haber implementado nuestra estrategia de *trading* sobre los datos históricos correspondientes a cada futuro. Todo esto a través del script **obtener\_curvas.py**, el cual puede verse en el anexo K y en el que se lleva a cabo la misma estrategia comentada hasta ahora para el trading en tiempo real. Como resultado obtendremos una curva de ganancias con el acumulado total. Los parámetros que tendremos en cuenta serán el umbral, la variación mínima, R, swing largo y swing corto. La idea es ir variándolos para encontrar la mejor configuración. Para cada uno de los contratos se requiere un saldo disponible con el que operar, en nuestro caso tenemos colocados unos 40.000 USD aproximadamente por futuro, en concreto para RTY, CL, NQ, ES tenemos 10.000 USD/contrato, 20.000 USD/contrato, 40.000 USD/contrato, 20.000 USD/contrato respectivamente, por lo que en cada orden estaremos manejando 4, 2, 1 y 2 contratos para cada futuro. Si cerramos una posición con ganancias, estas se añadirán al saldo disponible y si son pérdidas se nos sustraerá de este. Siempre que compramos/vendemos hay que cerrar la posición con la correspondiente venta/compra, esto constituirá un *trade*. El CL cotiza directamente en dólares por barril de petróleo, cada contrato representa 1.000 barriles, en el caso del RTY cada contrato cotiza en puntos del índice Russell, NQ cotiza en puntos del índice Nasdaq-100, ES en puntos del índice S&P 500 y cada uno de estos puntos equivale a 50 USD, menos en el caso de NQ que son 20 USD. Con esta información estamos en disposición de presentar los resultados. Añadir que existen comisiones que se calculan por contrato individual negociado. Por tanto, el coste de una operación depende del número de contratos involucrados en cada orden de compra o venta.

Las comisiones aplicadas por Interactive Brokers en la negociación de contratos de futuros responden a un modelo competitivo que combina su propia tarifa de intermediación con los costes asociados a los mercados (*exchange fees*) y las tasas regulatorias. Estas comisiones reflejan el acceso directo que el bróker proporciona a los mercados, así como el coste por garantizar una ejecución eficiente y segura de las órdenes. Además, el importe final por contrato varía en función del tipo de activo y del

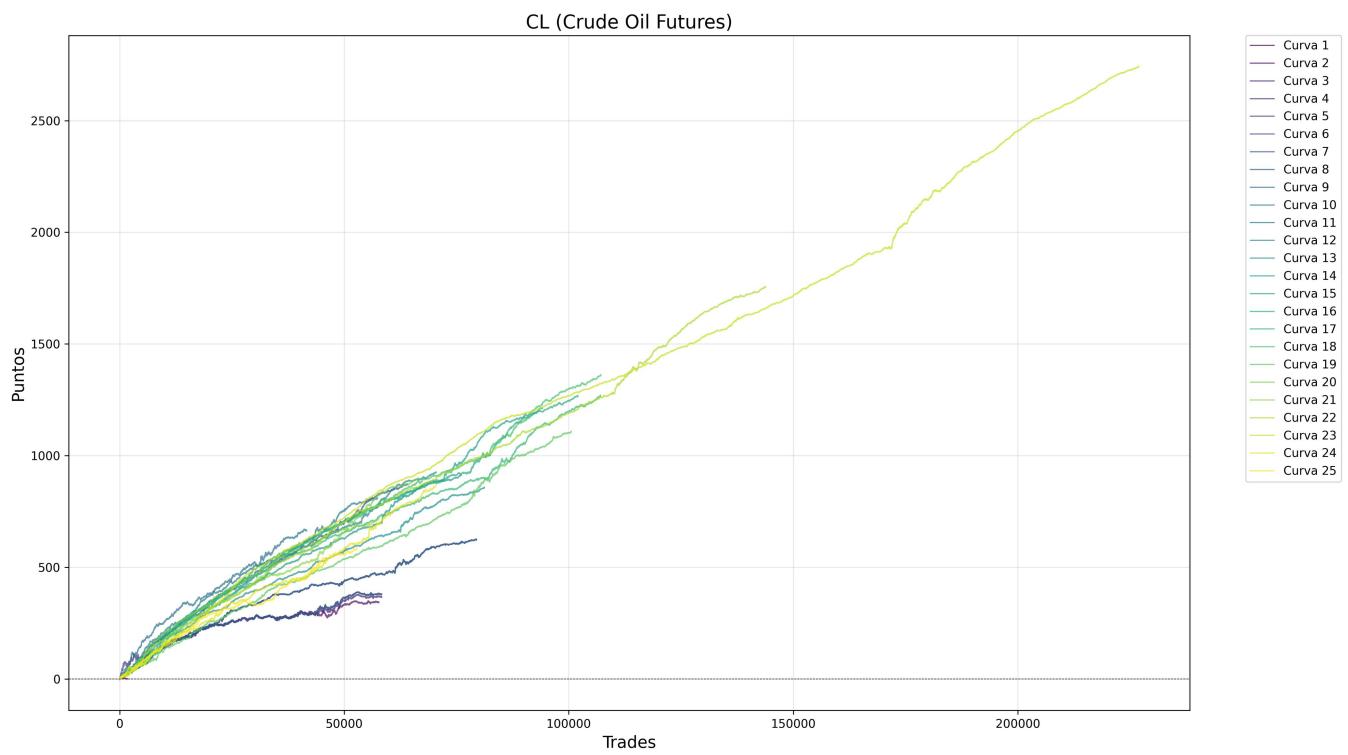
exchange donde se negocia. En el caso de RTY, NQ, ES y CL, de 2,25 y 2,37 USD, por lo que tendremos una comisión total por cada *trade* de 4,5 y 4,74 USD.

RTY:



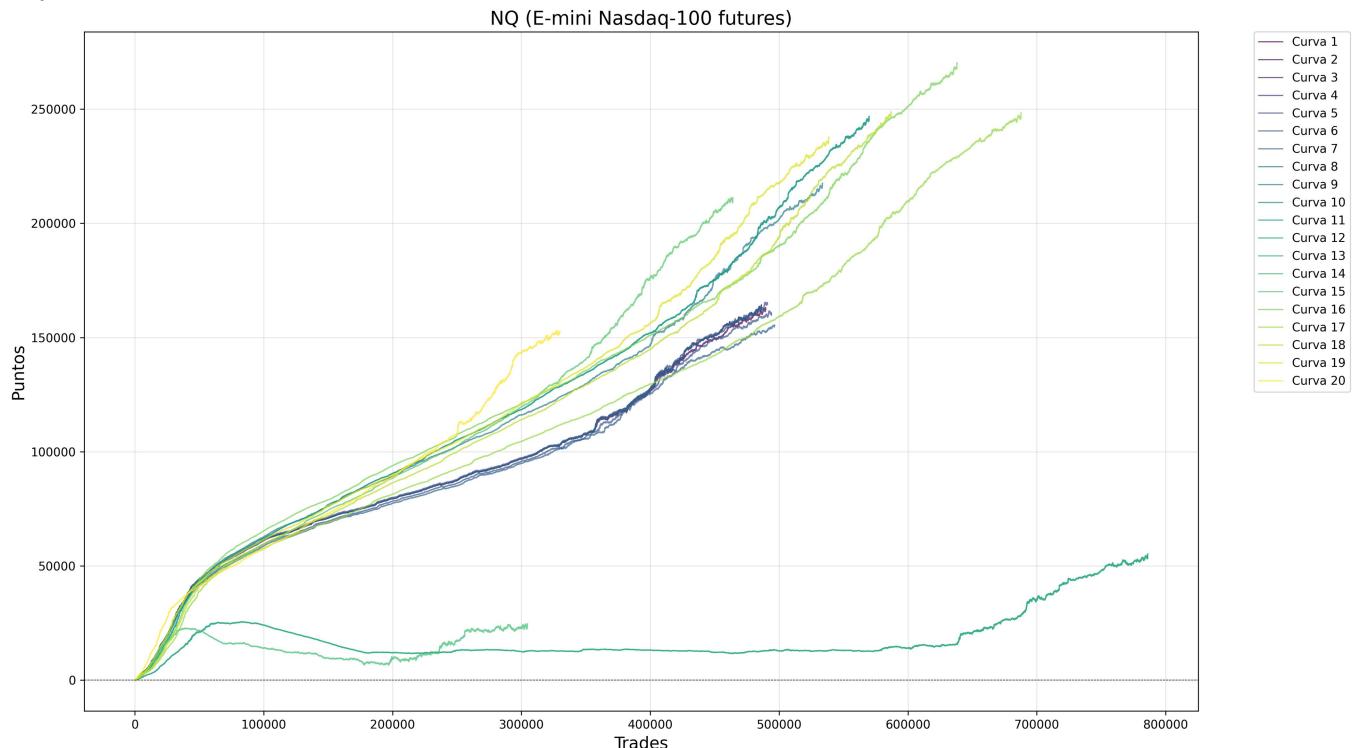
Cada una de estas curvas se obtuvo variando los parámetros mencionados, primero variábamos uno hasta encontrar un buen resultado y entonces saltábamos al siguiente, así con todos. La curva 28 es la que reporta más ganancias, los parámetros son: umbral = 200, variación mínima = 0,05, R = 3,5, swing largo = 3, swing corto = 1. El acumulado total de puntos es 33.445,1 y un total de 280.783 *trades*, por lo que las ganancias por contrato serían de  $(33.445,1 * 50 \text{ USD}) - (280.783 * 2,25 \text{ USD} * 2)$  = 408.731,5 USD y teniendo en cuenta que operamos con 4 contratos las ganancias totales hubiesen sido de **1.634.926 USD**. El .min almacena 5.677.328 minutos, unos 10.8 años, todos estos minutos correspondientes a horas donde el mercado está abierto y disponible para operar, esto no incluye festivos y fines de semana. Por lo que las ganancias comprenden ese periodo. Siendo las ganancias medias por operación de  $1.634.926 \text{ USD} / 280.783 \text{ trades} = 5,82 \text{ USD/trade}$

Lo mismo para CL:



Umbral= 20, variación mínima = 0,005, R = 2,5, swing largo = 2, swing corto = 1.  $(2.743,28 \text{ USD} * 1000) - (226.941 * 2,37 \text{ USD} * 2) = 1.667.579,66 \text{ USD}$ , siendo las ganancias totales de **3.335.159,32 USD** habiendo operado sobre 4.911.465 minutos, es decir unos 9,34 años. Siendo las ganancias medias por operación de  $3.335.159,32 \text{ USD} / 226.941 \text{ trades} = 14,70 \text{ USD/trade}$

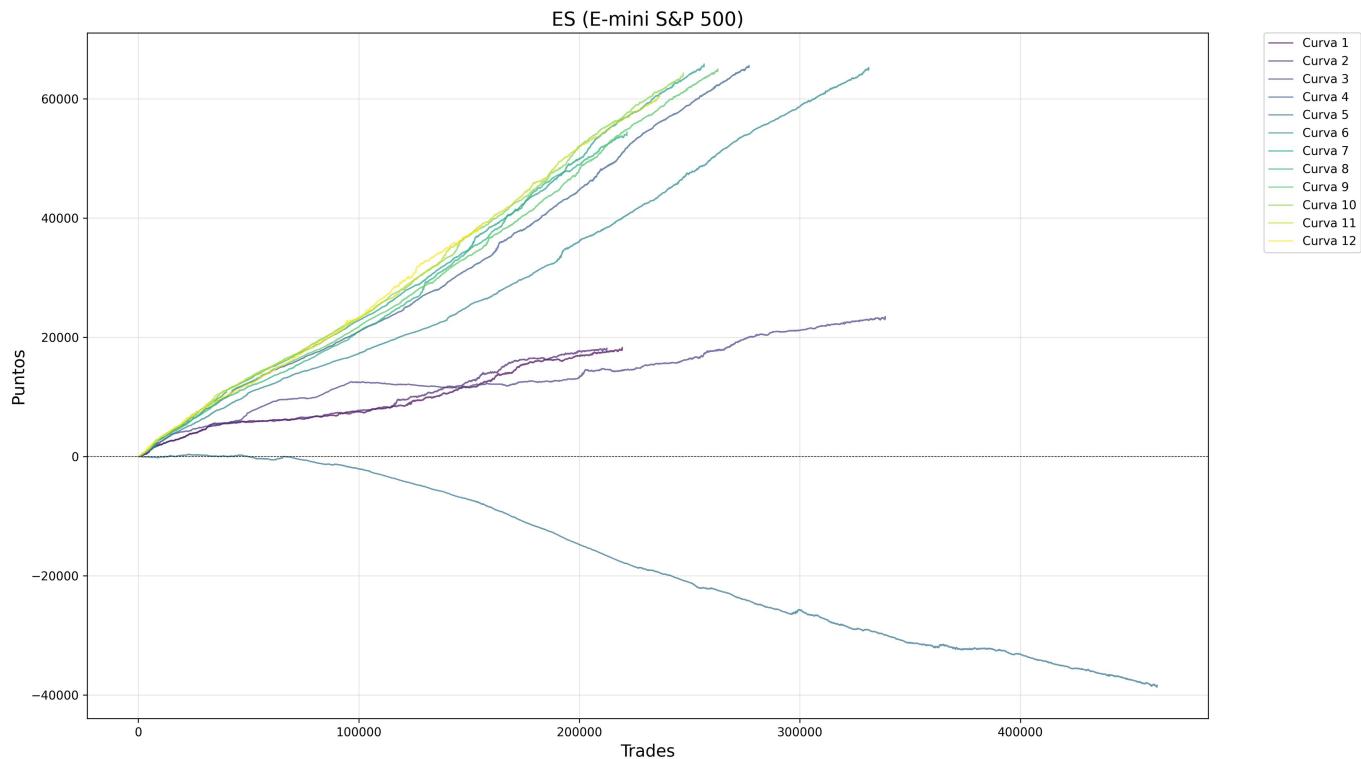
NQ:



Umbral= 800, variación mínima = 0,00625, R = 2,25, swing largo = 2, swing corto = 1.  $(270.262,64 * 20 \text{ USD}) - (638.073 * 2,25 \text{ USD} * 2) = 2.533.924,3 \text{ USD}$ , siendo estas las ganancias totales habiendo

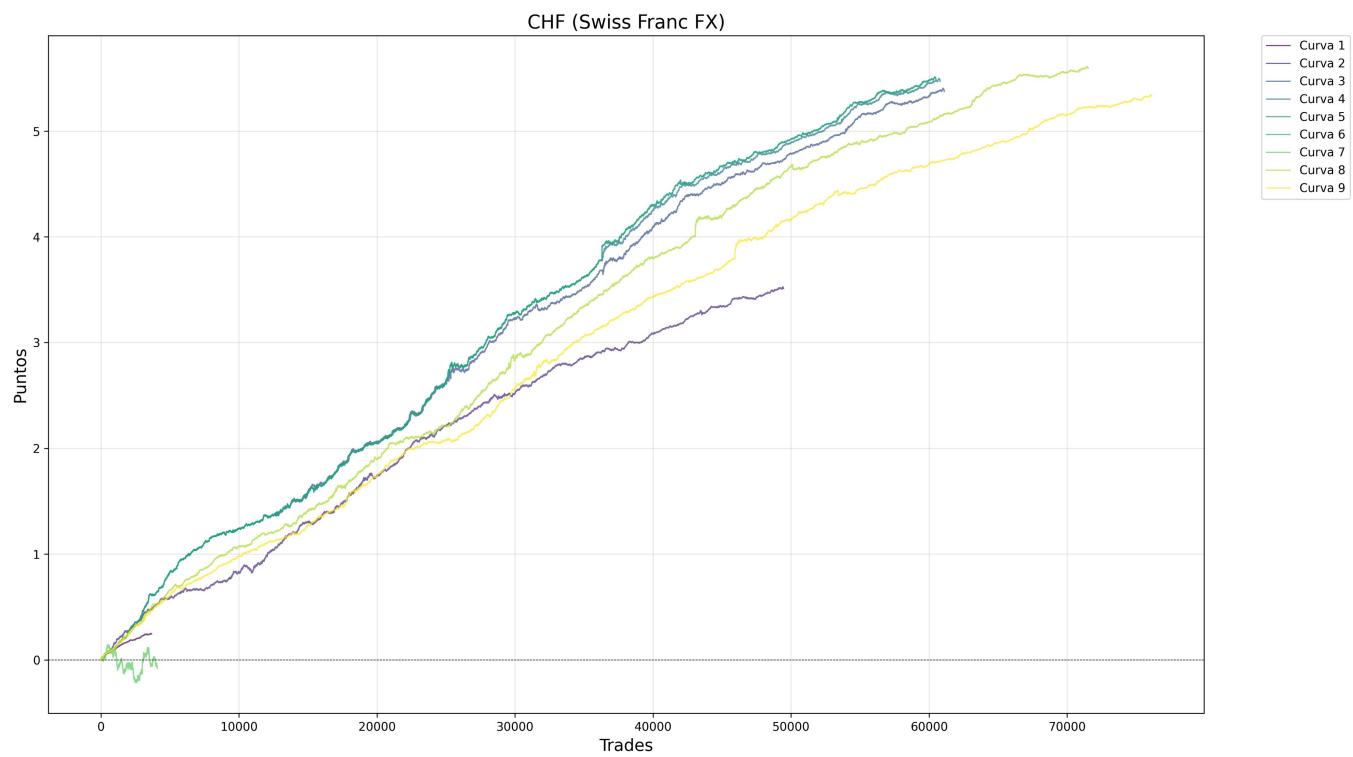
operado sobre 7.521.582 minutos, es decir unos 14,3 años. Siendo las ganancias medias por operación de 2.533.924,3 USD / 638.073 trades = **3,97 USD/trade**

