

Python vs MQL: no compiten, se complementan

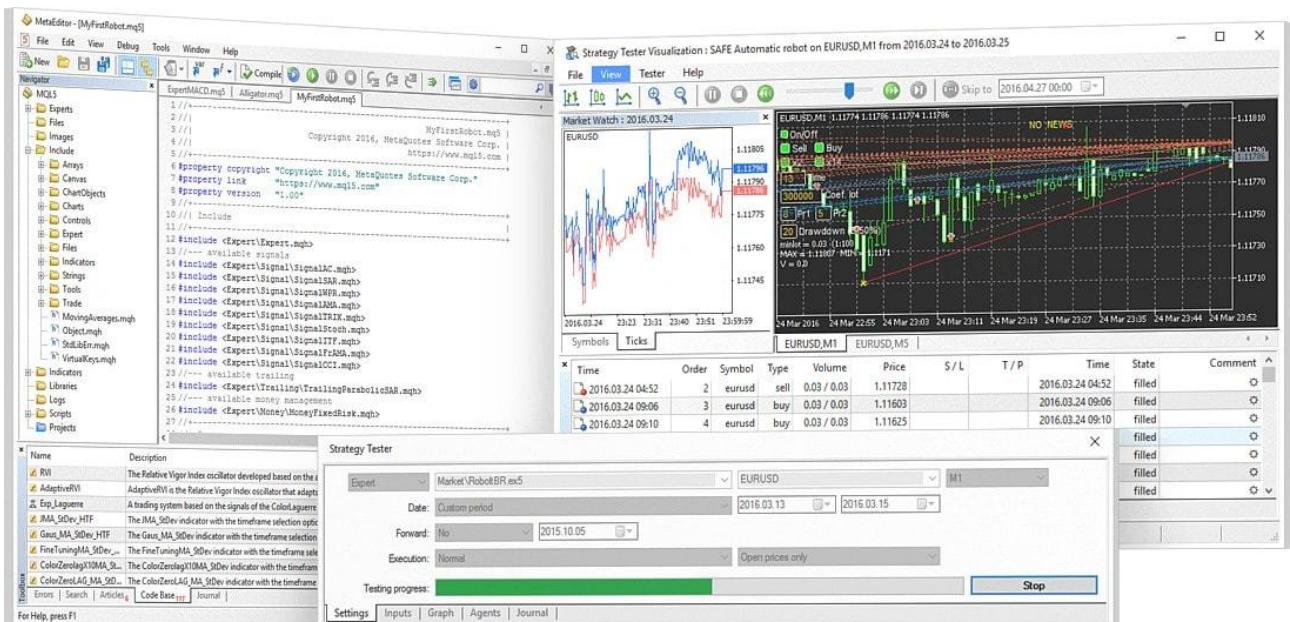
◊ ¿Dónde queda MQL?

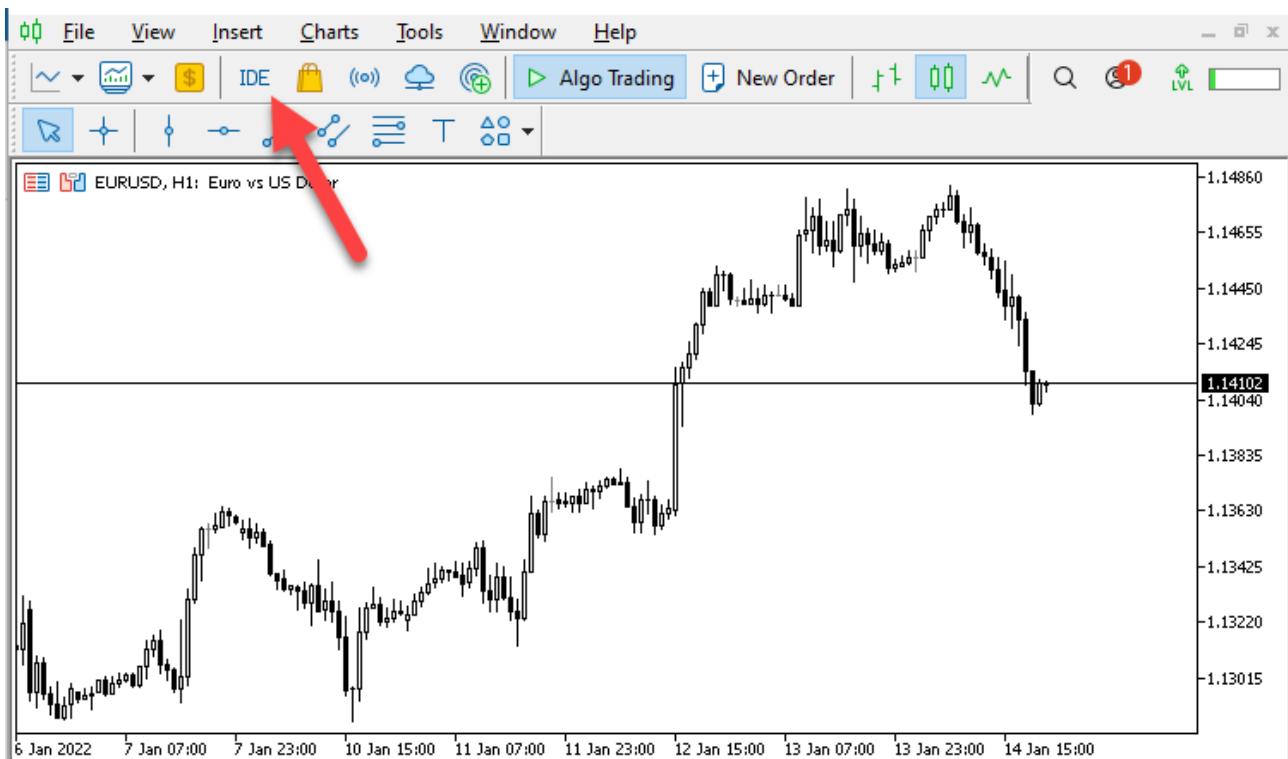
MQL (MetaQuotes Language) es el **lenguaje nativo** de **MetaTrader 4 / MetaTrader 5**.

