**Documentación de Ejecución de la Aplicación IA Trading**

Esta guía proporciona instrucciones detalladas para instalar, configurar y ejecutar paso a paso la aplicación de trading algorítmico desarrollada en este proyecto. Incluye explicación de cada parámetro, comandos disponibles y ejemplos de uso.

**1. Requisitos Previos**

* **Entorno Python:** Versión >= 3.9
* **Dependencias:** Instalar con:
* pip install -r requirements.txt
* **Datos Históricos:** CSV de datos OHLC con columnas open, high, low, close y fecha en el índice.
* **Archivo de Configuración:** config/config.yaml debe existir y contener rutas y parámetros necesarios.

**2. Carga de Datos**

Antes de cualquier procesamiento, la aplicación debe cargar los datos históricos:

1. **Ubicación**: Coloca los CSV en:
2. data/histdata/<INSTRUMENTO>/<archivo.csv>
3. **Comando de CLI**:
4. >python src/data\_pipeline.py
5. **Salida**: Los datos procesados se guardan en data/processed/<INSTRUMENTO>\_YYYYMMDD\_YYYYMMDD.csv.
6. **Validación**: La aplicación verifica
   * Formato de columnas.
   * No hay fechas duplicadas.
   * Rango solicitado coincide con los datos.

Ejemplo:

python -m src.cli cargar\_datos --instrumento EURUSD --periodo 2021-01-01:2021-06-30

**3. Estructura del Proyecto**

proyectoIA/

├─ src/

│ ├─ cli.py # Interfaz de línea de comandos

│ ├─ data\_pipeline.py # Funciones de carga y procesamiento de datos

│ ├─ optimize.py # Optimización de estrategias (Optuna)

│ ├─ validate\_params.py# Validación de parámetros

│ ├─ train.py # Entrenamiento de modelo o generación de señales

│ ├─ signals.py # Lógica de generación de señales

│ ├─ risk\_tuning.py # Ajuste de riesgo y confianza

│ ├─ grid\_search.py # Búsqueda de parámetros por Sharpe

│ ├─ evaluate.py # Evaluación de rendimiento

│ └─ utils.py # Utilidades generales (carga de datos, métricas)

├─ config/

│ ├─ config.yaml # Parámetros de configuración

│ └─ \_\_init\_\_.py # Paquete Python

├─ data/

│ ├─ histdata/... # Datos de mercado históricos (raw)

│ └─ processed/... # Datos preprocesados por `cargar\_datos`

└─ requirements.txt # Dependencias del proyecto

**4. Configuración del Archivo config/config.yaml**

# Rutas a carpetas o archivos de datos

data\_paths:

raw: data/histdata

processed: data/processed

# Periodicidades a evaluar

timeframes:

- daily

- hourly

# Parámetros específicos de la estrategia (ejemplo)

parameters:

ema\_short: 12

ema\_long: 26

risk\_level: 0.02

# Bot de optimización (Optuna)

optuna:

n\_trials: 50

timeout: 600 # segundos

* **data\_paths.raw:** Carpeta raíz con subcarpetas por instrumento.
* **data\_paths.processed:** Carpeta destino para datos transformados.
* **timeframes:** Lista de granularidades para analizar (daily, hourly, etc.).
* **parameters:** Valores iniciales de indicadores y risk\_level.
* **optuna.n\_trials / timeout:** Número de iteraciones y tiempo máximo para optimización.

**5. Comandos CLI Disponibles y Ejemplos**

La aplicación se maneja mediante cli.py. Sintaxis general:

python -m src.cli <comando> [--opciones]

**5.1 cargar\_datos**

**Descripción:** Carga y valida datos históricos de mercado.

python -m src.cli cargar\_datos --instrumento EURUSD --periodo 2020-01-01:2020-04-01

* --instrumento: Carpeta con CSV.
* --periodo: Rango de fechas.

**5.2 optimize**

**Descripción:** Ejecuta la optimización de parámetros de la estrategia con Optuna.

python -m src.cli optimize

**Opciones:**

* --trials N (opcional): Número de pruebas (sobrepone optuna.n\_trials).
* --timeout S (opcional): Tiempo máximo en segundos.

**5.3 validar\_parametros**

**Descripción:** Verifica que los parámetros generados cumplan restricciones (rangos, lógica interna).

python -m src.cli validar\_parametros

**5.4 entrenar**

**Descripción:** Entrena el modelo o genera métricas de backtest usando parámetros validados.

python -m src.cli entrenar

**5.5 generar\_senales**

**Descripción:** Crea señales de compra/venta en base al modelo entrenado.

python -m src.cli generar\_senales --periodo YYYY-MM-DD:YYYY-MM-DD

* **--periodo:** Rango de fechas p.ej. 2020-01-01:2020-04-01.

**5.6 tune\_risk\_confianza**

**Descripción:** Ajusta el nivel de riesgo (position sizing) y umbral de confianza.

python -m src.cli tune\_risk\_confianza

**5.7 grid\_search\_sharpe**

**Descripción:** Realiza búsqueda en cuadrícula de parámetros para maximizar el ratio de Sharpe.

python -m src.cli grid\_search\_sharpe

**5.8 evaluar\_rentabilidad**

**Descripción:** Genera un informe de rentabilidad y métricas finales.

python -m src.cli evaluar\_rentabilidad

**6. Flujo Completo de Ejecución**

python -m src.cli cargar\_datos --instrumento EURUSD --periodo 2021-01-01:2021-06-30 \

&& python -m src.cli optimize \

&& python -m src.cli validar\_parametros \

&& python -m src.cli entrenar \

&& python -m src.cli generar\_senales --periodo 2021-01-01:2021-06-30 \

&& python -m src.cli tune\_risk\_confianza \

&& python -m src.cli grid\_search\_sharpe \

&& python -m src.cli evaluar\_rentabilidad

Cada comando mostrará en consola el progreso y guardará resultados en la carpeta outputs/.

**7. Consejos y Buenas Prácticas**

* **Entornos Virtuales:** Use venv o conda para aislar dependencias.
* **Backtesting:** Compruebe visualmente los resultados generados en outputs/plots/.
* **Logging:** Revise logs/ para diagnósticos detallados.
* **Extensibilidad:** Añada nuevos indicadores ajustando parameters en config.yaml.

Con esta documentación dispones de una guía completa para poner en marcha, calibrar, evaluar y depurar tu sistema de trading algorítmico. ¡Éxito con tus pruebas y mejoras!