ES:



Umbral= 800, variación mínima = 0,00625, R = 2,75, swing largo = 2, swing corto = 1.  $(65.767,21 * 50 \text{ USD}) - (256.609 * 2,25 \text{ USD} * 2) = 2.134.620 \text{ USD}$ , y al ser 2 contratos por orden las ganancias totales son **4.269240 USD** habiendo operado sobre 4.533.244 minutos, es decir unos 8,6 años. Siendo las ganancias medias por operación de 4.269.240 USD / 256.609 trades = **16,63 USD/trade**

Añadimos también el CHF, el futuro sobre el franco suizo, negociados en CME con un margen por contrato de 5.000 USD:



Umbral= 2, variación mínima = 0,00005, R = 3, swing largo = 60, swing corto = 3.  $(5,48315 * 125.000 \text{ USD}) - (60.502 * 2,47 \text{ USD} * 2) = 386.513,87 \text{ USD}$ , y al ser 8 contratos por orden las ganancias totales son **3.092.110,96 USD** habiendo operado sobre 4.760.346 minutos, es decir unos 9,06 años. Siendo las ganancias medias por operación de  $3.092.110,96 \text{ USD} / 60.502 \text{ trades} = \textbf{51,11 USD/trade}$

Símbolo	Contratos por orden	Comisión por contrato (USD)	Ganancias totales (USD)	Ganancia media por trade (USD)
RTY	4	2,25	1.634.926	5,82
CL	2	2,37	3.335.159,32	14,70
NQ	1	2,25	2.533.924,30	3,97
ES	2	2,25	4.269.240	16,63
CHF	8	2,47	3.092.110,96	51,11

## 4. CONCLUSIONES Y LÍNEAS FUTURAS

### 4.1. CONCLUSIONES

A lo largo de este trabajo se ha desarrollado un sistema de trading algorítmico capaz de operar de forma autónoma en los mercados financieros mediante el uso de contratos de futuros. Se ha implementado un enfoque estructurado basado en programación en Python y en el uso de la API de Interactive Brokers, permitiendo automatizar todo el proceso de recogida de datos, análisis, ejecución de órdenes y gestión operativa.

El proyecto ha cumplido con los objetivos planteados inicialmente. En primer lugar, se ha logrado construir una estrategia programable y reproducible, capaz de evaluar datos históricos y adaptarse a distintos activos financieros. Además, se ha diseñado un sistema robusto de almacenamiento y actualización continua de datos, fundamental para mantener la consistencia del análisis técnico. La integración con Trader Workstation (TWS) y/o IB Gateway se ha llevado a cabo con éxito, permitiendo enviar órdenes al mercado en tiempo real y recoger información precisa sobre su ejecución.

Una parte esencial del desarrollo ha sido la gestión de la estabilidad del sistema, especialmente ante eventuales pérdidas de conexión con la plataforma de *trading*. Para ello, se han implementado mecanismos automáticos de reconexión y validación del estado del sistema, asegurando la continuidad operativa y minimizando los riesgos de fallo.

Además, se ha creado una estructura modular que facilita la prueba y optimización de estrategias mediante datos históricos, lo que permite evaluar el comportamiento del sistema bajo diferentes condiciones de mercado. Este entorno de pruebas constituye un primer paso hacia una posible futura expansión del proyecto, ya sea mediante la incorporación de inteligencia artificial, *backtesting* más avanzado o el acceso a otros mercados y productos financieros.

Durante el desarrollo se ha utilizado la API de Interactive Brokers (concretamente mediante la librería `ib_insync` en Python), lo que ha permitido automatizar completamente el proceso de conexión, suscripción a datos en tiempo real, gestión de posiciones y ejecución de órdenes.

#### Ventajas:

- **Acceso directo y completo a funciones de trading:** permite crear y gestionar órdenes complejas (bracket orders, trailing stops, OCO, etc.) con gran flexibilidad.
- **Integración sencilla con herramientas de análisis en Python:** es compatible con librerías como Pandas o NumPy, lo que facilita la implementación de estrategias basadas en datos históricos y análisis en tiempo real.
- **Asincronía y eficiencia:** con `ib_insync`, la gestión asíncrona de múltiples contratos es estable y escalable, ideal para operar en varios mercados simultáneamente.
- **Coste cero por uso de la API:** a diferencia de otras plataformas, IB no cobra por acceder a su API, lo que la hace muy atractiva para pequeños desarrolladores o investigadores.

#### Desventajas:

- **Dependencia de TWS o IB Gateway:** la API requiere que uno de estos programas esté siempre activo, lo que puede generar problemas de desconexión, reinicio o actualización automática.
- **Complejidad inicial de configuración:** la documentación oficial puede resultar poco accesible para principiantes, y la curva de aprendizaje no es trivial.

- **Limitaciones en los datos históricos:** el acceso a datos pasados es restringido y puede requerir pausas o segmentación para evitar bloqueos por parte de IB.
- **Riesgos en tiempo real:** si no se gestiona correctamente la asincronía, pueden producirse condiciones de carrera o errores de ejecución difíciles de depurar.

En conjunto, el uso de la API de IB en Python es una herramienta potente y versátil para desarrollar sistemas de trading algorítmico, especialmente cuando se combina con buenas prácticas de programación y control de errores. Sin embargo, requiere una arquitectura robusta y un monitoreo constante para asegurar su estabilidad operativa.

En resumen, el trabajo demuestra que es posible construir una base sólida para un sistema de trading algorítmico funcional y escalable utilizando herramientas de código abierto y plataformas profesionales como Interactive Brokers. Aunque todavía existen áreas de mejora y desarrollo, los resultados obtenidos son prometedores y sientan las bases para investigaciones o aplicaciones más complejas en el ámbito del trading automatizado.

## 4.2. LÍNEAS FUTURAS

En futuras fases de desarrollo de este proyecto, se plantea la posibilidad de implementar un sistema de *money management* que permita ajustar el número de contratos gestionados por el algoritmo en función de la evolución del saldo disponible en la cuenta. De este modo, el sistema podría aumentar progresivamente el tamaño de las posiciones conforme se acumulan beneficios, aplicando un esquema de crecimiento controlado. Este enfoque permitiría aprovechar las rachas positivas para incrementar la rentabilidad, al tiempo que se podrían establecer límites para reducir el tamaño de las posiciones en caso de que el sistema experimente una serie de pérdidas, contribuyendo así a una gestión más robusta del riesgo.

Entre las opciones a valorar se encuentran la utilización de un modelo de riesgo fijo por operación (en el que se arriesga un porcentaje determinado del capital en cada operación), la aplicación de técnicas basadas en fracciones óptimas, o la adopción de un esquema escalonado que incremente el número de contratos al superar ciertos umbrales de beneficio acumulado. Estas técnicas formarían parte de un módulo adicional de gestión monetaria que reforzaría la seguridad y la eficiencia del sistema automatizado.

Tambien el uso del volumen, que es el número de contratos que se han negociado en un intervalo de tiempo concreto para un activo financiero determinado, es una de las mejoras que se podrían implementar en el sistema. Hasta ahora simplemente detectamos la situación de doble máximo/mínimo siempre y cuando se cumpla la condición del umbral, la idea sería incluir un nuevo condicionante que nos aporte más evidencias del posible éxito de ese movimiento, se ha observado que en el swing chart, si el volumen es mayor en el instante posterior que en el anterior es más probable que el doble máximo/mínimo se cumpla, consistiría en comparar ambos volúmenes, cada uno habiéndosele aplicado un promediado temporal para que esta comparación sea consistente con el tiempo transcurrido entre swings consecutivos.

## 5. BIBLIOGRAFÍA

- [1] «API Interactive Brokers», [en línea]. Available: <https://ib-insync.readthedocs.io/api.html>.
- [2] David E. Bowden, «Safety in the market, the smarter starter pack»
- [3] «Clasificación de los tipos de trading financieros: comparación entre el tradicional, el algorítmico y el cuantitativo.», [en línea]. Available: <https://fazilcrypto.com/blog/trader-algoritmico/>
- [4] [en línea]. Available: <https://www.cmegroup.com/>
- [5] [en línea]. Available: <https://www.bolsasymercados.es/es/home.html>

## ANEXO A: ASPECTOS ÉTICOS, ECONÓMICOS, SOCIALES Y AMBIENTALES

### A.1 INTRODUCCIÓN

El presente proyecto se enmarca en el ámbito del *trading* algorítmico aplicado a los mercados financieros, y tiene como objetivo principal el desarrollo de un sistema automatizado capaz de operar con contratos de futuros. Este tipo de tecnología responde a la necesidad de mejorar la eficiencia en la operativa financiera, reduciendo la intervención manual, minimizando errores y optimizando la ejecución de estrategias de inversión.

Desde el punto de vista económico, el sistema contribuye a la reducción de costes asociados a la gestión de órdenes y al análisis de datos, aspectos clave para profesionales y pequeñas empresas del sector financiero. En el ámbito social y ético, el proyecto plantea desafíos relacionados con el uso responsable de tecnologías de automatización en los mercados y la necesidad de respetar los principios de integridad, transparencia y equidad. Aunque el impacto ambiental directo del sistema es reducido, se ha considerado el consumo de recursos tecnológicos y el uso eficiente de la infraestructura computacional como elementos relevantes.

El proyecto se ha desarrollado cumpliendo con la normativa vigente en materia de uso de datos financieros y plataformas de trading, respetando los marcos legales y regulatorios aplicables a este tipo de actividad.

### A.2 DESCRIPCIÓN DE IMPACTOS RELEVANTES RELACIONADOS CON EL PROYECTO

El desarrollo de un sistema de *trading* algorítmico presenta una serie de impactos que han sido identificados y analizados durante el proyecto:

- Impactos económicos: el proyecto puede facilitar el acceso de pequeños operadores y desarrolladores a herramientas que tradicionalmente estaban limitadas a grandes instituciones, contribuyendo así a una mayor democratización de los recursos tecnológicos en los mercados financieros. No obstante, también existe el riesgo de incrementar la competencia y la presión en el entorno de trading, lo que puede afectar a determinados actores del mercado.
- Impactos sociales y éticos: la automatización de operaciones financieras debe gestionarse con responsabilidad para evitar contribuir a la generación de volatilidad innecesaria o a la creación de desigualdades en el acceso a las oportunidades del mercado. Además, el uso de algoritmos en mercados financieros requiere un compromiso ético claro para evitar prácticas abusivas o manipuladoras.
- Impactos ambientales: aunque el impacto directo es bajo, el consumo energético asociado al uso continuo de plataformas de *trading* y servidores para el procesamiento de datos y la ejecución de algoritmos es un aspecto que debe tenerse en cuenta, fomentando la eficiencia en el diseño y operación del sistema.

Entre los grupos de interés más relevantes se encuentran los usuarios y operadores de la plataforma (*traders* e instituciones financieras), los organismos reguladores del mercado y los proveedores de tecnología.

### A.3 ANÁLISIS DETALLADO DE ALGUNO DE LOS PRINCIPALES IMPACTOS

En el presente apartado se analiza con mayor detalle el impacto económico que puede generar el desarrollo e implantación del sistema automatizado de trading propuesto. Este impacto se manifiesta a distintos niveles y afecta a diversos grupos de interés del ecosistema financiero.

En primer lugar, el proyecto contribuye a reducir los costes de operación para los usuarios que empleen el sistema. Gracias a la automatización de tareas, como el análisis de datos de mercado, la colocación de órdenes y el seguimiento de las operaciones abiertas, se eliminan una gran parte de los costes asociados a la intervención manual y al tiempo de dedicación de los operadores. Esto supone una ventaja competitiva significativa, especialmente para pequeños inversores o desarrolladores independientes que, de otro modo, no podrían competir con grandes instituciones dotadas de amplios recursos humanos y tecnológicos.

Además, el sistema ofrece un potencial de escalabilidad económica. La posibilidad de operar de forma continua y sobre múltiples activos o contratos sin un incremento proporcional de costes operativos permite que el usuario pueda aumentar su volumen de actividad sin necesidad de asumir nuevas inversiones en personal o infraestructura. Esto genera oportunidades de crecimiento económico, tanto para individuos como para pequeñas empresas que deseen ampliar sus operaciones en los mercados financieros.

Por otro lado, la democratización de este tipo de herramientas puede tener un efecto dinamizador en los mercados, al incrementar la participación de nuevos actores con capacidad de operar de forma competitiva. Sin embargo, esto también podría traducirse en un aumento de la presión competitiva, lo que en determinados casos podría afectar a operadores tradicionales o menos tecnificados, que podrían verse obligados a adaptar sus procesos o asumir un mayor riesgo para mantener su rentabilidad.

Finalmente, cabe destacar que el proyecto, al centrarse en el uso de plataformas e infraestructuras tecnológicas ya existentes (como Trader Workstation o IB Gateway), minimiza la necesidad de grandes inversiones adicionales en hardware o software propietario. Esto permite un aprovechamiento eficiente de los recursos disponibles, optimizando así el retorno de la inversión asociada al desarrollo del sistema. A largo plazo, el impacto económico positivo podría incrementarse si se complementa el sistema con módulos de gestión avanzada de capital (por ejemplo, esquemas de money management adaptativos) que ajusten dinámicamente el tamaño de las posiciones en función de las ganancias acumuladas y del riesgo asumido.

### A.4 CONCLUSIONES

El desarrollo de este proyecto ha permitido reflexionar y aplicar criterios éticos, sociales, económicos y medioambientales desde las primeras fases de diseño. El respeto a la normativa y el uso responsable de la tecnología han sido principios fundamentales durante el desarrollo. Se ha procurado que el sistema opere de forma transparente y justa, evitando introducir prácticas que pudieran suponer un riesgo para la integridad de los mercados o la equidad entre los participantes.

Desde un punto de vista económico, el sistema puede aportar valor al ofrecer soluciones automatizadas asequibles para distintos perfiles de usuarios.

En términos ambientales, el proyecto demuestra que es posible diseñar herramientas eficientes que minimicen el uso de recursos computacionales mediante una programación optimizada y el uso de infraestructuras ya existentes.

En definitiva, la integración de criterios de sostenibilidad y responsabilidad en el desarrollo del proyecto ha aportado un valor añadido significativo, no solo en términos de cumplimiento de la normativa, sino también en la creación de una solución alineada con los principios de ingeniería ética y sostenible.