👉 Su territorio natural es:

- **Forex**
- **CFDs**
- Trading **retail**
- Ejecución directa en broker

```
1 //+-----+
2 //|                                     Moving Average.mq4 |
3 //|                                     Copyright 2005-2014, MetaQuotes Software Corp. |
4 //|                                         http://www.mql4.com |
5 //+-----+
6 #property copyright "2005-2014, MetaQuotes Software Corp."
7 #property link      "http://www.mql4.com"
8 #property description "Moving Average sample expert advisor"
9
10 #include <mq4.mqh>
11
12 #define MAGICMA 20131111
13 //--- Inputs
14 input double Lots          =0.1;
15 input double MaximumRisk   =0.02;
16 input double DecreaseFactor=3;
17 input int   MovingPeriod   =12;
18 input int   MovingShift    =6;
19
20
21 int MAHandle = INVALID_HANDLE;
22
23
24 int OnInit()
25 {
26
27     return (INIT_SUCCEEDED);
28 }
```





💡 ¿Para qué es EXCELENTE MQL?

MQL brilla cuando necesitas:

- Ejecución en tiempo real**
- Expert Advisors (EAs)** rápidos y estables
- Acceso directo al bróker**
- Backtesting integrado** (Strategy Tester)
- Baja latencia** (mejor que Python puro)
-  Para un trader de Forex **MQL es una joya**, no un “lenguaje menor”.



¿Entonces por qué Python es “el rey”?

Porque domina **todo lo que rodea** al trading algorítmico:

Área	Python	MQL
Exploración de datos	● ● ●	●
Machine Learning	● ● ●	●
Backtesting avanzado	● ●	●
Visualización	● ● ●	●
Ejecución en broker	●	● ●
Producción institucional	● ● ●	●

Python gana por ecosistema, no por velocidad de ejecución.



La arquitectura REAL (la que usan los que saben)

Los traders serios hacen esto:

Python 🧠

- análisis
- feature engineering
- optimización
- ML / estadística

⬇ señales

MQL ⚡

- ejecución
- gestión de órdenes
- stops / trailing
- conexión broker

 Python piensa, MQL ejecuta.

Errores comunes sobre MQL

-  “MQL está obsoleto” → **Falso**
-  “Python lo reemplazó” → **Falso**
-  “MQL no es serio” → **Falso**

Lo que sí es cierto:

- **MQL no es para data science**
 - **MQL no es multiplataforma**
 - **MQL vive dentro de MetaTrader**
-

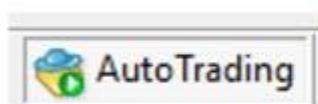
¿Cuál deberías usar TÚ?

Depende de tu objetivo:

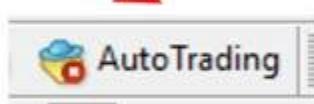
- **Solo Forex / MT4–MT5 → MQL**
 - **Estrategias cuantitativas / ML → Python**
 - **Sistema profesional → Python + MQL**
 - **Alta frecuencia real → C++ (otra liga **)
-

1 ¿Qué significan el globo en MetaTrader?

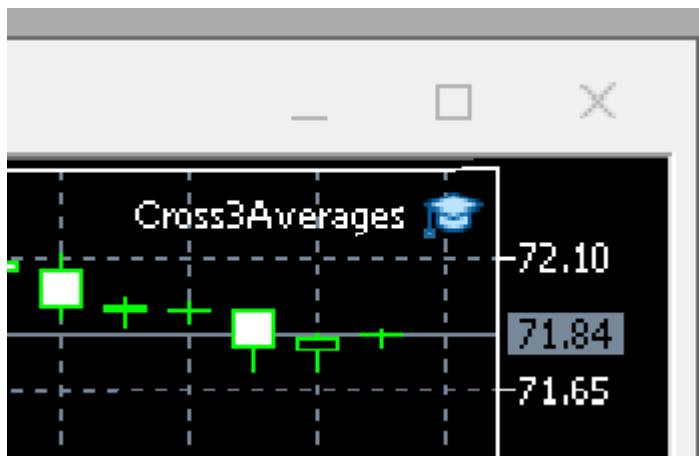
En **MetaTrader 4 / MetaTrader 5**, ese ícono **indica el estado del trading algorítmico** (Expert Advisors).



**Activates/Allows
automatic trading
in MT4 account**



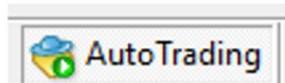
**Deactivates/denies
the automatic tra-
ding in MT4 account**



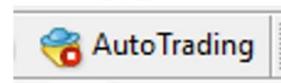
Control autotrading en MetaTrader 4



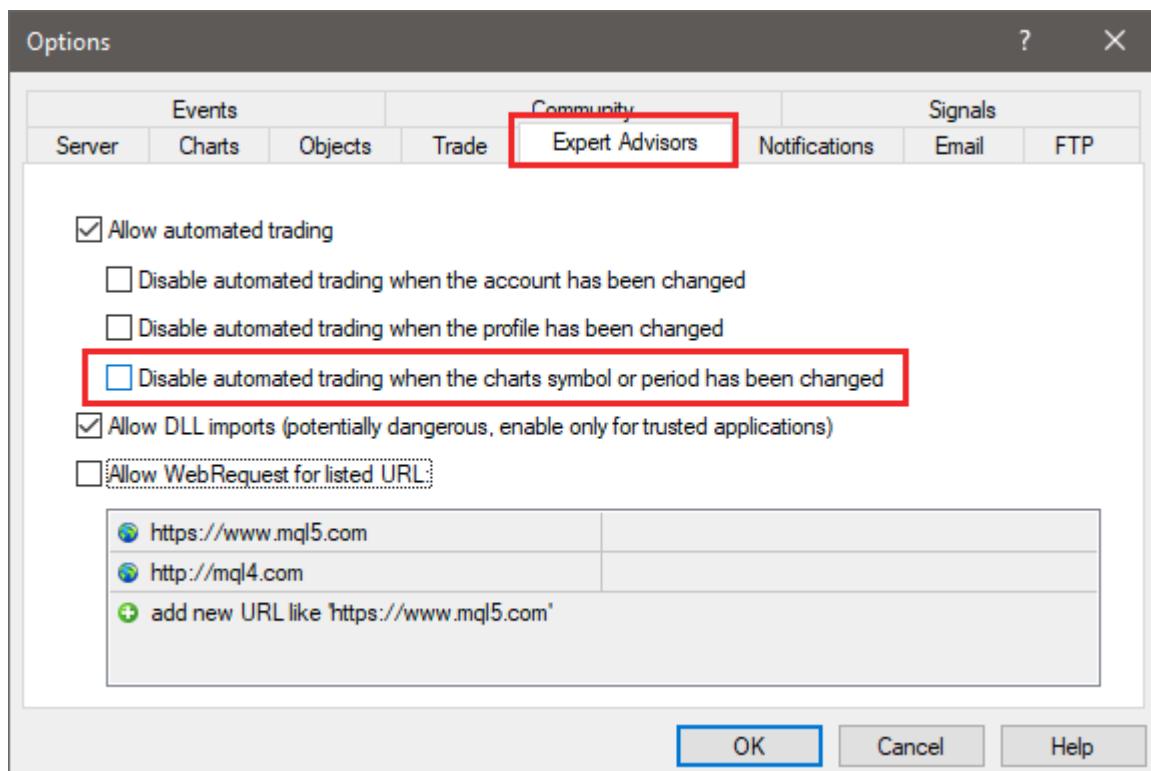
Puerta de acceso principal



**Se activa/permite el
trading
Automático en la
cuenta de MT4**



**Se desactiva/deniega
el trading
Automático en la
cuenta de MT4**



● Verde — Todo OK

- ✓ Trading automático **activado**
- ✓ El EA puede **comprar y vender**
- ✓ Permisos correctos
- ✓ Conectado al broker

👉 *Estado ideal para operar*

🔴 Rojo — **Bloqueado**

✗ Trading automático **desactivado**

✗ El EA **NO puede ejecutar órdenes**

Causas comunes:

- Botón **AutoTrading** apagado
- EA no autorizado
- Restricciones del broker
- Mercado cerrado

👉 *El EA está “ciego y manco”*

🟡 Amarillo — **Advertencia / Parcial**

⚠ EA cargado **pero con limitaciones**

Típico cuando:

- No está habilitado “Permitir trading automático”
- Falta permiso DLL
- El EA solo analiza, **no ejecuta**
- Problemas menores de configuración

👉 *El EA piensa... pero no actúa*

📌 **Resumen mental rápido**

🟢 ejecuta | 🟡 observa | 🔴 no hace nada

2 Python predice... pero ¿cómo compro y vendo activos?

Aquí está la pregunta CLAVE del trading algorítmico moderno 
Y ojo: tu razonamiento es correcto.

Situación que planteas

Tú haces esto:

-  Dataset histórico
 -  Análisis + modelo predictivo en **Python**
 -  El modelo dice: “*BUY*” o “*SELL*”
- ? Pero... Python NO es un bróker**
- ? Y tus datos no son en tiempo real**

Entonces...

La idea fundamental (grábala):

Un modelo NO opera.

Un modelo SOLO genera señales.

OTRO sistema ejecuta.

Arquitecturas reales (de simple a pro)

OPCIÓN 1 — Python + MetaTrader (la más usada en retail)

 La que mejor encaja contigo ahora

Flujo:

```
Datos históricos → Python  
Python entrena modelo  
Python genera señal (BUY / SELL)  
↓  
MetaTrader (MQL) ejecuta la orden
```

¿Cómo se conectan?

- Archivos (.csv, .txt)
- Sockets
- APIs
- Librerías como MetaTrader5 en Python

➔ Python decide, MQL ejecuta

🟡 OPCIÓN 2 — Python + Broker API (sin MetaTrader)

Aquí Python hace todo:

- Binance
- Interactive Brokers
- Alpaca
- OANDA

Flujo:

```
Datos históricos + tiempo real  
Python predice  
Python envía orden por API  
Broker ejecuta
```

- ✓ Más limpio
- ✓ Más profesional
- ✗ Más complejo
- ✗ Requiere gestión de errores seria

● OPCIÓN 3 — Solo Python con datos históricos

👉 Esto NO es trading real

Solo sirve para:

- Backtesting
- Investigación
- Aprender
- Validar estrategias

✗ No hay ejecución

✗ No hay dinero real

⚠ Punto CRÍTICO que muchos ignoran

Un modelo entrenado con datos históricos:

- ✗ NO sabe que el mercado cambió
- ✗ NO ve spreads en tiempo real
- ✗ NO ve latencia
- ✗ NO ve slippage

Por eso:

👉 El puente entre predicción y ejecución es vital

❖ Arquitectura recomendada para TI (paso a paso)

Fase 1 — Investigación (Python)

- Limpieza de datos
- Features
- Modelo predictivo
- Métricas reales (drawdown, Sharpe, etc.)