## ANEXO B: PRESUPUESTO ECONÓMICO

### COSTE DE MANO DE OBRA (coste directo)

Horas	Precio/hora	Total
360	20 €	<b>7.200 €</b>

### COSTE DE RECURSOS MATERIALES (coste directo)

	Precio de compra	Uso en meses	Amortización (en años)	Total
Ordenador personal (Software incluido)	500,00 €	6	5	50,00 €

### COSTE TOTAL DE RECURSOS MATERIALES

**50,00 €**

### GASTOS GENERALES (costes indirectos)

15% sobre CD **1.087,50 €**

### BENEFICIO INDUSTRIAL

6% sobre CD+CI **500,25 €**

### SUBTOTAL PRESUPUESTO

**8.837,75 €**

### IVA APPLICABLE

21% **1.855,93 €**

### TOTAL PRESUPUESTO

**10.693,68 €**

## ANEXO C: PROCESADO.PY

```

from ib_insync import *
from datetime import datetime, timezone, timedelta
from zoneinfo import ZoneInfo
import struct
import numpy as np
import os

def borrar_ultimas_n_lineas_min(nombre_archivo, n_lineas):
    TAM_REGISTRO = struct.calcsize("ii f f f f ii") # 32 bytes por registro
    """
    Borra las últimas `n_lineas` de un archivo .min manteniendo la estructura binaria.

    Parámetros:
        nombre_archivo (str): Ruta del archivo .min.
        n_lineas (int): Número de registros a eliminar (los más recientes).
    """
    try:
        with open(nombre_archivo, "rb") as f:
            datos = f.read() # Leer todo el archivo en memoria

        total_registros = len(datos) // TAM_REGISTRO # Calcular cuántos registros hay

        if total_registros == 0:
            print("El archivo está vacío, no hay nada que borrar.")
            return

        if n_lineas >= total_registros:
            print("Intentaste borrar más registros de los que existen. Se eliminará todo el archivo.")
            os.remove(nombre_archivo) # Borra completamente el archivo
            return

        # Obtener solo la parte inicial (excluyendo los últimos `n_lineas` registros)
        datos_restantes = datos[: (total_registros - n_lineas) * TAM_REGISTRO]

        # Sobrescribir el archivo con los datos restantes
        with open(nombre_archivo, "wb") as f:
            f.write(datos_restantes)

        #print(f"Se eliminaron las últimas {n_lineas} líneas de {nombre_archivo} correctamente.")

    except Exception as e:
        print(f"Error al borrar líneas: {e}")

```

```

def leer_min(file_path, zona_horaria):
    TAM_REGISTRO = struct.calcsize("ii f f f f ii") # 32 bytes
    caldayi = []
    tradayi = []
    hora = []
    hii = []
    loi = []
    cli = []
    opi = []
    voli = []

    with open(file_path, "rb") as f:
        f.seek(0, 0) # Asegurar que leemos desde el principio real

        while True:
            data = f.read(TAM_REGISTRO)
            if len(data) < TAM_REGISTRO:
                break # Evita errores si el archivo termina antes

            # Desempaquetar datos
            fecha, time, op, hi, lo, cl, vol, _ = struct.unpack("ii f f f f ii", data)

            # Convertir la fecha al formato datetime
            año = fecha // 500
            aux = fecha % 500
            mes = aux // 32
            dia = aux % 32
            hour = time // 3600
            minute = (time % 3600) // 60

            fecha_datetime = datetime(año, mes, dia, hour, minute,
tzinfo=ZoneInfo(zona_horaria))
            caldayi.append(fecha_datetime)
            tradayi.append(len(caldayi))
            opi.append(op)
            hii.append(hi)
            loi.append(lo)
            cli.append(cl)
            voli.append(vol)

    return caldayi, tradayi, hii, loi, cli, opi, voli

def swing1(hi, lo, cl, op, dhi, dlo):
    dias = len(hi)
    swings = []
    trasw = []
    calsw = []

    for i in range(dias):
        if hi[i] > dhi:
            swings.append((dhi, hi[i]))
            trasw.append((dhi, hi[i]))
            calsw.append((dhi, hi[i]))
        elif hi[i] < dlo:
            swings.append((hi[i], dlo))
            trasw.append((hi[i], dlo))
            calsw.append((hi[i], dlo))
        else:
            swings.append((hi[i], hi[i]))
            trasw.append((hi[i], hi[i]))
            calsw.append((hi[i], hi[i]))
```

```

if op[0] < cl[0]:
    swings.extend([lo[0], hi[0]])
    trasw.extend([1, 1])
    calsw.extend([dlo[0], dhi[0]])
    direccion = 1
else:
    swings.extend([hi[0], lo[0]])
    trasw.extend([1, 1])
    calsw.extend([dhi[0], dlo[0]])
    direccion = -1

indiceswing = 1

for i in range(dias - 1):
    if (hi[i + 1] > hi[i]) and (lo[i + 1] < lo[i]): # outside day
        if dlo[i + 1] < dhi[i + 1]:
            if (direccion == 1 and lo[i + 1] < swings[indiceswing]):
                direccion = -1
                swings.append(lo[i + 1])
                trasw.append(i + 1)
                calsw.append(dlo[i + 1])
                indiceswing += 1
            elif (direccion == -1 and lo[i + 1] < swings[indiceswing]):
                swings[indiceswing] = lo[i + 1]
                trasw[indiceswing] = i + 1
                calsw[indiceswing] = dlo[i + 1]

            direccion = 1
            swings.append(hi[i + 1])
            trasw.append(i + 1)
            calsw.append(dhi[i + 1])
            indiceswing += 1
        else:
            if (direccion == -1 and hi[i + 1] > swings[indiceswing]):
                direccion = 1
                swings.append(hi[i + 1])
                trasw.append(i + 1)
                calsw.append(dhi[i + 1])
                indiceswing += 1
            elif (direccion == 1 and hi[i + 1] > swings[indiceswing]):
                swings[indiceswing] = hi[i + 1]
                trasw[indiceswing] = i + 1
                calsw[indiceswing] = dhi[i + 1]

            direccion = -1
            swings.append(lo[i + 1])
            trasw.append(i + 1)

```

```

        calsw.append(dlo[i + 1])
        indiceswing += 1
    elif hi[i + 1] > hi[i]:
        if (direccion == -1 and hi[i + 1] > swings[indiceswing]):
            direccion = 1
            swings.append(hi[i + 1])
            trasw.append(i + 1)
            calsw.append(dhi[i + 1])
            indiceswing += 1
        elif (direccion == 1 and hi[i + 1] > swings[indiceswing]):
            swings[indiceswing] = hi[i + 1]
            trasw[indiceswing] = i + 1
            calsw[indiceswing] = dhi[i + 1]
    elif lo[i + 1] < lo[i]:
        if (direccion == 1 and lo[i + 1] < swings[indiceswing]):
            direccion = -1
            swings.append(lo[i + 1])
            trasw.append(i + 1)
            calsw.append(dlo[i + 1])
            indiceswing += 1
        elif (direccion == -1 and lo[i + 1] < swings[indiceswing]):
            swings[indiceswing] = lo[i + 1]
            trasw[indiceswing] = i + 1
            calsw[indiceswing] = dlo[i + 1]

    return np.array(swings), np.array(calsw), np.array(trasw)

def dividir_min_en_grupos_para_swings(caldayi, hii, loi, cli, opi, voli, n_min_barras,
n_grupos, inicio_relativo):
    """
    Divide los datos en grupos hacia atrás y calcula métricas de swing simultáneamente.

    Devuelve:
    - grupos: Lista de diccionarios con los datos crudos de cada grupo
    - hi_swings: Array con los máximos high de cada grupo
    - lo_swings: Array con los mínimos low de cada grupo
    - op_swings: Array con los opens más antiguos (primer elemento)
    - cl_swings: Array con los closes más recientes (último elemento)
    - dhi_swings: Array con las fechas donde ocurrieron los máximos
    - dlo_swings: Array con las fechas donde ocurrieron los mínimos
    """
    num_barras = len(caldayi)
    total_min_deseados = n_min_barras * n_grupos

    # Validaciones
    if num_barras < total_min_deseados:

```

```

        raise ValueError("No hay suficientes datos para formar los grupos
especificados.")

    if inicio_relativo is None:
        inicio_relativo = num_barras

    if inicio_relativo < total_min_deseados:
        raise ValueError(f"El inicio_relativo ({inicio_relativo}) es insuficiente para
{n_grupos} grupos de {n_min_barras} mins.")

    # Inicializamos arrays para los resultados
    hi_swings = np.empty(n_grupos)
    lo_swings = np.empty(n_grupos)
    op_swings = np.empty(n_grupos)
    cl_swings = np.empty(n_grupos)
    dhi_swings = np.empty(n_grupos, dtype=caldayi.dtype)
    dlo_swings = np.empty(n_grupos, dtype=caldayi.dtype)

    fin_segmento = inicio_relativo + 1
    inicio_segmento = fin_segmento - total_min_deseados

    for i in range(n_grupos):
        # Calculamos los índices del grupo actual

        grupo_inicio = i * n_min_barras
        grupo_fin = grupo_inicio + n_min_barras
        # Extraemos el grupo
        hi_grupo = hii[inicio_segmento + grupo_inicio : inicio_segmento + grupo_fin]
        loi_grupo = loi[inicio_segmento + grupo_inicio : inicio_segmento + grupo_fin]

        # Calculamos métricas
        max_hi_idx = np.argmax(hi_grupo)
        min_loi_idx = np.argmin(loi_grupo)

        # Almacenamos resultados
        hi_swings[i] = hi_grupo[max_hi_idx]
        lo_swings[i] = loi_grupo[min_loi_idx]
        op_swings[i] = opi[inicio_segmento + grupo_inicio] # Primer open
        cl_swings[i] = cli[inicio_segmento + grupo_fin - 1] # Último close
        dhi_swings[i] = caldayi[inicio_segmento + grupo_inicio + max_hi_idx]
        dlo_swings[i] = caldayi[inicio_segmento + grupo_inicio + min_loi_idx]

    return hi_swings, lo_swings, cl_swings, op_swings, dhi_swings, dlo_swings

def actualizar_archivo_min(ib, file_path, contract, zona_horaria):
    """Actualiza el archivo .min con nuevos datos."""

```

```

def leer_min(file_path):
    """Lee el último registro del archivo .min y devuelve la fecha más reciente."""
    try:
        with open(file_path, "rb") as f:
            f.seek(-32, 2)
            data = f.read(32)
            if len(data) < 32:
                return None

            fecha, tiempo, *_ = struct.unpack("ii f f f f ii", data)
            ano = fecha // 500
            aux = fecha % 500
            mes = aux // 32
            dia = aux % 32
            hour = tiempo // 3600
            minute = (tiempo % 3600) // 60
            return datetime(ano, mes, dia, hour, minute, second=59,
tzinfo=ZoneInfo(zona_horaria)) # <- Añade second=59
    except FileNotFoundError:
        return None

def escribir_min(file_path, barras):
    """Escribe nuevas barras en el archivo .min evitando duplicados."""
    with open(file_path, "ab") as f:
        for bar in barras:
            dt = bar.date
            fecha = dt.year * 500 + dt.month * 32 + dt.day
            tiempo = dt.hour * 3600 + dt.minute * 60

            data = struct.pack("ii f f f f ii", fecha, tiempo, bar.open, bar.high,
bar.low, bar.close, int(bar.volume), 0)
            f.write(data)

    ultima_fecha = leer_min(file_path)
    print(f"Última fecha en el archivo de {contract.symbol}: {ultima_fecha}")
    if ultima_fecha is None:
        print(f"Error al leer el archivo {file_path}")
        return

    ahora_utc = datetime.now(timezone.utc)
    diferencia_segundos = int((ahora_utc -
ultima_fecha.astimezone(timezone.utc)).total_seconds())
    dias = diferencia_segundos // 86400
    segundos_restantes = diferencia_segundos % 86400
    barras_totales = []

    def filtrar_nuevas_barras(barras):
        return [bar for bar in barras if bar.date >= ultima_fecha]

```

```

# **Solicitar datos históricos**
if dias > 0:
    bars = ib.reqHistoricalData(
        contract,
        endDateTime='',
        durationStr=f'{dias} D',
        barSizeSetting='1 min',
        whatToShow='TRADES',
        useRTH=False,
        formatDate=1
    )
    if bars:
        barras_totales.extend(filtrar_nuevas_barras(bars))

    if segundos_restantes > 0:
        bars = ib.reqHistoricalData(
            contract,
            endTime='',
            durationStr=f'{segundos_restantes} S',
            barSizeSetting='1 min',
            whatToShow='TRADES',
            useRTH=False,
            formatDate=1
        )
        if bars:
            barras_totales.extend(filtrar_nuevas_barras(bars))

    if barras_totales:
        escribir_min(file_path, barras_totales)
        borrar_ultimas_n_lineas_min(file_path, 1)
        print(f"Se han agregado {len(barras_totales)} registros al archivo .min de {contract.symbol}.")
    else:
        print(f"No hay nuevos registros para agregar en {contract.symbol}.")

```

## ANEXO D: ESTRATEGIA\_LIMIT.PY

```

from variables_por_contrato import *

def procesar_doble_top_bottom(swingsy, calswy, umbral, ib, contract, orden_activa,
high_barra_actual, low_barra_actual, client_id, cantidad, variacion_minima, R_multiplier,
num_decimales, zona_horaria):

    from ib_insync import StopOrder, LimitOrder
    from datetime import datetime, timedelta
    from datetime import timezone

```

```

from zoneinfo import ZoneInfo

tiempo_max_espera = timedelta(minutes=5) # Máximo 5 minutos entre detección y entrada

lsw = len(swingsy)

if orden_activa.get('entry_order') is not None:
    # Si ya hay una orden enviada, verificamos si fue ejecutada o hay que cancelarla
    if orden_activa.get('entry_order').isActive() == False:
        entry_order = orden_activa.get('entry_order')
        tp_trade = orden_activa.get('tp_order')
        sl_trade = orden_activa.get('sl_order')

        if tp_trade and tp_trade.orderStatus.status == 'Filled':
            print("Take Profit ejecutado.")
            orden_activa.update({'enviada': False, 'entry_order': None, 'tipo': None,
'tp_order': None, 'sl_order': None, 'take_profit_price': None, 'stop_loss_price': None,
'en_espera': False})

        elif sl_trade and sl_trade.orderStatus.status == 'Filled':
            print("Stop Loss ejecutado.")
            orden_activa.update({'enviada': False, 'entry_order': None, 'tipo': None,
'tp_order': None, 'sl_order': None, 'take_profit_price': None, 'stop_loss_price': None,
'en_espera': False})

        elif entry_order.orderStatus.status == 'Filled':
            print("Entrada ejecutada. Esperando a que se ejecuten TP o SL...")
            return

        elif entry_order.orderStatus.status in ('Cancelled', 'Inactive'):
            print("Orden cancelada o inactiva. Se puede buscar nueva entrada.")
            orden_activa.update({'enviada': False, 'entry_order': None, 'tipo': None,
'tp_order': None, 'sl_order': None, 'take_profit_price': None, 'stop_loss_price': None,
'en_espera': False})

    else:
        entry_order = orden_activa.get('entry_order')
        try:
            if orden_activa.get('en_espera') == True:
                print("Orden en espera, no se cancela.")
                return

            if orden_activa.get('tipo') == 'double_top':
                if swingsy[-1] > swingsy[-2]: # posible double top

```

```

        for k in range(1, (lsw - 1) // 2 + 1):
            dif = swingsy[-1] - swingsy[(-2 * k)-1]
            if abs(dif) < umbral:
                print("Cancelando orden anterior que no se ejecutó...")
                ib.cancelOrder(entry_order.order)
            orden_activa.update({
                'enviada': False,
                'entry_order': None,
                'tipo': None,
                'tp_order': None,
                'sl_order': None,
                'take_profit_price': None,
                'stop_loss_price': None,
                'en_espera': False
            })
            break
        elif dif <= -umbral:
            orden_activa['en_espera'] = True
            return
        else:
            orden_activa['en_espera'] = True
            return
    else:
        orden_activa['en_espera'] = True
        return

else: # possible double bottom
    if swingsy[-1] < swingsy[-2]: # possible double bottom
        for k in range(1, (lsw - 1) // 2 + 1):
            dif = swingsy[-1] - swingsy[(-2 * k)-1]
            if abs(dif) < umbral:
                print("Cancelando orden anterior que no se ejecutó...")
                ib.cancelOrder(entry_order.order)
            orden_activa.update({
                'enviada': False,
                'entry_order': None,
                'tipo': None,
                'tp_order': None,
                'sl_order': None,
                'take_profit_price': None,
                'stop_loss_price': None,
                'en_espera': False
            })
            break
        elif dif >= umbral:
            orden_activa['en_espera'] = True
            return

```

```

        else:
            orden_activa['en_espera'] = True
            return

        else:
            orden_activa['en_espera'] = True
            return

    except Exception as e:
        print(f"Error durante el proceso de cancelación: {str(e)}")
        # No es critico si falla la cancelación

# Si no hay orden activa, buscamos nuevos patrones
if lsw >= 3:
    tipo = None
    swing_detectado = None
    tiempo_deteccion = None

    if swingsy[-1] < swingsy[-2]: # posible double bottom
        for k in range(1, (lsw - 1) // 2 + 1):
            dif = swingsy[-1] - swingsy[(-2 * k)-1]
            if abs(dif) < umbral:
                tipo = 'double_bottom'
                swing_detectado = swingsy[-1] # Precio del swing
                tiempo_deteccion = calswy[-1] # Hora de detección (datetime)
                break
            elif dif >= umbral:
                break

    elif swingsy[-1] > swingsy[-2]: # posible double top
        for k in range(1, (lsw - 1) // 2 + 1):
            dif = swingsy[-1] - swingsy[(-2 * k)-1]
            if abs(dif) < umbral:
                tipo = 'double_top'
                swing_detectado = swingsy[-1]
                tiempo_deteccion = calswy[-1]
                break
            elif dif <= -umbral:
                break

    # --- NUEVAS VALIDACIONES ---
    if tipo and swing_detectado and tiempo_deteccion:
        # 1. Verificar que no hayan pasado más de 5 minutos desde la detección
        tiempo_actual_utc = datetime.now(timezone.utc)
        tiempo_actual_local = tiempo_actual_utc.astimezone(ZoneInfo(zona_horaria)) # Ajustar a la zona horaria deseada
        tiempo_transcurrido = tiempo_actual_local - tiempo_deteccion