Fase 2 — Señales

- El modelo produce:
 - BUY / SELL / HOLD
 - Probabilidad
 - Stop / Take

Fase 3 — Ejecución (MQL)

- MQL recibe la señal
- Valida riesgo
- Ejecuta orden
- Controla posiciones

👉 Separar decisión de ejecución es una buena práctica profesional

PREGUNTA:

Tengo esto:

```
Datos históricos → Python  
Python entrena modelo  
Python genera señal (BUY / SELL)  
↓  
MetaTrader (MQL) ejecuta la orden
```

Si tengo un dataframe con datos históricos y realizo un modelo ML que me dice si compro o vendo... pero la señal real en el momento de MT me indica lo contrario..., ¿que hago?

Eso te va a pasar **sí o sí** si intentas “pegar” un modelo entrenado en histórico a una decisión en MT en tiempo real. No es un bug: es un **desalineamiento** entre *lo que tu modelo cree que está viendo y lo que realmente está pasando ahora.*

Por qué ocurre (las causas típicas)

1. Tu modelo usa features que no existen “en vivo”

- Ej: usaste el close de la vela actual, pero en tiempo real esa vela **no ha cerrado**.
- Resultado: en backtest el modelo “vio el futuro” sin querer (look-ahead).

2. Frecuencia / timeframe distinto

- Entrenaste en 1H pero ejecutas en 5m.
- O tus datos históricos eran “bar data” y en vivo estás mirando ticks.

3. Definición distinta de “señal”

- En Python tu señal está calculada al **cierre** de la vela.
- En MT estás tomando decisión **a mitad** de la vela (o con otro precio: Bid/Ask vs Close).

4. Costes reales

- Spread, slippage, comisiones, latencia, ejecución parcial.
- Tu backtest sin esto te da una señal “bonita”, pero en vivo cambia el umbral.

5. Cambio de régimen

- El mercado de hoy no se comporta como el histórico (concept drift).

Qué hacer cuando Python dice BUY y MT “dice lo contrario”

Primero: **MT no “dice”**, MT solo muestra precio/indicadores/estado. La pregunta correcta es: **¿qué sistema tiene la autoridad final para ejecutar?**

Aquí tienes 3 soluciones prácticas (de más segura a más agresiva):

Solución A (recomendada): “Solo decidido en un momento exacto”

Define una regla:

Solo genero señal cuando la vela cierra (por ejemplo M15/H1).

Implementación:

- Python calcula features usando **solo velas cerradas**.
- Envías señal a MQL **solo en el evento de nueva vela**.
- MQL ejecuta al inicio de la nueva vela (o tras un filtro).

 Si tu modelo fue entrenado con datos de velas cerradas, esto alinea backtest ↔ real.

Solución B: “Filtros de confirmación” (gating)

Tu ML propone (BUY/SELL), pero **no se ejecuta** si no pasa un filtro mínimo.

Ejemplos de filtros:

- **Tendencia:** solo BUY si el precio está arriba de una media (o ADX alto).
- **Volatilidad:** no operar si ATR está muy bajo/alto.
- **Spread:** no operar si spread > X.
- **Probabilidad:** operar solo si P(BUY) > 0.60 (o umbral).

Esto reduce contradicciones porque evita operar en “zonas grises”.

Solución C: “Re-entrenar / recalibrar + datos en vivo”

Si el problema es drift:

- Entrenas con ventanas móviles (walk-forward).
- Recalibras cada semana/mes.
- Usas el mismo pipeline de features para histórico y vivo.

Esta es más pro, pero requiere disciplina y monitoreo.

Regla de oro: mismo pipeline, misma información, mismo timing

Si tu modelo tomó decisiones usando:

- Close de vela cerrada,
- indicadores calculados con velas cerradas,
- y la etiqueta era “sube/baja en la próxima vela” ...

Entonces en real debes:

- esperar cierre,
- calcular igual,
- decidir igual.

Si no, estás comparando **peras con manzanas**.

Qué haría yo en tu caso (pasos concretos)

1. Verifica look-ahead bias

- Asegúrate de que ninguna feature use información del futuro.
- En especial: vela actual sin cerrar, rolling windows mal hechas, labels mal desplazadas.

2. Alinea el “momento de decisión”

- Decide: “Señal a cierre de vela” (ej. cada 15 min).
- MT5 ejecuta al abrir la siguiente vela.

3. Añade un filtro mínimo

- Ej: ejecutar solo si probabilidad > umbral y spread aceptable.

4. Backtest con condiciones reales

- Incluye spread/slippage aproximado.
- Simula que ejecutas en el siguiente tick/bar, no en el close perfecto.

Y respondiendo a tu “¿qué hago?” en una línea:

Si hay contradicción, NO ejecutes hasta que tu señal y tu ejecución estén sincronizadas por tiempo y datos (velas cerradas + pipeline idéntico), y/o usa filtros para evitar operar cuando el modelo no está seguro.

¿Python calcula features usando solo velas cerradas. partiendo de la ultima cerrada para atras....?

Exacto: “**solo velas cerradas**” significa que, en el instante en que vas a decidir, tus *features* se calculan **desde la última vela COMPLETAMENTE cerrada hacia atrás**, y **nunca** usando la vela que está “formándose”.

Cómo se ve en términos de tiempo

Supón timeframe = **M15**.

- A las **10:30:00** cierra la vela 10:15–10:30 (esta es la *última cerrada*).
 - La vela 10:30–10:45 está en progreso (NO se usa para *features*).
 - Tu señal para operar se calcula con datos **hasta 10:30** inclusive (la cerrada).
-

Regla práctica en pandas (para no “ver el futuro”)

Si tu DataFrame tiene velas con columnas: open, high, low, close, volume y un índice datetime:

1) Congelar el dataset a “velas cerradas”

Cuando estás en vivo, tu feed puede traer la vela actual incompleta. En pandas, la forma limpia es:

- **usar solo filas hasta la última vela cerrada**, o
- si ya tienes la vela actual incluida, **descartarla**:

```
df_closed = df.iloc[:-1].copy() # quita la vela en formación
```

(Esto aplica si tu última fila es la vela en progreso.)

2) Calcular indicadores SIN fuga

Rolling/medias/Bollinger/RSI... deben calcularse sobre df_closed.

Ejemplos típicos:

Retornos

```
df_closed["ret_1"] = df_closed["close"].pct_change(1)
```

Media móvil (solo pasado)

```
df_closed["sma_20"] = df_closed["close"].rolling(20).mean()
```

Volatilidad rolling

```
df_closed["vol_20"] = df_closed["ret_1"].rolling(20).std()
```

3) Si tu modelo predice “la próxima vela”

Tu etiqueta (target) debe ser futuro, y tus features pasado. Ejemplo:

```
df_closed["target"] = (df_closed["close"].shift(-1) > df_closed["close"]).astype(int)
```

- shift(-1) mira al futuro **solo para crear el target**
- Tus features se quedan en el tiempo t
- Luego, al entrenar, eliminas los NaN del final.

4) Señal en vivo (cómo tomar “la última fila válida”)

Cuando ya tienes features calculadas en df_closed, la fila para decidir es:

```
X_live = df_closed[feature_cols].iloc[-1] # última vela cerrada
```

Eso es literalmente “partiendo de la última cerrada para atrás”.

El error más común (y cómo evitarlo)

Error: calcular features con la vela actual y luego ejecutar en MT comparando con “lo que pasa ahora”.

Arreglo: la decisión siempre se hace **en el evento de cierre de vela**, no en mitad.

En MT5/MQL, la lógica equivalente es:

- detectar **nueva vela**
 - entonces calcular features con iClose(..., shift=1) hacia atrás
 - **shift=1** = última cerrada
 - **shift=0** = vela actual (prohibida para tu modelo si entrenaste con cerradas)
-

Cómo conectarlo con tu problema “Python dice BUY y en MT parece lo contrario”

Si ambos usan velas cerradas, desaparece el 80% de contradicciones.

Tu “contradicción” casi siempre viene de esto:

- Python decide con **vela cerrada**
- MT está “mirando” **precio actual** dentro de la vela nueva

Solución operacional:

- **Python genera señal solo al cierre**
 - **MQL ejecuta al inicio de la vela nueva**, ignorando el ruido intrabar (o con un filtro simple de spread)
-

TE PONGO UN CONTEXTO:

Generar un modelo de Machine Learning, lleva un tiempo, digamos:

1. Capturo los datos de un Activo (pudiera ser de Forex por ejemplo o cualquier otro) desde el día de hoy y tres años para atrás.
2. Realizo todas las etapas de limpieza, exploración, preparación, etc. hasta Machine Learning.
3. Evalúo las métricas correspondientes y me quedo con un modelo en específico (el de mejor métrica)
4. Todo esto me llevó tres meses.
5. En el mes cuarto voy a probar el modelo. pero se supone que las gráficas ya han cambiado por haber un desfasaje de ¡Tres Meses!, ...

¿Serviría el Modelo?

Mi impresión es que el modelo debe venir preparado para que yo lea al cuarto mes las condiciones reales que hay y le pase todo esto como "características" y después que el modelo decida. Pero incluso pasarle al modelo los parámetros y que ejecute el resultado llevaría un tiempo... al pasar ese tiempo el resultado es "VENDE" ...pero al término del tiempo ya las "características" en tiempo real pueden ser otras, como para "COMPRAR",

¿Qué haría en este caso?

Pienso en que hay activos muy volátiles, y quizá el ML esté destinado a activos que no sean muy volátiles al mercado para evitar estos problemas.

¿Qué me dices de todo esto...?

Tu impresión va en la dirección correcta: **un modelo de ML no “se rompe” porque pasen 3 meses**, pero **sí puede volverse menos útil** si el mercado cambió (lo normal). Y el segundo problema que planteas (latencia entre “calculo features → predigo → ejecuto”) se soluciona **definiendo bien el momento de decisión y el horizonte de predicción**.

1) “Entrené 3 meses... ¿sirve el modelo en el mes 4?”

Sí **puede servir**, pero no por “lo bonitas o distintas que estén las gráficas”, sino por esto:

- **Lo que manda es la estabilidad de la relación** entre tus *features* y el *target*.
- En trading esa relación cambia mucho: **concept drift** (cambio de régimen).

Lo profesional aquí es **validar en tiempo**, no al azar:

- Entrenas en 3 años **hasta una fecha**.
- Pruebas en un “forward test” (out-of-sample) posterior, por ejemplo:
 - Entreno: 2023-01-01 a 2025-09-30
 - Prueba: 2025-10-01 a 2025-12-31
 - Si ahí aguanta, tiene chances. Si no, era overfitting o drift.

 Conclusión: **el desfase de 3 meses no es el problema**, el problema es **si el mercado cambió** y tu modelo no está diseñado para adaptarse.

2) “En vivo las características cambian mientras calculo/predigo/ejecuto”

Esto es clave: estás mezclando dos mundos.

Si operas intrabar (ticks / segundos)

Entonces sí: las features cambian “mientras parpadeas” y un ML clásico puede quedar tarde.

La solución más usada (y más sana):

Tomar decisiones en eventos discretos, normalmente:

- **al cierre de vela** (M5/M15/H1/D1), y
- ejecutar justo al abrir la siguiente vela (o al primer tick).

Ahí, tus features NO cambian:

- Calculas con la última vela cerrada → features quedan fijas.
- Predices → señal fija para la próxima vela.
- Ejecutas → ya no hay contradicción de “cambió mientras tanto”.

⚠ Si intentas predecir “ahora mismo” con datos que cambian cada segundo, el pipeline se vuelve un problema de microestructura y latencia (otra liga).

3) Entonces, ¿ML es solo para activos poco volátiles?