```

```

        if tiempo_transcurrido > tiempo_max_espera:
            print(f"⚠ {tipo} detectado hace {tiempo_transcurrido}. Demasiado tarde
para entrar.")
            return

        # 2. Verificar que el precio actual está dentro del rango permitido
        precio_actual = high_barra_actual if tipo == 'double_bottom' else
low_barra_actual
        distancia_al_swing = abs(precio_actual - swing_detectado)

        if distancia_al_swing > umbral:
            print(f"⚠ {tipo} detectado en {swing_detectado}, pero precio actual
({precio_actual}) está fuera de rango.")
            return

        # --- LÓGICA DE ENTRADA (código existente, ahora con validaciones) ---
        print(f"☑{tipo.replace('_', ' ')} válido. Swing: {swing_detectado} a las
{tiempo_deteccion} y {swingsy[(-2 * k)-1]} a las {calswy[(-2 * k)-1]}, Precio actual:
{precio_actual}")
        order_timer = (datetime.now() + timedelta(minutes=5)).strftime("%H:%M:%S")

        if tipo == 'double_top':
            action = 'BUY'
            quantity = cantidad
            entry_limit_price = round(low_barra_actual - variacion_minima,
num_decimales) # ej: calculado a partir del swing
            take_profit_price = round(swingsy[-1] + variacion_minima, num_decimales)
            stop_loss_price = round(entry_limit_price - (R_multiplier *
(take_profit_price - entry_limit_price)), num_decimales)
            # Orden de entrada
            orden_entrada = LimitOrder(
                action=action,
                totalQuantity=quantity,
                lmtPrice=entry_limit_price,
                tif='GTD',
                goodTillDate=order_timer,
                outsideRth=True,
                transmit=False,
                orderId=ib.client.getReqId()
            )

            elif tipo == 'double_bottom':
                action = 'SELL'
                quantity = cantidad
                entry_limit_price = round(high_barra_actual + variacion_minima,
num_decimales)
                take_profit_price = round(swingsy[-1] - variacion_minima, num_decimales)

```

```

        stop_loss_price = round(entry_limit_price + (R_multiplier *
(entry_limit_price - take_profit_price)), num_decimales)
        # Orden de entrada
        orden_entrada = LimitOrder(
            action=action,
            totalQuantity=quantity,
            lmtPrice=entry_limit_price,
            tif='GTD',
            goodTillDate=order_timer,
            outsideRth=True,
            transmit=False,
            orderId=ib.client.getReqId()
        )

        tp = LimitOrder(
            action='SELL' if action == 'BUY' else 'BUY',
            totalQuantity=quantity,
            lmtPrice=take_profit_price, # Precio take-profit
            outsideRth=True,
            parentId=orden_entrada.orderId,
            transmit=False,
            orderId=ib.client.getReqId()
        )

        sl = StopOrder(
            action='SELL' if action == 'BUY' else 'BUY',
            totalQuantity=quantity,
            stopPrice=stop_loss_price, # Precio stop-loss
            outsideRth=True,
            parentId=orden_entrada.orderId,
            transmit=True, # Última orden del bracket debe ser True
            orderId=ib.client.getReqId()
        )

        orden_activa.update({
            'enviada': True,
            'entry_order': ib.placeOrder(contract, orden_entrada),
            'tipo': tipo,
            'tp_order': ib.placeOrder(contract, tp),
            'sl_order': ib.placeOrder(contract, sl),
            'take_profit_price': take_profit_price,
            'stop_loss_price': stop_loss_price,
            'en_espera': False
        })
    )
}

```

## ANEXO E: ESTRATEGIA\_STOP.PY

```
from variables_por_contrato import *
```

```

def procesar_doble_top_bottom(swingsy, calswy, umbral, ib, contract, orden_activa,
high_barra_actual, low_barra_actual, client_id, cantidad, variacion_minima, R_multiplier,
num_decimales, zona_horaria):

    from ib_insync import StopOrder, LimitOrder
    from datetime import datetime, timedelta
    from datetime import timezone
    from zoneinfo import ZoneInfo

    tiempo_max_espera = timedelta(minutes=5) # Máximo 5 minutos entre detección y entrada

    lsw = len(swingsy)

    # Si ya hay una orden enviada, verificamos si fue ejecutada o hay que cancelarla
    if orden_activa['enviada']:

        if orden_activa['tp_order'] and orden_activa['tp_order'].orderStatus.status == 'Filled':
            print("Take Profit ejecutado.")
            orden_activa.update({'enviada': False, 'entry_order': None, 'tipo': None,
'tp_order': None, 'sl_order': None, 'take_profit_price': None, 'stop_loss_price': None,
'en_espera': False})

        elif orden_activa['sl_order'] and orden_activa['sl_order'].orderStatus.status == 'Filled':
            print("Stop Loss ejecutado.")
            orden_activa.update({'enviada': False, 'entry_order': None, 'tipo': None,
'tp_order': None, 'sl_order': None, 'take_profit_price': None, 'stop_loss_price': None,
'en_espera': False})

        elif orden_activa['entry_order'].orderStatus.status == 'Filled':
            print("Entrada ejecutada. Esperando a que se ejecuten TP o SL...")
            return

        elif orden_activa['entry_order'].orderStatus.status in ('Cancelled', 'Inactive'):
            print("Orden cancelada o inactiva. Se puede buscar nueva entrada.")
            orden_activa.update({'enviada': False, 'entry_order': None, 'tipo': None,
'tp_order': None, 'sl_order': None, 'take_profit_price': None, 'stop_loss_price': None,
'en_espera': False})

    else:
        try:

```

```
# Verificación basada en el estado de la orden
if (orden_activa['entry_order'].isActive() == True):

    if orden_activa['en_espera'] == True:
        print("Orden en espera, no se cancela.")
        return

    if orden_activa['tipo'] == 'double_top':
        if swingsy[-1] > swingsy[-2]: # posible double top
            for k in range(1, (lsw - 1) // 2 + 1):
                dif = swingsy[-1] - swingsy[(-2 * k)-1]
                if abs(dif) < umbral:
                    print("Cancelando orden anterior que no se ejecutó...")
                    ib.cancelOrder(orden_activa['entry_order'].order)
                    orden_activa.update({
                        'enviada': False,
                        'entry_order': None,
                        'tipo': None,
                        'tp_order': None,
                        'sl_order': None,
                        'take_profit_price': None,
                        'stop_loss_price': None,
                        'en_espera': False
                    })
                    break
            elif dif <= -umbral:
                orden_activa['en_espera'] = True
                return
            else:
                orden_activa['en_espera'] = True
                return

        else: # posible double bottom
            if swingsy[-1] < swingsy[-2]: # posible double bottom
                for k in range(1, (lsw - 1) // 2 + 1):
                    dif = swingsy[-1] - swingsy[(-2 * k)-1]
                    if abs(dif) < umbral:
                        print("Cancelando orden anterior que no se ejecutó...")
                        ib.cancelOrder(orden_activa['entry_order'].order)
                        orden_activa.update({
                            'enviada': False,
                            'entry_order': None,
                            'tipo': None,
                            'tp_order': None,
                            'sl_order': None,
```

```

        'take_profit_price': None,
        'stop_loss_price': None,
        'en_espera': False
    })
    break
elif dif >= umbral:
    orden_activa['en_espera'] = True
    return
else:
    orden_activa['en_espera'] = True
    return

else:
    orden_activa['en_espera'] = True
    return

except Exception as e:
    print(f"Error durante el proceso de cancelación: {str(e)}")
    # No es crítico si falla la cancelación


# Si no hay orden activa, buscamos nuevos patrones
if lsw >= 3:
    tipo = None
    swing_detectado = None
    tiempo_deteccion = None

    if swingsy[-1] < swingsy[-2]: # posible double bottom
        for k in range(1, (lsw - 1) // 2 + 1):
            dif = swingsy[-1] - swingsy[(-2 * k)-1]
            if abs(dif) < umbral:
                tipo = 'double_bottom'
                swing_detectado = swingsy[-1] # Precio del swing
                tiempo_deteccion = calswy[-1] # Hora de detección (datetime)
                break
            elif dif >= umbral:
                break

    elif swingsy[-1] > swingsy[-2]: # posible double top
        for k in range(1, (lsw - 1) // 2 + 1):
            dif = swingsy[-1] - swingsy[(-2 * k)-1]
            if abs(dif) < umbral:
                tipo = 'double_top'
                swing_detectado = swingsy[-1]
                tiempo_deteccion = calswy[-1]
                break
            elif dif <= -umbral:
                break

```

```

# --- NUEVAS VALIDACIONES ---
if tipo and swing_detectado and tiempo_deteccion:
    # 1. Verificar que no hayan pasado más de 5 minutos desde la detección
    tiempo_actual_utc = datetime.now(timezone.utc)
    tiempo_actual_local = tiempo_actual_utc.astimezone(ZoneInfo(zona_horaria)) # Ajustar a la zona horaria deseada
    tiempo_transcurrido = tiempo_actual_local - tiempo_deteccion

    if tiempo_transcurrido > tiempo_max_espera:
        print(f"⚠ {tipo} detectado hace {tiempo_transcurrido}. Demasiado tarde para entrar.")
        return

    # 2. Verificar que el precio actual está dentro del rango permitido
    precio_actual = high_barra_actual if tipo == 'double_bottom' else low_barra_actual
    distancia_al_swing = abs(precio_actual - swing_detectado)

    if distancia_al_swing > umbral:
        print(f"⚠ {tipo} detectado en {swing_detectado}, pero precio actual ({precio_actual}) está fuera de rango.")
        return

# --- LÓGICA DE ENTRADA (código existente, ahora con validaciones) ---
print(f"⚠{tipo.replace('_', ' ')} válido. Swing: {swing_detectado} a las {tiempo_deteccion} y {swingsy[(-2 * k)-1]} a las {calswy[(-2 * k)-1]}, Precio actual: {precio_actual}")
order_timer = (datetime.now() + timedelta(minutes=5)).strftime("%H:%M:%S")

if tipo == 'double_top':
    action = 'SELL'
    quantity = cantidad
    entry_stop_price = round(low_barra_actual - variacion_minima, num_decimales) # ej: calculado a partir del swing
    stop_loss_price = round(swingsy[-1] + variacion_minima, num_decimales)
    take_profit_price = round(entry_stop_price - (R_multiplier * (stop_loss_price - entry_stop_price)), num_decimales)
    # Orden de entrada
    entry_order = StopOrder(
        action=action,
        totalQuantity=quantity,
        stopPrice=entry_stop_price,
        tif='GTD',
        goodTillDate=order_timer,
        outsideRth=True,
        transmit=False,
        orderId=ib.client.getNextId()

```

```

        )

    elif tipo == 'double_bottom':
        action = 'BUY'
        quantity = cantidad
        entry_stop_price = round(high_barra_actual + variacion_minima,
num_decimales)
        stop_loss_price = round(swingsy[-1] - variacion_minima, num_decimales)
        take_profit_price = round(entry_stop_price + (R_multiplier *
(entry_stop_price - stop_loss_price)), num_decimales)
        # Orden de entrada
        entry_order = StopOrder(
            action=action,
            totalQuantity=quantity,
            stopPrice=entry_stop_price,
            tif='GTD',
            goodTillDate=order_timer,
            outsideRth=True,
            transmit=False,
            orderId=ib.client.getReqId()
        )

        take_profit = LimitOrder(
            action='SELL' if action == 'BUY' else 'BUY',
            totalQuantity=quantity,
            lmtPrice=take_profit_price, # Precio take-profit
            outsideRth=True,
            parentId=entry_order.orderId,
            transmit=False,
            orderId=ib.client.getReqId()
        )

        stop_loss = StopOrder(
            action='SELL' if action == 'BUY' else 'BUY',
            totalQuantity=quantity,
            stopPrice=stop_loss_price, # Precio stop-loss
            outsideRth=True,
            parentId=entry_order.orderId,
            transmit=True, # Última orden del bracket debe ser True
            orderId=ib.client.getReqId()
        )

        orden_activa['entry_order']=ib.placeOrder(contract, entry_order)
        orden_activa['tp_order']=ib.placeOrder(contract, take_profit)
        orden_activa['sl_order']=ib.placeOrder(contract, stop_loss)

        orden_activa['take_profit_price'] = take_profit_price
        orden_activa['stop_loss_price'] = stop_loss_price
    
```

```
orden_activa['enviada'] = True
orden_activa['tipo'] = tipo
```

## ANEXO F: MAIN\_SECUENCIAL.PY

```
import time
from ib_insync import *
import variables_por_contrato
from procesado import *
from estrategia_limit import procesar_doble_top_bottom

CONTRATOS = [
    {'symbol': 'ES', 'expiry': '202506', 'exchange': 'CME', 'client_id': 1},
    {'symbol': 'NQ', 'expiry': '202506', 'exchange': 'CME', 'client_id': 2},
    {'symbol': 'CL', 'expiry': '202507', 'exchange': 'NYMEX', 'client_id': 3},
    {'symbol': 'GC', 'expiry': '202506', 'exchange': 'COMEX', 'client_id': 4},
    {'symbol': 'IBEX35', 'expiry': '202506', 'exchange': 'MEFFRV', 'client_id': 5},
    {'symbol': 'HG', 'expiry': '202506', 'exchange': 'COMEX', 'client_id': 6},
    {'symbol': 'NG', 'expiry': '202506', 'exchange': 'NYMEX', 'client_id': 7},
    {'symbol': 'HO', 'expiry': '202506', 'exchange': 'NYMEX', 'client_id': 8},
    {'symbol': 'YM', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 9},
    {'symbol': 'RTY', 'expiry': '202506', 'exchange': 'CME', 'client_id': 10},
    {'symbol': 'JPY', 'expiry': '202506', 'exchange': 'CME', 'client_id': 11},
    {'symbol': 'EUR', 'expiry': '202506', 'exchange': 'CME', 'client_id': 12},
    {'symbol': 'GBP', 'expiry': '202506', 'exchange': 'CME', 'client_id': 13},
    {'symbol': 'AUD', 'expiry': '202506', 'exchange': 'CME', 'client_id': 14},
    {'symbol': 'CHF', 'expiry': '202506', 'exchange': 'CME', 'client_id': 15},
    {'symbol': 'PA', 'expiry': '202506', 'exchange': 'NYMEX', 'client_id': 16},
    {'symbol': 'PL', 'expiry': '202507', 'exchange': 'NYMEX', 'client_id': 17},
    {'symbol': 'ZS', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 18},
    {'symbol': 'ZM', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 19},
    {'symbol': 'ZC', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 20},
    {'symbol': 'ZW', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 21},
]

def configurar_contrato(symbol):
    # Diccionario completo con configuración por contrato
    config = {
        'ES': {'umbral': 2.5, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago'},
        'NQ': {'umbral': 10.0, 'margen': 40000, 'contratos_orden': 1, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago'},
        'YM': {'umbral': 20.0, 'margen': 13000, 'contratos_orden': 3, 'variacion_minima': 1.0, 'R': 3, 'num_decimales': None, 'zona_horaria': 'America/Chicago'},
        'RTY': {'umbral': 1.0, 'margen': 10000, 'contratos_orden': 4, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/Chicago'},
    }
```

```

        'JPY': {'umbral': 1.0, 'margen': 4000, 'contratos_orden': 10, 'variacion_minima': 0.000001, 'R': 3, 'num_decimales': 6, 'zona_horaria': 'America/Chicago'},
        'EUR': {'umbral': 0.0002, 'margen': 5000, 'contratos_orden': 8, 'variacion_minima': 0.00005, 'R': 3, 'num_decimales': 5, 'zona_horaria': 'America/Chicago'},
        'GBP': {'umbral': 0.0002, 'margen': 2500, 'contratos_orden': 16, 'variacion_minima': 0.0001, 'R': 3, 'num_decimales': 4, 'zona_horaria': 'America/Chicago'},
        'AUD': {'umbral': 0.0002, 'margen': 2500, 'contratos_orden': 16, 'variacion_minima': 0.00005, 'R': 3, 'num_decimales': 5, 'zona_horaria': 'America/Chicago'},
        'CHF': {'umbral': 0.0002, 'margen': 5000, 'contratos_orden': 8, 'variacion_minima': 0.00005, 'R': 3, 'num_decimales': 5, 'zona_horaria': 'America/Chicago'},
        'CL': {'umbral': 0.03, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.01, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/New_York'},
        'NG': {'umbral': 0.002, 'margen': 11000, 'contratos_orden': 3, 'variacion_minima': 0.001, 'R': 3, 'num_decimales': 3, 'zona_horaria': 'America/New_York'},
        'HO': {'umbral': 0.0002, 'margen': 26000, 'contratos_orden': 1, 'variacion_minima': 0.0001, 'R': 3, 'num_decimales': 4, 'zona_horaria': 'America/New_York'},
        'GC': {'umbral': 15.0, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/New_York'},
        'PA': {'umbral': 5.0, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.5, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/New_York'},
        'PL': {'umbral': 5.0, 'margen': 6000, 'contratos_orden': 6, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/New_York'},
        'HG': {'umbral': 0.02, 'margen': 13000, 'contratos_orden': 3, 'variacion_minima': 0.0005, 'R': 3, 'num_decimales': 4, 'zona_horaria': 'America/New_York'},
        'ZS': {'umbral': 5.0, 'margen': 3500, 'contratos_orden': 10, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago'},
        'ZM': {'umbral': 1.0, 'margen': 3200, 'contratos_orden': 11, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/Chicago'},
        'ZC': {'umbral': 2.0, 'margen': 2200, 'contratos_orden': 18, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago'},
        'ZW': {'umbral': 2.0, 'margen': 3300, 'contratos_orden': 11, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago'},
        'IBEX35': {'umbral': 30.0, 'margen': 13000, 'contratos_orden': 3, 'variacion_minima': 1.0, 'R': 3, 'num_decimales': None, 'zona_horaria': 'Europe/Madrid'},
    }

    # Obtenemos la configuración o valores por defecto si el símbolo no existe
    contrato = config.get(symbol, {
        'umbral': 1.0,
        'margen': 10000,
        'contratos_orden': 1,
        'variacion_minima': 0.01, # Valor por defecto para variación mínima
        'R': 3,
        'num_decimales': 2, # Valor por defecto para número de decimales
        'zona_horaria': 'America/New_York' # Valor por defecto para zona horaria
    })