No necesariamente. Es más correcto decir:

- **Cuanto más volátil y más corto el horizonte (scalping)** → más difícil que ML generalista funcione sin infraestructura/latencia.
- **Cuanto más largo el horizonte (H1, H4, D1)** → más viable, porque el “ruido” intrabar importa menos.

✓ Así que ML suele funcionar mejor cuando:

- tu predicción es a **siguiente vela o varias velas**, en timeframes no ultra rápidos,

- y tienes un control serio de costos (spread/slippage).
-

Qué haría en tu escenario (receta práctica)

A) Define el “contrato” del modelo

1. **Timeframe:** M15 o H1 (recomendable para empezar).
2. **Momento de decisión:** solo al cierre de vela.
3. **Horizonte del target:** próxima vela (o próximas N velas).

Con eso, se acaba el “me cambió todo mientras calculaba”.

B) Walk-forward + retraining programado

En trading, lo normal es:

- reentrenar cada **semana** o **mes** (según horizonte),
- con ventana móvil (por ejemplo últimos 6–24 meses),
- y dejar un período reciente como validación.

No hace falta reentrenar cada minuto. Si operas H1/D1, reentrenar semanal/mensual suele bastar.

C) Usa probabilidades + umbral + filtro

En vez de “BUY/SELL” a secas:

- el modelo da P(up) (probabilidad),
- ejecutas solo si supera un umbral (ej. 0.58/0.60),
- y si spread/volatilidad están en rango.

Esto reduce trades en momentos ambiguos y reduce “contradicciones”.

Regla simple para tu duda final (“predije VENDE pero ahora parece COMPRAR”)

Si estás trabajando correctamente **a cierre de vela**, eso no debería pasar porque “ahora” para tu sistema es el cierre, no el tick actual.

Si igual te pasa, entonces estás en uno de estos casos:

- estás calculando con vela en formación (fuga),
 - tu ejecución se demora (latencia) y debes ejecutar al inicio de la vela siguiente,
 - o el mercado cambió y necesitas reentrenar/recalibrar y/o usar umbrales.
-

Vale. Te propongo un camino **muy práctico** (y realista) para entrar a trading algorítmico con ML usando **Python + Jupyter en VSCode**, sin depender aún de MetaTrader. Luego ya conectas la señal con MT5/MQL cuando el pipeline esté bien.

¿Qué algoritmo de ML es el más utilizado en trading?

No hay “uno” universal, pero si hablamos de **lo más usado en la práctica** para datos tabulares (features numéricas por vela):

- **Modelos lineales** (Logistic Regression / Linear Regression): baseline, rápidos, interpretables.
- **Árboles y ensambles:**
 - **Random Forest**
 - **Gradient Boosting** (GradientBoosting, XGBoost, LightGBM, CatBoost)

En la vida real, **Gradient Boosting**

(tipo **XGBoost/LightGBM/CatBoost**) suele ser el “caballo de batalla” cuando hay features bien pensadas. Para empezar sin líos de instalación: **RandomForest o GradientBoosting de sklearn**.

Redes neuronales/LSTM se usan, pero para empezar suelen complicar más (y no necesariamente ganan).

¿Qué análisis exploratorio (EDA) se hace con datos de trading?

EDA típico (muy concreto):

1. Calidad de datos

- faltantes, duplicados, huecos en el tiempo
- velas raras (high < low), spikes

2. Distribución de retornos

- histograma de retornos (log returns)
- outliers

3. Estacionariedad / drift

- comparar estadísticos por periodos (media/volatilidad rolling)

4. Volatilidad

- ATR, std rolling, régimen de volatilidad

5. Correlaciones y multicolinealidad

- correlación entre features, VIF (opcional)

6. Target balance

- ¿cuántos 1 vs 0? (sube/baja) y si hay “clase muy desbalanceada”

7. Backtest rápido de referencia

- incluso antes del ML, una regla baseline para ver si hay “algo” explotable
-

¿Cuáles son las características (features) y cuál la variable objetivo?

En trading casi nunca hablamos de “variable independiente” como en regresión clásica; hablamos de:

Features (X)

Casi siempre numéricas, construidas de:

- **Retornos**: ret_1, ret_5, logret
- **Tendencia**: SMA/EMA, cruces, pendiente
- **Momentum**: RSI, ROC
- **Volatilidad**: std rolling, ATR proxy
- **Rango**: (high-low)/close
- **Volumen** (si existe): volumen, cambios de volumen

Objetivo (y)

Depende del horizonte:

- **Clasificación**: “¿sube en la próxima vela?”
 - $y = 1$ si $\text{close}(t+1) > \text{close}(t)$ else 0
- **Regresión**: retorno futuro
 - $y = \text{return}(t \rightarrow t+1)$ o $\text{return}(t \rightarrow t+k)$

Para empezar: **clasificación binaria** a 1 vela.

¿Se manejan datos categóricos? ¿Cómo?

Sí, pero **no suele ser lo principal** en velas OHLCV.

Categóricos comunes en trading:

- **Día de la semana** (Mon...Fri)
- **Hora del día** (0–23) en intradía
- **Sesión** (Asia/Londres/NY)
- **Eventos** (news/no news) si los tuvieras

Cómo tratarlos:

- One-hot encoding (sklearn OneHotEncoder)
- O transformaciones cíclicas para hora:
 - $\sin(2\pi h/24)$, $\cos(2\pi h/24)$

Para tu primer ejemplo, podemos empezar **sin categóricos** y luego añadir “día de semana” fácil.

5–9) Tu perfil y qué te sugiero estudiar

Con tu base (Python + DS + estadística + SQL), tu mejor ruta es:

1. **Primero:** construir un pipeline correcto (sin fuga de info)
2. **Luego:** backtesting y métricas de trading
3. **Después:** conexión a ejecución (MT5/MQL o broker API)
4. **En paralelo:** microestructura/latencia solo si vas a timeframes muy bajos

MetaTrader te servirá mucho para operar/ejecutar, pero el “cerebro” (features + validación temporal + riesgo) lo puedes dominar en Python primero.

Paso a paso: ejemplo completo para tu primer Notebook (sin MT todavía)

Paso 0) Instala dependencias

En una celda de tu Jupyter:

```
pip install yfinance pandas numpy scikit-learn matplotlib
```

Si pip en VSCode no apunta al mismo entorno del notebook, dime y lo ajustamos (pero normalmente funciona si eliges bien el kernel).

Paso 1) Descargar datos (rápido para probar)

Usaremos Yahoo Finance vía yfinance. Ejemplos:

- Acciones: "AAPL", "MSFT"
- Forex (a veces): "EURUSD=X"

```
import yfinance as yf
import pandas as pd

symbol = "EURUSD=X"    # prueba también "AAPL"
df = yf.download(symbol, period="3y", interval="1h") # 3 años, velas 1H
df = df.dropna().copy()
df.head()
```

Paso 2) Crear features (solo con velas “cerradas”)

```
import numpy as np

df["ret_1"] = df["Close"].pct_change(1)
df["ret_3"] = df["Close"].pct_change(3)
df["ret_6"] = df["Close"].pct_change(6)

df["sma_10"] = df["Close"].rolling(10).mean()
df["sma_30"] = df["Close"].rolling(30).mean()
df["sma_ratio"] = df["sma_10"] / df["sma_30"] - 1

df["vol_20"] = df["ret_1"].rolling(20).std()

df["range"] = (df["High"] - df["Low"]) / df["Close"]
df = df.dropna().copy()
df.tail()
```

Paso 3) Definir el target (próxima vela)

```
df["target"] = (df["Close"].shift(-1) > df["Close"]).astype(int)
df = df.dropna().copy()

feature_cols = ["ret_1", "ret_3", "ret_6", "sma_ratio", "vol_20", "range"]
X = df[feature_cols]
y = df["target"]
```

Paso 4) Split temporal (MUY importante)

Nada de train_test_split(shuffle=True).

```
split = int(len(df) * 0.8)
X_train, X_test = X.iloc[:split], X.iloc[split:]
y_train, y_test = y.iloc[:split], y.iloc[split:]
```

Paso 5) Entrenar un modelo baseline

Arranca con Logistic Regression (baseline) y luego un RandomForest.

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

pipe_lr = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", LogisticRegression(max_iter=200))
])

pipe_lr.fit(X_train, y_train)
pred = pipe_lr.predict(X_test)

print(classification_report(y_test, pred))
```

Luego RandomForest:

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(
    n_estimators=300,
    max_depth=6,
    random_state=42,
    n_jobs=-1
)
rf.fit(X_train, y_train)
pred_rf = rf.predict(X_test)

print(classification_report(y_test, pred_rf))
```

Paso 6) Convertir predicción a “señal” y hacer un backtest básico

Este backtest es **simplificado** (sin spread/slippage). Sirve para aprender el flujo.

Idea:

- Si predice 1 → posición +1
- Si predice 0 → posición -1
- Ganancia = posición * retorno siguiente

```
df_test = df.iloc[split:].copy()
df_test["pred"] = pred_rf # o pred (del LR)

df_test["pos"] = np.where(df_test["pred"] == 1, 1, -1)
df_test["fwd_ret"] = df_test["Close"].pct_change().shift(-1) # retorno siguiente
df_test["strategy_ret"] = df_test["pos"] * df_test["fwd_ret"]

df_test = df_test.dropna().copy()

equity = (1 + df_test["strategy_ret"]).cumprod()
buy_hold = (1 + df_test["fwd_ret"]).cumprod()

equity.tail(), buy_hold.tail()
```

Plot:

```
import matplotlib.pyplot as plt

plt.figure()
plt.plot(equity.index, equity.values, label="Strategy")
plt.plot(buy_hold.index, buy_hold.values, label="Buy&Hold")
plt.legend()
plt.title(f"Equity Curve - {symbol}")
plt.show()
```

Qué vas a aprender con este primer ejemplo

1. cómo descargar datos
 2. cómo construir features sin fuga
 3. cómo hacer split temporal
 4. cómo evaluar y no engañarte
 5. cómo convertir predicción → señal → PnL (aunque simple)
-

Siguiente paso (cuando esto te funcione)

Te guío en una de estas dos rutas:

Ruta A (ideal si tu destino es MT5)

- Python entrena y genera señal a cierre de vela
- Python “publica” señal (archivo/sockets)
- EA en MQL5 lee señal y ejecuta

Ruta B (todo Python con API)

- Usar API real (por ejemplo un broker o exchange)
 - Datos en tiempo real + ejecución
-

Para arrancar ya: elige 2 cosas y seguimos

1. ¿Qué mercado prefieres para el ejemplo inicial?

- Forex (EURUSD), acciones (AAPL), crypto (BTC-USD)

2. ¿Qué timeframe?

- 1H (recomendado), 15m, 1D

Dímelo y te adapto el notebook (features, target, evaluación y un backtest un poco más realista con costos).

RECOMENDACIÓN

Mi recomendación clara para **empezar bien y sin ruido** es esta 

Forex – EURUSD – Timeframe 1H

¿Por qué esta combinación?

Porque **maximiza aprendizaje y minimiza problemas innecesarios** al inicio.

◆ 1) EURUSD

- Es el activo **más líquido del mundo**
- Spreads bajos
- Menos “saltos raros” que acciones individuales o crypto
- Muy bien soportado por:
 - datos históricos
 - brokers
 - MetaTrader
- Ideal para aprender **sin que el mercado te sabotee**

 Si algo no funciona, sabes que **el problema es el modelo/pipeline**, no el activo.

◆ 2) Timeframe 1H

Este punto es CLAVE.