```

```

# Devolvemos todos los parámetros necesarios
return (
    contrato['umbral'],
    contrato['contratos_orden'],
    contrato['variacion_minima'],
    contrato['R'],
    contrato['num_decimales'],
    contrato['zona_horaria']
)

def recuperar_trade_por_id(ib, order_id: int) -> Trade:

    for trade in ib.trades():
        if trade.order.permId == order_id:
            return trade

    return None

def main():

    max_reintentos = 20 # Máximo de intentos de reconexión
    reintentos = 0
    primeraConexion = True
    ib = None

    while reintentos < max_reintentos: # Bucle externo para controlar reintentos
        try:
            if ib is None or not ib.isConnected():
                ib = IB()
                ib.connect('127.0.0.1', 7497, clientId=1)
                print("Conexión IB establecida")
                reintentos = 0

            if primeraConexion:
                # Prepara todos los contratos
                contratos_ib = [Future(c['symbol'], c['expiry'], c['exchange']) for c in
CONTRATOS]
                vars_contratos = {c['symbol']: variables_por_contrato.obtener_variables(c['symbol']) for c in CONTRATOS}

            for config, contract in zip(CONTRATOS, contratos_ib):
                symbol = config['symbol']
                umbral, cantidad, variacion_minima, r, num_decimales, zona_horaria =
configurar_contrato(config['symbol'])
                vars_contratos[symbol].update({'umbral': umbral, 'quantity': cantidad,
'variacion_minima': variacion_minima, 'R': r, 'num_decimales': num_decimales,
'zona_horaria': zona_horaria})
                file_path = f"{symbol}.min"

```

```

        primeraConexion = False

    else:
        for config, contrato in zip(contratos, contratos_ib):
            symbol = config['symbol']
            vars_contrato = vars_contratos[symbol]

            # RECONSTRUCCIÓN POST-RECONEXIÓN

            print("Entrando en reconstrucción post-reconexión") # Debug
            if vars_contrato['orden_activa']['entry_order'].order.permId:
                vars_contrato['orden_activa']['entry_order'] =
                recuperar_trade_por_id(ib, vars_contrato['orden_activa']['entry_order'].order.permId)

            if vars_contrato['orden_activa']['tp_order'].order.permId:
                vars_contrato['orden_activa']['tp_order'] =
                recuperar_trade_por_id(ib, vars_contrato['orden_activa']['tp_order'].order.permId)

            if vars_contrato['orden_activa']['sl_order'].order.permId:
                vars_contrato['orden_activa']['sl_order'] =
                recuperar_trade_por_id(ib, vars_contrato['orden_activa']['sl_order'].order.permId)

    while True:
        for config, contrato in zip(contratos, contratos_ib):
            # Verificar conexión antes de cada operación
            if not ib.isConnected():
                raise ConnectionError("Conexión perdida con IB")

        try:
            # Procesamiento por contrato
            symbol = config['symbol']
            file_path = f"{symbol}.min"

            # 1. Actualizar datos
            actualizar_archivo_min(ib, file_path, contrato,
vars_contratos[symbol]['zona_horaria'])

            caldayi, _, hii, loi, cli, opi, voli = leer_min(file_path,
vars_contratos[symbol]['zona_horaria'])
            caldayi = np.array(caldayi)

```

```

        hii = np.array(hii)
        loi = np.array(loi)
        cli = np.array(cli)
        opi = np.array(opi)
        voli = np.array(voli)
        hi_agrupacion, lo_agrupacion, _, _, _, _ =
dividir_min_en_grupos_para_swings(caldayi, hii, loi, cli, opi, voli, 3, 1, len(caldayi)-1)

        # 2. Calcular en swings
        hi, lo, cl, op, dhi, dlo =
dividir_min_en_grupos_para_swings(caldayi, hii, loi, cli, opi, voli, 2, 80, len(caldayi)-1)
        swings, tiempos, _ = swing1(hi, lo, cl, op, dhi, dlo)

        # 3. Ejecutar estrategia
        procesar_doble_top_bottom(
            swings, tiempos, vars_contratos[symbol]['umbral'], ib,
contract,
            vars_contratos[symbol]["orden_activa"], hi_agrupacion[0],
lo_agrupacion[0],
            config['client_id'], vars_contratos[symbol]['quantity'],
vars_contratos[symbol]['variacion_minima'], vars_contratos[symbol]['R'],
vars_contratos[symbol]['num_decimales'], vars_contratos[symbol]['zona_horaria'],
        )

    except ConnectionError as e:
        print(f"\n {config['symbol']}: Error de conexión - {str(e)}")
        if ib.isConnected():
            ib.disconnect()
            time.sleep(10)
        break # Sale del bucle interno para reintentar conexión

    except Exception as e:
        print(f"\n Error en {symbol}: {str(e)}")

    # Pausa principal del bucle (con ib.sleep())
#ib.sleep(50) # Mantiene la conexión activa

except KeyboardInterrupt:
    print("\n Interrupción por usuario")
    break

except Exception as e:
    reintentos += 1
    print(f"\n {config['symbol']}: Error crítico (intento
{reintentos}/{max_reintentos}) - {str(e)}")

```

```
time.sleep(10)

# --- Limpieza Final ---
if ib and ib.isConnected():
    ib.disconnect()
print("▣ Programa terminado")

if __name__ == '__main__':
    print("""
=====
▣ BOT MULTICONTRATO (ÚNICA INSTANCIA IB)
=====
""")
    main()
```

## ANEXO G: MAIN\_MULTIPROCESSING.PY

```

import time
from multiprocessing import Process
from ib_insync import *
import variables_por_contrato
from procesado import *
from estrategia_limit import procesar_doble_top_bottom

CONTRATOS = [
    {'symbol': 'ES', 'expiry': '202506', 'exchange': 'CME', 'client_id': 1},
    {'symbol': 'NQ', 'expiry': '202506', 'exchange': 'CME', 'client_id': 2},
    {'symbol': 'CL', 'expiry': '202507', 'exchange': 'NYMEX', 'client_id': 3},
    {'symbol': 'GC', 'expiry': '202506', 'exchange': 'COMEX', 'client_id': 4},
    {'symbol': 'IBEX35', 'expiry': '202506', 'exchange': 'MEFFRV', 'client_id': 5},
    {'symbol': 'HG', 'expiry': '202506', 'exchange': 'COMEX', 'client_id': 6},
    {'symbol': 'NG', 'expiry': '202506', 'exchange': 'NYMEX', 'client_id': 7},
    {'symbol': 'HO', 'expiry': '202506', 'exchange': 'NYMEX', 'client_id': 8},
    {'symbol': 'YM', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 9},
    {'symbol': 'RTY', 'expiry': '202506', 'exchange': 'CME', 'client_id': 10},
    {'symbol': 'JPY', 'expiry': '202506', 'exchange': 'CME', 'client_id': 11},
    {'symbol': 'EUR', 'expiry': '202506', 'exchange': 'CME', 'client_id': 12},
    {'symbol': 'GBP', 'expiry': '202506', 'exchange': 'CME', 'client_id': 13},
    {'symbol': 'AUD', 'expiry': '202506', 'exchange': 'CME', 'client_id': 14},
    {'symbol': 'CHF', 'expiry': '202506', 'exchange': 'CME', 'client_id': 15},
    {'symbol': 'PA', 'expiry': '202506', 'exchange': 'NYMEX', 'client_id': 16},
    {'symbol': 'PL', 'expiry': '202507', 'exchange': 'NYMEX', 'client_id': 17},
    {'symbol': 'ZS', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 18},
    {'symbol': 'ZM', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 19},
    {'symbol': 'ZC', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 20},
    {'symbol': 'ZW', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 21},
]

def configurar_contrato(symbol):
    # Diccionario completo con configuración por contrato
    config = {
        'ES': {'umbral': 2.5, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago'},
        'NQ': {'umbral': 10.0, 'margen': 40000, 'contratos_orden': 1, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago'},
        'YM': {'umbral': 20.0, 'margen': 13000, 'contratos_orden': 3, 'variacion_minima': 1.0, 'R': 3, 'num_decimales': None, 'zona_horaria': 'America/Chicago'},
        'RTY': {'umbral': 1.0, 'margen': 10000, 'contratos_orden': 4, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/Chicago'},
        'JPY': {'umbral': 1.0, 'margen': 4000, 'contratos_orden': 10, 'variacion_minima': 0.000001, 'R': 3, 'num_decimales': 6, 'zona_horaria': 'America/Chicago'}
    }

```

```

    'EUR': {'umbral': 0.0002, 'margen': 5000, 'contratos_orden': 8, 'variacion_minima': 0.00005, 'R': 3, 'num_decimales': 5, 'zona_horaria': 'America/Chicago'},
    'GBP': {'umbral': 0.0002, 'margen': 2500, 'contratos_orden': 16, 'variacion_minima': 0.0001, 'R': 3, 'num_decimales': 4, 'zona_horaria': 'America/Chicago'},
    'AUD': {'umbral': 0.0002, 'margen': 2500, 'contratos_orden': 16, 'variacion_minima': 0.00005, 'R': 3, 'num_decimales': 5, 'zona_horaria': 'America/Chicago'},
    'CHF': {'umbral': 0.0002, 'margen': 5000, 'contratos_orden': 8, 'variacion_minima': 0.00005, 'R': 3, 'num_decimales': 5, 'zona_horaria': 'America/Chicago'},
    'CL': {'umbral': 0.03, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.01, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/New_York'},
    'NG': {'umbral': 0.002, 'margen': 11000, 'contratos_orden': 3, 'variacion_minima': 0.001, 'R': 3, 'num_decimales': 3, 'zona_horaria': 'America/New_York'},
    'HO': {'umbral': 0.0002, 'margen': 26000, 'contratos_orden': 1, 'variacion_minima': 0.0001, 'R': 3, 'num_decimales': 4, 'zona_horaria': 'America/New_York'},
    'GC': {'umbral': 15.0, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/New_York'},
    'PA': {'umbral': 5.0, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.5, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/New_York'},
    'PL': {'umbral': 5.0, 'margen': 6000, 'contratos_orden': 6, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/New_York'},
    'HG': {'umbral': 0.02, 'margen': 13000, 'contratos_orden': 3, 'variacion_minima': 0.0005, 'R': 3, 'num_decimales': 4, 'zona_horaria': 'America/New_York'},
    'ZS': {'umbral': 5.0, 'margen': 3500, 'contratos_orden': 10, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago'},
    'ZM': {'umbral': 1.0, 'margen': 3200, 'contratos_orden': 11, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/Chicago'},
    'ZC': {'umbral': 2.0, 'margen': 2200, 'contratos_orden': 18, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago'},
    'ZW': {'umbral': 2.0, 'margen': 3300, 'contratos_orden': 11, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago'},
    'IBEX35': {'umbral': 30.0, 'margen': 13000, 'contratos_orden': 3, 'variacion_minima': 1.0, 'R': 3, 'num_decimales': None, 'zona_horaria': 'Europe/Madrid'},
    #'SI': {'umbral': 0.5, 'margen': 25000, 'contratos_orden': 1, 'variacion_minima': 0.005} # Agregado Silver
}

# Obtenemos la configuración o valores por defecto si el símbolo no existe
contrato = config.get(symbol, {
    'umbral': 1.0,
    'margen': 10000,
    'contratos_orden': 1,
    'variacion_minima': 0.01, # Valor por defecto para variación mínima
    'R': 3,
    'num_decimales': 2, # Valor por defecto para número de decimales
    'zona_horaria': 'America/New_York' # Valor por defecto para zona horaria
})

```

```

# Devolvemos todos los parámetros necesarios
return (
    contrato['umbral'],
    contrato['contratos_orden'],
    contrato['variacion_minima'],
    contrato['R'],
    contrato['num_decimales'],
    contrato['zona_horaria']
)

def recuperar_trade_por_id(ib, order_id: int) -> Trade:

    for trade in ib.trades():
        if trade.order.permId == order_id:
            return trade

    return None

def ejecutar_contrato(config):
    max_reintentos = 20 # Máximo de intentos de reconexión
    reintentos = 0
    primeraConexion = True
    ib = None

    while reintentos < max_reintentos: # Bucle externo para controlar reintentos
        try:
            # --- Conexión Inicial ---

            if ib is None or not ib.isConnected():
                print(f"⚡{config['symbol']} Conectando a IB...")
                ib = IB()
                ib.connect('127.0.0.1', 7497, clientId=config['client_id'])
                print(f"🔗{config['symbol']} Conectado")
                reintentos = 0

            # --- Configuración del contrato (siempre se ejecuta) ---
            contract = Future(config['symbol'], config['expiry'], config['exchange'])
            file_path = f"{config['symbol']}.min"

            # --- Gestión del estado ---
            if primeraConexion:
                # Configuración inicial para primera conexión
                vars_contrato = variables_por_contrato.obtener_variables(config['symbol'])
                umbral, cantidad, variacion_minima, r, num_decimales, zona_horaria =
                configurar_contrato(config['symbol'])


```

```

vars_contrato.update({
    'umbral': umbral,
    'cantidad': cantidad,
    'variacion_minima': variacion_minima,
    'R': r,
    'num_decimales': num_decimales,
    'zona_horaria': zona_horaria
})
primeraConexion = False
else:
    # RECONSTRUCCIÓN POST-RECONEXIÓN
    print("Entrando en reconstrucción post-reconexión") # Debug
    if vars_contrato['orden_activa']['entry_order'] and
hasattr(vars_contrato['orden_activa']['entry_order'].order, 'permId'):
        vars_contrato['orden_activa']['entry_order'] =
recuperar_trade_por_id(ib, vars_contrato['orden_activa']['entry_order'].order.permId)

    if vars_contrato['orden_activa']['tp_order'] and
hasattr(vars_contrato['orden_activa']['tp_order'].order, 'permId'):
        vars_contrato['orden_activa']['tp_order'] = recuperar_trade_por_id(ib,
vars_contrato['orden_activa']['tp_order'].order.permId)

    if vars_contrato['orden_activa']['sl_order'] and
hasattr(vars_contrato['orden_activa']['sl_order'].order, 'permId'):
        vars_contrato['orden_activa']['sl_order'] = recuperar_trade_por_id(ib,
vars_contrato['orden_activa']['sl_order'].order.permId)
    # --- Bucle principal de trading ---
    while True:
        try:
            # Verificar conexión antes de cada operación
            if not ib.isConnected():
                raise ConnectionError("Conexión perdida con IB")

            # 1. Actualizar datos
            actualizar_archivo_min(ib, file_path, contract,
vars_contrato['zona_horaria'])

            caldayi, _, hii, loi, cli, opi, voli = leer_min(file_path,
vars_contrato['zona_horaria']))
            caldayi = np.array(caldayi)
            hii = np.array(hii)
            loi = np.array(loi)
            cli = np.array(cli)
            opi = np.array(opi)
            voli = np.array(voli)
            hi_agrupacion, lo_agrupacion, _, _, _, _ =
dividir_min_en_grupos_para_swings(caldayi, hii, loi, cli, opi, voli, 3, 1, len(caldayi)-1)

```

```
# 2. Calcular en swings
hi, lo, cl, op, dhi, dlo = dividir_min_en_grupos_para_swings(caldayi,
hi, loi, cli, opi, voli, 2, 80, len(caldayi)-1)
swings, tiempos, _ = swing1(hi, lo, cl, op, dhi, dlo)

# 3. Ejecutar estrategia
procesar_doble_top_bottom(
    swings, tiempos, vars_contrato['umbral'], ib, contract,
    vars_contrato["orden_activa"], hi_agrupacion[0], lo_agrupacion[0],
    config['client_id'], vars_contrato['quantity'],
vars_contrato['variacion_minima'], vars_contrato['R'], vars_contrato['num_decimales'],
vars_contrato['zona_horaria'],
)

ib.sleep(50)

except ConnectionError as e:
    print(f"⚠ {config['symbol']}: Error de conexión - {str(e)}")
    if ib.isConnected():
        ib.disconnect()
    time.sleep(10)
    break # Sale del bucle interno para reintentar conexión

except Exception as e:
    print(f"⚠ {config['symbol']}: Error inesperado - {str(e)}")
    time.sleep(10)

except KeyboardInterrupt:
    break # Permite salir con Ctrl+C

except Exception as e:
    reintentos += 1
    print(f"⚠ {config['symbol']}: Error crítico (intento
{reintentos}/{max_reintentos}) - {str(e)}")
    time.sleep(10)

# --- Limpieza final ---
if ib and ib.isConnected():
    ib.disconnect()
print(f"⚠ {config['symbol']}: Desconectado")

if __name__ == '__main__':
    print("""
=====
    ⚡ Iniciando Bot Multi-Contrato
```

```
=====
""")

procesos = []
for config in CONTRATOS:
    p = Process(target=ejecutar_contrato, args=(config,))
    p.start()
    procesos.append(p)

try:
    for p in procesos:
        p.join() # Espera a que todos los procesos terminen
except KeyboardInterrupt:
    print("\nRecibida señal de interrupción. Deteniendo procesos...")
    for p in procesos:
        p.terminate() # Detención limpia
finally:
    print("Todos los procesos terminados")
```

## ANEXO H: MAIN\_VISUALCHART\_SECUENCIAL.PY

```
import time
from ib_insync import *
import variables_por_contrato
from procesado import *
from estrategia_limit import procesar_doble_top_bottom

CONTRATOS = [
    {'symbol': 'ES', 'expiry': '202506', 'exchange': 'CME', 'client_id': 1},
    {'symbol': 'NQ', 'expiry': '202506', 'exchange': 'CME', 'client_id': 2},
    {'symbol': 'CL', 'expiry': '202507', 'exchange': 'NYMEX', 'client_id': 3},
    {'symbol': 'GC', 'expiry': '202506', 'exchange': 'COMEX', 'client_id': 4},
    {'symbol': 'IBEX35', 'expiry': '202506', 'exchange': 'MEFFRV', 'client_id': 5},
    {'symbol': 'HG', 'expiry': '202506', 'exchange': 'COMEX', 'client_id': 6},
    {'symbol': 'NG', 'expiry': '202506', 'exchange': 'NYMEX', 'client_id': 7},
    {'symbol': 'HO', 'expiry': '202506', 'exchange': 'NYMEX', 'client_id': 8},
    {'symbol': 'YM', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 9},
    {'symbol': 'RTY', 'expiry': '202506', 'exchange': 'CME', 'client_id': 10},
    {'symbol': 'JPY', 'expiry': '202506', 'exchange': 'CME', 'client_id': 11},
    {'symbol': 'EUR', 'expiry': '202506', 'exchange': 'CME', 'client_id': 12},
    {'symbol': 'GBP', 'expiry': '202506', 'exchange': 'CME', 'client_id': 13},
    {'symbol': 'AUD', 'expiry': '202506', 'exchange': 'CME', 'client_id': 14},
    {'symbol': 'CHF', 'expiry': '202506', 'exchange': 'CME', 'client_id': 15},
    {'symbol': 'PA', 'expiry': '202506', 'exchange': 'NYMEX', 'client_id': 16},
    {'symbol': 'PL', 'expiry': '202507', 'exchange': 'NYMEX', 'client_id': 17},
    {'symbol': 'ZS', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 18},
    {'symbol': 'ZM', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 19},
    {'symbol': 'ZC', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 20},
```

```

        {'symbol': 'ZW', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 21},
    ]

def configurar_contrato(symbol):
    # Diccionario completo con configuración por contrato
    config = {
        'ES': {'umbral': 2.5, 'margen': 2000, 'contratos_orden': 2, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'ES.min'},
        'NQ': {'umbral': 10.0, 'margen': 4000, 'contratos_orden': 1, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'NQ.min'},
        'YM': {'umbral': 20.0, 'margen': 13000, 'contratos_orden': 3, 'variacion_minima': 1.0, 'R': 3, 'num_decimales': None, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'YM.min'},
        'RTY': {'umbral': 1.0, 'margen': 10000, 'contratos_orden': 4, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'RTY.min'},
        'JPY': {'umbral': 1.0, 'margen': 4000, 'contratos_orden': 10, 'variacion_minima': 0.000001, 'R': 3, 'num_decimales': 6, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'JPY.min'},
        'EUR': {'umbral': 0.0002, 'margen': 5000, 'contratos_orden': 8, 'variacion_minima': 0.00005, 'R': 3, 'num_decimales': 5, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'EUR.min'},
        'GBP': {'umbral': 0.0002, 'margen': 2500, 'contratos_orden': 16, 'variacion_minima': 0.0001, 'R': 3, 'num_decimales': 4, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'GBP.min'},
        'AUD': {'umbral': 0.0002, 'margen': 2500, 'contratos_orden': 16, 'variacion_minima': 0.00005, 'R': 3, 'num_decimales': 5, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'AUD.min'},
        'CHF': {'umbral': 0.0002, 'margen': 5000, 'contratos_orden': 8, 'variacion_minima': 0.00005, 'R': 3, 'num_decimales': 5, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'CHF.min'},
        'CL': {'umbral': 0.03, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.01, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/New_York', 'ruta_archivo': 'CL.min'},
        'NG': {'umbral': 0.002, 'margen': 11000, 'contratos_orden': 3, 'variacion_minima': 0.001, 'R': 3, 'num_decimales': 3, 'zona_horaria': 'America/New_York', 'ruta_archivo': 'NG.min'},
        'HO': {'umbral': 0.0002, 'margen': 26000, 'contratos_orden': 1, 'variacion_minima': 0.0001, 'R': 3, 'num_decimales': 4, 'zona_horaria': 'America/New_York', 'ruta_archivo': 'HO.min'},
        'GC': {'umbral': 15.0, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/New_York', 'ruta_archivo': 'GC.min'},
        'PA': {'umbral': 5.0, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.5, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/New_York', 'ruta_archivo': 'PA.min'},
    }

```

```

        'PL': {'umbral': 5.0, 'margen': 6000, 'contratos_orden': 6, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/New_York', 'ruta_archivo': 'PL.min'},
        'HG': {'umbral': 0.02, 'margen': 13000, 'contratos_orden': 3, 'variacion_minima': 0.0005, 'R': 3, 'num_decimales': 4, 'zona_horaria': 'America/New_York', 'ruta_archivo': 'HG.min'},
        'ZS': {'umbral': 5.0, 'margen': 3500, 'contratos_orden': 10, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'ZS.min'},
        'ZM': {'umbral': 1.0, 'margen': 3200, 'contratos_orden': 11, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'ZM.min'},
        'ZC': {'umbral': 2.0, 'margen': 2200, 'contratos_orden': 18, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'ZC.min'},
        'ZW': {'umbral': 2.0, 'margen': 3300, 'contratos_orden': 11, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'ZW.min'},
        'IBEX35': {'umbral': 30.0, 'margen': 13000, 'contratos_orden': 3, 'variacion_minima': 1.0, 'R': 3, 'num_decimales': None, 'zona_horaria': 'Europe/Madrid', 'ruta_archivo': 'IBEX35.min'},
    }

    # Obtenemos la configuración o valores por defecto si el símbolo no existe
    contrato = config.get(symbol, {
        'umbral': 1.0,
        'margen': 10000,
        'contratos_orden': 1,
        'variacion_minima': 0.01, # Valor por defecto para variación mínima
        'R': 3,
        'num_decimales': 2, # Valor por defecto para número de decimales
        'zona_horaria': 'America/New_York', # Valor por defecto para zona horaria
        'ruta_archivo': f'{symbol}.min' # Ruta por defecto para el archivo
    })

    # Devolvemos todos los parámetros necesarios
    return (
        contrato['umbral'],
        contrato['contratos_orden'],
        contrato['variacion_minima'],
        contrato['R'],
        contrato['num_decimales'],
        contrato['zona_horaria'],
        contrato['ruta_archivo']
    )

def recuperar_trade_por_id(ib, order_id: int) -> Trade:

```

```

for trade in ib.trades():
    if trade.order.permId == order_id:
        return trade

return None

def main():
    max_reintentos = 20 # Máximo de intentos de reconexión
    reintentos = 0
    primeraConexion = True
    ib = None

    while reintentos < max_reintentos: # Bucle externo para controlar reintentos
        try:
            if ib is None or not ib.isConnected():
                ib = IB()
                ib.connect('127.0.0.1', 7497, clientId=1)
                print("Conexión IB establecida")
                reintentos = 0

            if primeraConexion:
                # Prepara todos los contratos
                contratos_ib = [Future(c['symbol'], c['expiry'], c['exchange']) for c in
CONTRATOS]
                vars_contratos = {c['symbol']:
variables_por_contrato.obtener_variables(c['symbol']) for c in CONTRATOS}

                for config, contract in zip(CONTRATOS, contratos_ib):
                    symbol = config['symbol']
                    umbral, cantidad, variacion_minima, r, num_decimales, zona_horaria,
ruta_archivo = configurar_contrato(config['symbol'])
                    vars_contratos[symbol].update({'umbral': umbral, 'quantity': cantidad,
'variacion_minima': variacion_minima, 'R': r, 'num_decimales': num_decimales,
'zona_horaria': zona_horaria, 'ruta_archivo': ruta_archivo})
                    file_path = vars_contratos[symbol]['ruta_archivo']

                primeraConexion = False

            else:
                for config, contrato in zip(CONTRATOS, contratos_ib):
                    symbol = config['symbol']
                    vars_contrato = vars_contratos[symbol]

                    # RECONSTRUCCIÓN POST-RECONEXIÓN

                    print("Entrando en reconstrucción post-reconexión") # Debug

```

```

        if vars_contrato['orden_activa']['entry_order'] and
hasattr(vars_contrato['orden_activa']['entry_order'].order, 'permId'):
            vars_contrato['orden_activa']['entry_order'] =
recuperar_trade_por_id(ib, vars_contrato['orden_activa']['entry_order'].order.permId)

        if vars_contrato['orden_activa']['tp_order'] and
hasattr(vars_contrato['orden_activa']['tp_order'].order, 'permId'):
            vars_contrato['orden_activa']['tp_order'] =
recuperar_trade_por_id(ib, vars_contrato['orden_activa']['tp_order'].order.permId)

        if vars_contrato['orden_activa']['sl_order'] and
hasattr(vars_contrato['orden_activa']['sl_order'].order, 'permId'):
            vars_contrato['orden_activa']['sl_order'] =
recuperar_trade_por_id(ib, vars_contrato['orden_activa']['sl_order'].order.permId)

while True:
    for config, contract in zip(CONTRATOS, contratos_ib):
        # Verificar conexión antes de cada operación
        if not ib.isConnected():
            raise ConnectionError("Conexión perdida con IB")

    try:
        # Procesamiento por contrato
        symbol = config['symbol']
        file_path = vars_contratos[symbol]['ruta_archivo']

        caldayi, _, hii, loi, cli, opi, voli = leer_min(file_path,
vars_contratos[symbol]['zona_horaria'])
        caldayi = np.array(caldayi)
        hii = np.array(hii)
        loi = np.array(loi)
        cli = np.array(cli)
        opi = np.array(opi)
        voli = np.array(voli)
        hi_agrupacion, lo_agrupacion, _, _, _, _ =
dividir_min_en_grupos_para_swings(caldayi, hii, loi, cli, opi, voli, 3, 1, len(caldayi)-1)

        # 2. Calcular en swings
        hi, lo, cl, op, dhi, dlo =
dividir_min_en_grupos_para_swings(caldayi, hii, loi, cli, opi, voli, 2, 80, len(caldayi)-1)
        swings, tiempos, _ = swing1(hi, lo, cl, op, dhi, dlo)

```

```

# 3. Ejecutar estrategia
procesar_doble_top_bottom(
    swings, tiempos, vars_contratos[symbol]['umbral'], ib,
contract,
    vars_contratos[symbol]["orden_activa"], hi_agrupacion[0],
lo_agrupacion[0],
    config['client_id'], vars_contratos[symbol]['quantity'],
vars_contratos[symbol]['variacion_minima'], vars_contratos[symbol]['R'],
vars_contratos[symbol]['num_decimales'], vars_contratos[symbol]['zona_horaria'],
)
)

except ConnectionError as e:
    print(f"\u25aa {config['symbol']}: Error de conexión - {str(e)}")
    if ib.isConnected():
        ib.disconnect()
    time.sleep(10)
    break # Sale del bucle interno para reintentar conexión

except Exception as e:
    print(f"\u25aa Error en {symbol}: {str(e)}")

# Pausa principal del bucle (con ib.sleep())
#ib.sleep(50) # Mantiene la conexión activa

except KeyboardInterrupt:
    print("\n\u25aa Interrupción por usuario")
    break

except Exception as e:
    reintentos += 1
    print(f"\u25aa {config['symbol']}: Error crítico (intento
{reintentos}/{max_reintentos}) - {str(e)}")
    time.sleep(10)

# --- Limpieza Final ---
if ib and ib.isConnected():
    ib.disconnect()
print(" \u25aa Programa terminado")

if __name__ == '__main__':
    print("""
=====
\u25aa BOT MULTICONTRATO (ÚNICA INSTANCIA IB)
=====
""")
    main()

```

## ANEXO I: MAIN\_VISUALCHART\_MULTIPROCESSING.PY

```

import time
from multiprocessing import Process
from ib_insync import *
import variables_por_contrato
from procesado import *
from estrategia_limit import procesar_doble_top_bottom

CONTRATOS = [
    {'symbol': 'ES', 'expiry': '202506', 'exchange': 'CME', 'client_id': 1},
    {'symbol': 'NQ', 'expiry': '202506', 'exchange': 'CME', 'client_id': 2},
    {'symbol': 'CL', 'expiry': '202507', 'exchange': 'NYMEX', 'client_id': 3},
    {'symbol': 'GC', 'expiry': '202506', 'exchange': 'COMEX', 'client_id': 4},
    {'symbol': 'IBEX35', 'expiry': '202506', 'exchange': 'MEFFRV', 'client_id': 5},
    {'symbol': 'HG', 'expiry': '202506', 'exchange': 'COMEX', 'client_id': 6},
    {'symbol': 'NG', 'expiry': '202506', 'exchange': 'NYMEX', 'client_id': 7},
    {'symbol': 'HO', 'expiry': '202506', 'exchange': 'NYMEX', 'client_id': 8},
    {'symbol': 'YM', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 9},
    {'symbol': 'RTY', 'expiry': '202506', 'exchange': 'CME', 'client_id': 10},
    {'symbol': 'JPY', 'expiry': '202506', 'exchange': 'CME', 'client_id': 11},
    {'symbol': 'EUR', 'expiry': '202506', 'exchange': 'CME', 'client_id': 12},
    {'symbol': 'GBP', 'expiry': '202506', 'exchange': 'CME', 'client_id': 13},
    {'symbol': 'AUD', 'expiry': '202506', 'exchange': 'CME', 'client_id': 14},
    {'symbol': 'CHF', 'expiry': '202506', 'exchange': 'CME', 'client_id': 15},
    {'symbol': 'PA', 'expiry': '202506', 'exchange': 'NYMEX', 'client_id': 16},
    {'symbol': 'PL', 'expiry': '202507', 'exchange': 'NYMEX', 'client_id': 17},
    {'symbol': 'ZS', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 18},
    {'symbol': 'ZM', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 19},
    {'symbol': 'ZC', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 20},
    {'symbol': 'ZW', 'expiry': '202506', 'exchange': 'CBOT', 'client_id': 21},
]

def configurar_contrato(symbol):
    # Diccionario completo con configuración por contrato
    config = {
        'ES': {'umbral': 2.5, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'ES.min'},
        'NQ': {'umbral': 10.0, 'margen': 40000, 'contratos_orden': 1, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'NQ.min'},
        'YM': {'umbral': 20.0, 'margen': 13000, 'contratos_orden': 3, 'variacion_minima': 1.0, 'R': 3, 'num_decimales': None, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'YM.min'}
    }

```

```

    'RTY': {'umbral': 1.0, 'margen': 10000, 'contratos_orden': 4, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'RTY.min'},
        'JPY': {'umbral': 1.0, 'margen': 4000, 'contratos_orden': 10, 'variacion_minima': 0.000001, 'R': 3, 'num_decimales': 6, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'JPY.min'},
            'EUR': {'umbral': 0.0002, 'margen': 5000, 'contratos_orden': 8, 'variacion_minima': 0.00005, 'R': 3, 'num_decimales': 5, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'EUR.min'},
                'GBP': {'umbral': 0.0002, 'margen': 2500, 'contratos_orden': 16, 'variacion_minima': 0.0001, 'R': 3, 'num_decimales': 4, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'GBP.min'},
                    'AUD': {'umbral': 0.0002, 'margen': 2500, 'contratos_orden': 16, 'variacion_minima': 0.00005, 'R': 3, 'num_decimales': 5, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'AUD.min'},
                        'CHF': {'umbral': 0.0002, 'margen': 5000, 'contratos_orden': 8, 'variacion_minima': 0.00005, 'R': 3, 'num_decimales': 5, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'CHF.min'},
                            'CL': {'umbral': 0.03, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.01, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/New_York', 'ruta_archivo': 'CL.min'},
                                'NG': {'umbral': 0.002, 'margen': 11000, 'contratos_orden': 3, 'variacion_minima': 0.001, 'R': 3, 'num_decimales': 3, 'zona_horaria': 'America/New_York', 'ruta_archivo': 'NG.min'},
                                    'HO': {'umbral': 0.0002, 'margen': 26000, 'contratos_orden': 1, 'variacion_minima': 0.0001, 'R': 3, 'num_decimales': 4, 'zona_horaria': 'America/New_York', 'ruta_archivo': 'HO.min'},
                                        'GC': {'umbral': 15.0, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/New_York', 'ruta_archivo': 'GC.min'},
                                            'PA': {'umbral': 5.0, 'margen': 20000, 'contratos_orden': 2, 'variacion_minima': 0.5, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/New_York', 'ruta_archivo': 'PA.min'},
                                                'PL': {'umbral': 5.0, 'margen': 6000, 'contratos_orden': 6, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/New_York', 'ruta_archivo': 'PL.min'},
                                                    'HG': {'umbral': 0.02, 'margen': 13000, 'contratos_orden': 3, 'variacion_minima': 0.0005, 'R': 3, 'num_decimales': 4, 'zona_horaria': 'America/New_York', 'ruta_archivo': 'HG.min'},
                                                        'ZS': {'umbral': 5.0, 'margen': 3500, 'contratos_orden': 10, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'ZS.min'},
                                                            'ZM': {'umbral': 1.0, 'margen': 3200, 'contratos_orden': 11, 'variacion_minima': 0.1, 'R': 3, 'num_decimales': 1, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'ZM.min'},
                                                                'ZC': {'umbral': 2.0, 'margen': 2200, 'contratos_orden': 18, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'ZC.min'},

```

```

        'ZW': {'umbral': 2.0, 'margen': 3300, 'contratos_orden': 11, 'variacion_minima': 0.25, 'R': 3, 'num_decimales': 2, 'zona_horaria': 'America/Chicago', 'ruta_archivo': 'ZW.min'},
        'IBEX35': {'umbral': 30.0, 'margen': 13000, 'contratos_orden': 3, 'variacion_minima': 1.0, 'R': 3, 'num_decimales': None, 'zona_horaria': 'Europe/Madrid', 'ruta_archivo': 'IBEX35.min'},
        #'SI': {'umbral': 0.5, 'margen': 25000, 'contratos_orden': 1, 'variacion_minima': 0.005} # Agregado Silver
    }

    # Obtenemos la configuración o valores por defecto si el símbolo no existe
    contrato = config.get(symbol, {
        'umbral': 1.0,
        'margen': 10000,
        'contratos_orden': 1,
        'variacion_minima': 0.01, # Valor por defecto para variación mínima
        'R': 3,
        'num_decimales': 2, # Valor por defecto para número de decimales
        'zona_horaria': 'America/New_York', # Valor por defecto para zona horaria
        'ruta_archivo': f'{symbol}.min' # Ruta por defecto para el archivo
    })

    # Devolvemos todos los parámetros necesarios
    return (
        contrato['umbral'],
        contrato['contratos_orden'],
        contrato['variacion_minima'],
        contrato['R'],
        contrato['num_decimales'],
        contrato['zona_horaria'],
        contrato['ruta_archivo']
    )
}

def recuperar_trade_por_id(ib, order_id: int) -> Trade:

    for trade in ib.trades():
        if trade.order.permId == order_id:
            return trade

    return None


def ejecutar_contrato(config):
    max_reintentos = 20 # Máximo de intentos de reconexión
    reintentos = 0
    primeraConexion = True
    ib = None

```

```

while reintentos < max_reintentos: # Bucle externo para controlar reintentos
    try:
        # --- Conexión Inicial ---

        if ib is None or not ib.isConnected():
            print(f"⚡{config['symbol']}]: Conectando a IB...")
            ib = IB()
            ib.connect('127.0.0.1', 7497, clientId=config['client_id'])
            print(f"🔗{config['symbol']}]: Conectado")
            reintentos = 0

        # --- Configuración del contrato (siempre se ejecuta) ---
        contract = Future(config['symbol'], config['expiry'], config['exchange'])
        vars_contrato = variables_por_contrato.obtener_variables(config['symbol'])
        file_path = vars_contrato['ruta_archivo']

        # --- Gestión del estado ---
        if primeraConexion:
            # Configuración inicial para primera conexión
            vars_contrato = variables_por_contrato.obtener_variables(config['symbol'])
            umbral, cantidad, variacion_minima, r, num_decimales, zona_horaria,
        ruta_archivo = configurar_contrato(config['symbol'])
        vars_contrato.update({
            'umbral': umbral,
            'cantidad': cantidad,
            'variacion_minima': variacion_minima,
            'R': r,
            'num_decimales': num_decimales,
            'zona_horaria': zona_horaria,
            'ruta_archivo': ruta_archivo,
        })
        primeraConexion = False
    else:
        # RECONSTRUCCIÓN POST-RECONEXIÓN
        print("Entrando en reconstrucción post-reconexión") # Debug
        if vars_contrato['orden_activa']['entry_order'] and
hasattr(vars_contrato['orden_activa']['entry_order'].order, 'permId'):
            vars_contrato['orden_activa']['entry_order'] =
recuperar_trade_por_id(ib, vars_contrato['orden_activa']['entry_order'].order.permId)

            if vars_contrato['orden_activa']['tp_order'] and
hasattr(vars_contrato['orden_activa']['tp_order'].order, 'permId'):
                vars_contrato['orden_activa']['tp_order'] = recuperar_trade_por_id(ib,
vars_contrato['orden_activa']['tp_order'].order.permId)

```

```

        if vars_contrato['orden_activa']['sl_order'] and
hasattr(vars_contrato['orden_activa']['sl_order'].order, 'permId'):
            vars_contrato['orden_activa']['sl_order'] = recuperar_trade_por_id(ib,
vars_contrato['orden_activa']['sl_order'].order.permId)
    # --- Bucle principal de trading ---
    while True:
        try:
            # Verificar conexión antes de cada operación
            if not ib.isConnected():
                raise ConnectionError("Conexión perdida con IB")

            caldayi, _, hii, loi, cli, opi, voli = leer_min(file_path,
vars_contrato['zona_horaria'])
            caldayi = np.array(caldayi)
            hii = np.array(hii)
            loi = np.array(loi)
            cli = np.array(cli)
            opi = np.array(opi)
            voli = np.array(voli)
            hi_agrupacion, lo_agrupacion, _, _, _, _ =
dividir_min_en_grupos_para_swings(caldayi, hii, loi, cli, opi, voli, 3, 1, len(caldayi)-1)

            # 2. Calcular en swings
            hi, lo, cl, op, dhi, dlo = dividir_min_en_grupos_para_swings(caldayi,
hii, loi, cli, opi, voli, 2, 80, len(caldayi)-1)
            swings, tiempos, _ = swing1(hi, lo, cl, op, dhi, dlo)

            # 3. Ejecutar estrategia
            procesar_doble_top_bottom(
                swings, tiempos, vars_contrato['umbral'], ib, contract,
                vars_contrato["orden_activa"], hi_agrupacion[0], lo_agrupacion[0],
                config['client_id'], vars_contrato['quantity'],
                vars_contrato['variacion_minima'], vars_contrato['R'], vars_contrato['num_decimales'],
                vars_contrato['zona_horaria'],
            )

            ib.sleep(50)

        except ConnectionError as e:
            print(f"□ {config['symbol']}: Error de conexión - {str(e)}")
            if ib.isConnected():
                ib.disconnect()
            time.sleep(10)
            break # Sale del bucle interno para reintentar conexión

        except Exception as e:

```

```

        print(f"\u2022 {config['symbol']}: Error inesperado - {str(e)}")
        time.sleep(10)

    except KeyboardInterrupt:
        break # Permite salir con Ctrl+C

    except Exception as e:
        reintentos += 1
        print(f"\u2022 {config['symbol']}: Error cr\u00f3tico (intento
{reintentos}/{max_reintentos}) - {str(e)}")
        time.sleep(10)

    # --- Limpieza final ---
    if ib and ib.isConnected():
        ib.disconnect()
    print(f"\u2022 {config['symbol']}: Desconectado")

if __name__ == '__main__':
    print("""
=====
\u2022 Iniciando Bot Multi-Contrato
=====
""")

    procesos = []
    for config in CONTRATOS:
        p = Process(target=ejecutar_contrato, args=(config,))
        p.start()
        procesos.append(p)

    try:
        for p in procesos:
            p.join() # Espera a que todos los procesos terminen
    except KeyboardInterrupt:
        print("\n\u2022 Recibida se\u00f1al de interrupci\u00f3n. Deteniendo procesos...")
        for p in procesos:
            p.terminate() # Detenci\u00f3n limpia
    finally:
        print(" \u2022 Todos los procesos terminados")

```

## ANEXO J: VARIABLES\_POR\_CONTRATO.PY

```

import numpy as np
from collections import defaultdict
from ib_insync import Trade

variables_por_contrato = defaultdict(lambda: {

```

```
'orden_activa': {
    'enviada': False,
    'entry_order': None,
    'tipo': None,
    'tp_order': None,
    'sl_order': None,
    'take_profit_price': None,
    'stop_loss_price': None,
    'en_espera': False,
},
'quantity': 0,
'umbral': 0.0,
'variacion_minima': 0.0,
'R': 0.0,
'num_decimales': 0,
'zona_horaria': None,
'ruta_archivo': None,
})
}

def obtener_variables(identificador):
    return variables_por_contrato[identificador]
```

## ANEXO K: OBTENER\_CURVAS.PY

```

direccion = -1

indiceswing = 1

for i in range(dias - 1):
    if (hi[i + 1] > hi[i]) and (lo[i + 1] < lo[i]): # outside day
        if dlo[i + 1] < dhi[i + 1]:
            if (direccion == 1 and lo[i + 1] < swings[indiceswing]):
                direccion = -1
                swings.append(lo[i + 1])
                trasw.append(i + 1)
                calsw.append(dlo[i + 1])
                indiceswing += 1
            elif (direccion == -1 and lo[i + 1] < swings[indiceswing]):
                swings[indiceswing] = lo[i + 1]
                trasw[indiceswing] = i + 1
                calsw[indiceswing] = dlo[i + 1]

            direccion = 1
            swings.append(hi[i + 1])
            trasw.append(i + 1)
            calsw.append(dhi[i + 1])
            indiceswing += 1
        else:
            if (direccion == -1 and hi[i + 1] > swings[indiceswing]):
                direccion = 1
                swings.append(hi[i + 1])
                trasw.append(i + 1)
                calsw.append(dhi[i + 1])
                indiceswing += 1
            elif (direccion == 1 and hi[i + 1] > swings[indiceswing]):
                swings[indiceswing] = hi[i + 1]
                trasw[indiceswing] = i + 1
                calsw[indiceswing] = dhi[i + 1]

            direccion = -1
            swings.append(lo[i + 1])
            trasw.append(i + 1)
            calsw.append(dlo[i + 1])
            indiceswing += 1
    elif hi[i + 1] > hi[i]:
        if (direccion == -1 and hi[i + 1] > swings[indiceswing]):
            direccion = 1
            swings.append(hi[i + 1])
            trasw.append(i + 1)
            calsw.append(dhi[i + 1])
            indiceswing += 1
        elif (direccion == 1 and hi[i + 1] > swings[indiceswing]):

```

```

        swings[indiceswing] = hi[i + 1]
        trasw[indiceswing] = i + 1
        calsw[indiceswing] = dhi[i + 1]
    elif lo[i + 1] < lo[i]:
        if (direccion == 1 and lo[i + 1] < swings[indiceswing]):
            direccion = -1
            swings.append(lo[i + 1])
            trasw.append(i + 1)
            calsw.append(dlo[i + 1])
            indiceswing += 1
        elif (direccion == -1 and lo[i + 1] < swings[indiceswing]):
            swings[indiceswing] = lo[i + 1]
            trasw[indiceswing] = i + 1
            calsw[indiceswing] = dlo[i + 1]

    return np.array(swings), np.array(calsw), np.array(trasw)

def leer_min(file_path):
    TAM_REGISTRO = struct.calcsize("ii f f f f ii") # 32 bytes
    caldayi = []
    tradayi = []
    hora = []
    hii = []
    loi = []
    cli = []
    opi = []
    voli = []

    with open(file_path, "rb") as f:
        f.seek(0, 0) # Asegurar que leemos desde el principio real

        while True:
            data = f.read(TAM_REGISTRO)
            if len(data) < TAM_REGISTRO:
                break # Evita errores si el archivo termina antes

            # Desempaquetar datos
            fecha, time, op, hi, lo, cl, vol, _ = struct.unpack("ii f f f f ii", data)

            # Convertir la fecha al formato datetime
            año = fecha // 500
            aux = fecha % 500
            mes = aux // 32
            dia = aux % 32
            hour = time // 3600
            minute = (time % 3600) // 60

```

```

        fecha_datetime = datetime(año, mes, dia, hour, minute, tzinfo=tzinfo_utc)
        caldayi.append(fecha_datetime)
        tradayi.append(len(caldayi))
        opi.append(op)
        hii.append(hi)
        loi.append(lo)
        cli.append(cl)
        voli.append(vol)

    return caldayi, tradayi, hii, loi, cli, opi, voli

def dividir_min_en_grupos_para_swings(caldayi, hii, loi, cli, opi, voli, n_min_barras,
n_grupos, inicio_relativo):

    num_barras = len(caldayi)
    total_min_deseados = n_min_barras * n_grupos

    # Validaciones
    if num_barras < total_min_deseados:
        raise ValueError("No hay suficientes datos para formar los grupos
especificados.")

    if inicio_relativo is None:
        inicio_relativo = num_barras

    if inicio_relativo < total_min_deseados:
        raise ValueError(f"El inicio_relativo ({inicio_relativo}) es insuficiente para
{n_grupos} grupos de {n_min_barras} mins.")

    # Inicializamos arrays para los resultados
    hi_swings = np.empty(n_grupos)
    lo_swings = np.empty(n_grupos)
    op_swings = np.empty(n_grupos)
    cl_swings = np.empty(n_grupos)
    dhi_swings = np.empty(n_grupos, dtype=caldayi.dtype)
    dlo_swings = np.empty(n_grupos, dtype=caldayi.dtype)

    grupos = []
    fin_segmento = inicio_relativo + 1
    inicio_segmento = fin_segmento - total_min_deseados

    for i in range(n_grupos):
        # Calculamos los índices del grupo actual

        grupo_inicio = i * n_min_barras
        grupo_fin = grupo_inicio + n_min_barras
        # Extraemos el grupo

```

```

        hi_grupo = hii[inicio_segmento + grupo_inicio : inicio_segmento + grupo_fin]
        loi_grupo = loi[inicio_segmento + grupo_inicio : inicio_segmento + grupo_fin]

        # Calculamos métricas
        max_hi_idx = np.argmax(hi_grupo)
        min_loi_idx = np.argmin(loi_grupo)

        # Almacenamos resultados
        hi_swings[i] = hi_grupo[max_hi_idx]
        lo_swings[i] = loi_grupo[min_loi_idx]
        op_swings[i] = opi[inicio_segmento + grupo_inicio] # Primer open
        cl_swings[i] = cli[inicio_segmento + grupo_fin - 1] # Último close
        dhi_swings[i] = caldayi[inicio_segmento + grupo_inicio + max_hi_idx]
        dlo_swings[i] = caldayi[inicio_segmento + grupo_inicio + min_loi_idx]

    return hi_swings, lo_swings, cl_swings, op_swings, dhi_swings, dlo_swings

def backtest_doble_top_bottom(swings, swing_times, high_actual, low_actual, umbral,
variacion_minima, R_multiplier, num_decimales, i):
    # Configuración del archivo de log
    log_file = "backtest_traces.log"

    # Función auxiliar para escribir en el log
    def write_log(message):
        with open(log_file, 'a') as f:
            f.write(f"{caldayi[i]}: {message}\n")

    tiempo_max_espera = timedelta(minutes=5)
    RESET_VALUES = {
        'tipo': None,
        'swing_detectado': None,
        'tiempo_deteccion': None,
        'entry_stop_price': 0.0,
        'stop_loss_price': 0.0,
        'take_profit_price': 0.0,
        'en_espera': False,
        'tiempo_orden': 0,
        'entry_stop_pendiente': False,
        'entry_stop_ejecutada': False
    }

    lsw = len(swings)

    if variables_backtest['entry_stop_pendiente'] is True:
        if variables_backtest['tipo'] == 'double_top':
            if loi[i] <= variables_backtest['entry_stop_price']:

```

```

        write_log(f"ENTRADA EJECUTADA - Double Top. Precio:
{variables_backtest['entry_stop_price']}]")
            variables_backtest['entry_stop_pendiente'] = False
            variables_backtest['entry_stop_ejecutada'] = True
            return
        else:
            if variables_backtest['en_espera'] is True:
                if variables_backtest['tiempo_orden'] > 0:
                    variables_backtest['tiempo_orden'] -= 1
                    return
                write_log("RESET - Tiempo de espera agotado (Double Top)")
                variables_backtest.update(RESET_VALUES)
            elif variables_backtest['en_espera'] is False:
                if swings[-1] > swings[-2]: # posible double top
                    for k in range(1, (lsw - 1) // 2 + 1):
                        dif = swings[-1] - swings[(-2 * k)-1]
                        if abs(dif) < umbral:
                            write_log("NO SE EJECUTO LA ORDEN, CANCELANDO")
                            variables_backtest.update(RESET_VALUES)
                            break
                elif dif <= -umbral:
                    write_log("EN ESPERA 5 MIN MÁS")
                    variables_backtest['en_espera'] = True
                    variables_backtest['tiempo_orden'] = 5
                    return
                else:
                    write_log("EN ESPERA 5 MIN MÁS")
                    variables_backtest['en_espera'] = True
                    variables_backtest['tiempo_orden'] = 5
                    return
            else:
                write_log("EN ESPERA 5 MIN MÁS")
                variables_backtest['en_espera'] = True
                variables_backtest['tiempo_orden'] = 5
                return

        elif variables_backtest['tipo'] == 'double_bottom':
            if hii[i] >= variables_backtest['entry_stop_price']:
                write_log(f"ENTRADA EJECUTADA - Double Bottom. Precio:
{variables_backtest['entry_stop_price']}]")
                    variables_backtest['entry_stop_pendiente'] = False
                    variables_backtest['entry_stop_ejecutada'] = True
                    return
            else:
                if variables_backtest['en_espera'] is True:
                    if variables_backtest['tiempo_orden'] > 0:
                        variables_backtest['tiempo_orden'] -= 1

```

```

        return
    write_log("RESET - Tiempo de espera agotado (Double Bottom)")
    variables_backtest.update(RESET_VALUES)
elif variables_backtest['en_espera'] is False:
    if swings[-1] < swings[-2]: # posible double bottom
        for k in range(1, (lsw - 1) // 2 + 1):
            dif = swings[-1] - swings[(-2 * k)-1]
        if abs(dif) < umbral:
            write_log("NO SE EJECUTO LA ORDEN, CANCELANDO")
            variables_backtest.update(RESET_VALUES)
            break
        elif dif >= umbral:
            write_log("EN ESPERA 5 MIN MÁS")
            variables_backtest['en_espera'] = True
            variables_backtest['tiempo_orden'] = 5
            return
    else:
        write_log("EN ESPERA 5 MIN MÁS")
        variables_backtest['en_espera'] = True
        variables_backtest['tiempo_orden'] = 5
        return

if variables_backtest['entry_stop_ejecutada'] is True:
    if variables_backtest['tipo'] == 'double_top':
        if hii[i] >= variables_backtest['stop_loss_price']:
            write_log(f"SALIDA POR STOP LOSS - Double Top. Precio:
{variables_backtest['stop_loss_price']}")

            if len(variables_backtest['curva']) == 0:
                variables_backtest['curva'].append(0.0)
            variables_backtest['curva'].append((variables_backtest['curva'][-1]) + (variables_backtest['entry_stop_price'] - variables_backtest['stop_loss_price']))
            variables_backtest['tiempo_trades'].append(caldayi[i])
            variables_backtest.update(RESET_VALUES)
    else:
        variables_backtest['curva'].append((variables_backtest['curva'][-1]) + (variables_backtest['entry_stop_price'] - variables_backtest['stop_loss_price']))
        variables_backtest['tiempo_trades'].append(caldayi[i])
        variables_backtest.update(RESET_VALUES)

elif loi[i] <= variables_backtest['take_profit_price']:
    write_log(f"SALIDA POR TAKE PROFIT - Double Top. Precio:
{variables_backtest['take_profit_price']}")

    if len(variables_backtest['curva']) == 0:

```

```

        variables_backtest['curva'].append(0.0)
        variables_backtest['curva'].append((variables_backtest['curva'][-1]) + (variables_backtest['entry_stop_price'] - variables_backtest['take_profit_price']))
        variables_backtest['tiempo_trades'].append(caldayi[i])
        variables_backtest.update(RESET_VALUES)
    else:
        variables_backtest['curva'].append((variables_backtest['curva'][-1]) + (variables_backtest['entry_stop_price'] - variables_backtest['take_profit_price']))
        variables_backtest['tiempo_trades'].append(caldayi[i])
        variables_backtest.update(RESET_VALUES)

    else:
        return

    elif variables_backtest['tipo'] == 'double_bottom':
        if loi[i] <= variables_backtest['stop_loss_price']:
            write_log(f"SALIDA POR STOP LOSS - Double Bottom. Precio: {variables_backtest['stop_loss_price']} ")
            if len(variables_backtest['curva']) == 0:
                variables_backtest['curva'].append(0.0)
                variables_backtest['curva'].append((variables_backtest['curva'][-1]) + (variables_backtest['stop_loss_price'] - variables_backtest['entry_stop_price']))
                variables_backtest['tiempo_trades'].append(caldayi[i])
                variables_backtest.update(RESET_VALUES)
            else:
                variables_backtest['curva'].append((variables_backtest['curva'][-1]) + (variables_backtest['stop_loss_price'] - variables_backtest['entry_stop_price']))
                variables_backtest['tiempo_trades'].append(caldayi[i])
                variables_backtest.update(RESET_VALUES)

        elif hii[i] >= variables_backtest['take_profit_price']:
            write_log(f"SALIDA POR TAKE PROFIT - Double Bottom. Precio: {variables_backtest['take_profit_price']} ")
            if len(variables_backtest['curva']) == 0:
                variables_backtest['curva'].append(0.0)
                variables_backtest['curva'].append((variables_backtest['curva'][-1]) + (variables_backtest['take_profit_price'] - variables_backtest['entry_stop_price']))
                variables_backtest['tiempo_trades'].append(caldayi[i])
                variables_backtest.update(RESET_VALUES)
            else:
                variables_backtest['curva'].append((variables_backtest['curva'][-1]) + (variables_backtest['take_profit_price'] - variables_backtest['entry_stop_price']))
                variables_backtest['tiempo_trades'].append(caldayi[i])
                variables_backtest.update(RESET_VALUES)

    else:
        return

```

```

if swings[-1] < swings[-2]: # posible double bottom
    for k in range(1, (lsw - 1) // 2 + 1):
        dif = swings[-1] - swings[(-2 * k)-1]
        if abs(dif) < umbral:
            write_log(f"PATRÓN DETECTADO - Double Bottom. Swing: {swings[-1]}, Tiempo: {swing_times[-1]}")
            variables_backtest['tipo'] = 'double_bottom'
            variables_backtest['swing_detectado'] = swings[-1]
            variables_backtest['tiempo_deteccion'] = swing_times[-1]
            break
        elif dif >= umbral:
            break

    elif swings[-1] > swings[-2]: # posible double top
        for k in range(1, (lsw - 1) // 2 + 1):
            dif = swings[-1] - swings[(-2 * k)-1]
            if abs(dif) < umbral:
                write_log(f"PATRÓN DETECTADO - Double Top. Swing: {swings[-1]}, Tiempo: {swing_times[-1]}")
                variables_backtest['tipo'] = 'double_top'
                variables_backtest['swing_detectado'] = swings[-1]
                variables_backtest['tiempo_deteccion'] = swing_times[-1]
                break
            elif dif <= -umbral:
                break

    # --- NUEVAS VALIDACIONES ---
    if variables_backtest['tipo'] and variables_backtest['swing_detectado'] and variables_backtest['tiempo_deteccion']:
        # 1. Verificar que no hayan pasado más de 5 minutos desde la detección
        tiempo_actual = caldayi[i]
        tiempo_transcurrido = tiempo_actual - variables_backtest['tiempo_deteccion']

        if tiempo_transcurrido > tiempo_max_espera:
            write_log("RESET - Tiempo máximo de espera excedido")
            variables_backtest.update(RESET_VALUES)
            return

        # 2. Verificar que el precio actual está dentro del rango permitido
        precio_actual = high_actual if variables_backtest['tipo'] == 'double_bottom' else low_actual
        distancia_al_swing = abs(precio_actual - variables_backtest['swing_detectado'])

        if distancia_al_swing > umbral:
            write_log(f"RESET - Distancia al swing excedida. Distancia: {distancia_al_swing}, Umbral: {umbral}")
            variables_backtest.update(RESET_VALUES)
            return

```

```

        if variables_backtest['tipo'] == 'double_top':
            variables_backtest['entry_stop_price'] = round(low_actual -
variacion_minima, num_decimales)
            variables_backtest['stop_loss_price'] = round(swings[-1] +
variacion_minima, num_decimales)
            variables_backtest['take_profit_price'] =
round(variables_backtest['entry_stop_price'] - (R_multiplier *
(variables_backtest['stop_loss_price'] - variables_backtest['entry_stop_price'])), num_decimales)
            variables_backtest['entry_stop_pendiente'] = True
            write_log(f"ORDEN CONFIGURADA - Double Top. Entry:
{variables_backtest['entry_stop_price']}, SL: {variables_backtest['stop_loss_price']}, TP:
{variables_backtest['take_profit_price']}")

        elif variables_backtest['tipo'] == 'double_bottom':
            variables_backtest['entry_stop_price'] = round(high_actual +
variacion_minima, num_decimales)
            variables_backtest['stop_loss_price'] = round(swings[-1] -
variacion_minima, num_decimales)
            variables_backtest['take_profit_price'] =
round(variables_backtest['entry_stop_price'] + (R_multiplier *
(variables_backtest['entry_stop_price'] - variables_backtest['stop_loss_price'])), num_decimales)
            variables_backtest['entry_stop_pendiente'] = True
            write_log(f"ORDEN CONFIGURADA - Double Bottom. Entry:
{variables_backtest['entry_stop_price']}, SL: {variables_backtest['stop_loss_price']}, TP:
{variables_backtest['take_profit_price']}")

caldayi, tradayi, hii, loi, cli, opi, voli = leer_min(file_path)

total_registros = len(caldayi)

caldayi = np.array(caldayi)
hii = np.array(hii)
loi = np.array(loi)
cli = np.array(cli)
opi = np.array(opi)
volfi = np.array(voli)

for i in range(57600, total_registros):
    if i % 1000 == 0: # Mostrar progreso cada 1000 registros
        print(f"Procesando {i}/{total_registros} ({(i-57600)/(total_registros-57600)*100:.1f}%)")

```

```

try:

    hi_agrupacion, lo_agrupacion, _, _, _, _ =
dividir_min_en_grupos_para_swings(caldayi, hii, loi, cli, opi, voli, swing_corto, 1, i)

        hi, lo, cl, op, dhi, dlo = dividir_min_en_grupos_para_swings(caldayi, hii, loi,
cli, opi, voli, swing_largo, n_grupos, i)

        swings, tiempos, _ = swing1(hi, lo, cl, op, dhi, dlo)
        backtest_doble_top_bottom(swings, tiempos, hi_agrupacion[0], lo_agrupacion[0],
umbral, variacion_minima, R, numero_decimales, i)

except IndexError:
    print(f"Error: No hay suficientes datos para formar {n_grupos} grupos de
{swing_largo} minutos desde el registro {i}")
    continue
except ValueError as e:
    print(f"Error: {e}")
    continue


df = pd.DataFrame(variables_backtest['curva'])
df_times = pd.DataFrame(variables_backtest['tiempo_trades'])
df_combinado = pd.concat([df_times, df], axis=1)
df_combinado.to_csv(
    'curva_y_tiempos_combinados.csv',
    sep=';',
    decimal=',',
    float_format='%.5f',
    index=False,
    date_format='%Y-%m-%d %H:%M:%S' # Formato de timestamps
)
#df.to_csv('curva_backtest.csv', sep=';', decimal=',', float_format='%.5f',
index=False)
curva("010001ES.min", 800 ,0.002, 2.5, 2, 1, 100, 2)

```

## ANEXO L: VARIABLES\_BACKTEST.PY

```

variables_backtest = {

'tipo': None,
'swing_detectado': None,
'tiempo_deteccion': None,
'entry_stop_price': 0.0,
'stop_loss_price': 0.0,
}

```

```

'take_profit_price': 0.0,
'en_espera': False,
'tiempo_orden': 0,
'entry_stop_pendiente': False,
'entry_stop_ejecutada': False,
'curva': [],
'tiempo_trades': [],
}

```

## ANEXO M: PINTAR\_CURVAS\_DEF.PY

```

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

# 1. Configuración y lectura del archivo
try:
    # Definir nombres para las 28 columnas
    column_names = [f'Curva {i}' for i in range(1, 21)]

    # Leer el archivo CSV
    df = pd.read_csv(
        'curvas_NQ.csv',
        sep=';',
        decimal=',',
        header=None,
        names=column_names
    )

    # Convertir todas las columnas a float (por si acaso)
    for col in column_names:
        df[col] = df[col].astype(float)

    print("¡Archivo leído con éxito!")
    #print(df.head())

except Exception as e:
    print(f"Error al leer el archivo: {e}")
    exit()

# 2. Configuración del gráfico
plt.figure(figsize=(16, 9)) # Tamaño más grande para mejor visualización

# 3. Generar colores distintos para cada columna
colors = plt.cm.viridis(np.linspace(0, 1, len(column_names)))

# 4. Graficar todas las columnas
for i, col in enumerate(column_names):

```

```
plt.plot(df[col], label=col, color=colors[i], linewidth=1, alpha=0.7)

# 5. Personalización del gráfico
plt.title('NQ (E-mini Nasdaq-100 futures)', fontsize=16)
plt.xlabel('Trades', fontsize=14)
plt.ylabel('Puntos', fontsize=14)
plt.axhline(0, color='black', linestyle='--', linewidth=0.5)

# Mejorar la leyenda (puede ser muy grande)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)

plt.grid(True, alpha=0.3)

# 6. Guardar y mostrar
plt.tight_layout() # Ajustar layout para que quepa la leyenda
plt.savefig('grafico_28_columnas.png', dpi=300, bbox_inches='tight')
plt.show()
```