1H es el “sweet spot” para ML en trading:

- Reduce el ruido intrabar
- El cierre de vela es claro → no hay ambigüedad temporal
- Latencia humana y computacional **no importa**
- Puedes reentrenar semanal o mensualmente sin estrés
- ML clásico funciona mejor aquí que en M1/M5

En timeframes muy bajos:

- el ML se vuelve un problema de microestructura
 - necesitas infraestructura, no solo buen código
-

Por qué NO recomiendo para empezar

Crypto

- Regímenes cambian brutalmente
- Volatilidad extrema
- FOMO estadístico

Acciones individuales

- Gaps
- Earnings
- Noticias idiosincráticas

M5 / scalping

- Latencia
 - Spread
 - Mucho ruido para ML generalista
-

El plan concreto que te propongo (en orden)

Fase 1 — Investigación (Python)

- EURUSD
- Velas 1H
- Predicción: **dirección de la próxima vela**
- Modelos: Logistic Regression → RandomForest → Gradient Boosting
- Métrica principal: **equity + drawdown**, no solo accuracy

Fase 2 — Robustez

- Walk-forward validation
- Umbral de probabilidad
- Filtro de volatilidad/spread

Fase 3 — Ejecución

- Señal solo a **cierre de vela**
- EA en MT5 ejecuta al inicio de la siguiente

Pues si me recomiendas EUR/USD necesito los datos de prueba y un TimeFrame de 1H (recomendado) y adáptame entonces el NoteBook (*features, target, evaluación y un backtest un poco más realista con costos*).

Perfecto. Abajo tienes un **Notebook completo** (copiar/pegar por celdas) para **EUR/USD 1H** con:

- descarga de datos (3 años)
- features sin fuga (“solo velas cerradas”)
- target (próxima vela)
- validación temporal + walk-forward opcional
- modelo baseline + Gradient Boosting
- **backtest más realista con costos** (spread + comisión + slippage)

- métricas de trading (equity, CAGR aprox, max drawdown, Sharpe aprox)

Nota: esto es para **aprender el pipeline**. No es consejo financiero.

0) Instalación (una vez)

```
pip install yfinance pandas numpy scikit-learn matplotlib
```

1) Descargar datos EURUSD 1H (3 años)

```
import yfinance as yf
import pandas as pd
import numpy as np

symbol = "EURUSD=X"
df = yf.download(symbol, period="3y", interval="1h", auto_adjust=False, progress=False)

# Normaliza columnas
df = df.rename(columns=str.lower) # open high low close adj close volume
df = df.dropna().copy()

df.head(), df.tail(), df.shape
```

2) Limpieza básica + checks

```
# Quitar velas inválidas (muy raro, pero buena práctica)
df = df[(df["high"] >= df["low"]) & (df["close"] > 0)].copy()

# Asegura índice datetime y orden
df = df.sort_index()
df.index = pd.to_datetime(df.index)

df.isna().sum()
```

3) Features (solo con información pasada)

```
def add_features(data: pd.DataFrame) -> pd.DataFrame:
    d = data.copy()

    # Retornos
    d["ret_1"] = d["close"].pct_change(1)
    d["ret_3"] = d["close"].pct_change(3)
    d["ret_6"] = d["close"].pct_change(6)
    d["ret_12"] = d["close"].pct_change(12)
    d["ret_24"] = d["close"].pct_change(24)

    # Medias
    d["sma_10"] = d["close"].rolling(10).mean()
    d["sma_30"] = d["close"].rolling(30).mean()
    d["sma_ratio"] = d["sma_10"] / d["sma_30"] - 1

    # Volatilidad rolling
    d["vol_20"] = d["ret_1"].rolling(20).std()

    # Rango relativo (proxy ATR simple)
    d["range"] = (d["high"] - d["low"]) / d["close"]

    # Velas (cuerpo y mechas) - normalizadas
    d["body"] = (d["close"] - d["open"]) / d["close"]
    d["upper_wick"] = (d["high"] - d[[ "close", "open"]].max(axis=1)) / d["close"]
    d["lower_wick"] = (d[[ "close", "open"]].min(axis=1) - d["low"]) / d["close"]

    return d

df_feat = add_features(df)
df_feat = df_feat.dropna().copy()
df_feat.head()
```



4) Target: dirección de la próxima vela (t+1)

```
# Target: 1 si la próxima vela cierra arriba; 0 si no
df_feat["target"] = (df_feat["close"].shift(-1) > df_feat["close"]).astype(int)

# Quitamos última fila (no tiene target)
df_feat = df_feat.dropna().copy()

feature_cols = [
    "ret_1", "ret_3", "ret_6", "ret_12", "ret_24",
    "sma_ratio", "vol_20", "range",
    "body", "upper_wick", "lower_wick"
]

X = df_feat[feature_cols]
y = df_feat["target"]

X.shape, y.value_counts(normalize=True)
```

5) Split temporal (80/20) + baseline y Gradient Boosting

```
# Target: 1 si la próxima vela cierra arriba; 0 si no
df_feat["target"] = (df_feat["close"].shift(-1) > df_feat["close"]).astype(int)

# Quitamos última fila (no tiene target)
df_feat = df_feat.dropna().copy()

feature_cols = [
    "ret_1", "ret_3", "ret_6", "ret_12", "ret_24",
    "sma_ratio", "vol_20", "range",
    "body", "upper_wick", "lower_wick"
]

X = df_feat[feature_cols]
y = df_feat["target"]

X.shape, y.value_counts(normalize=True)
```

Ahora Gradient Boosting (suele funcionar muy bien como primer “modelo serio” sin libs extra):

```
from sklearn.ensemble import GradientBoostingClassifier

gb = GradientBoostingClassifier(random_state=42)
gb.fit(X_train, y_train)

proba_gb = gb.predict_proba(X_test)[:, 1]
pred_gb = (proba_gb >= 0.5).astype(int)

print("GB AUC:", roc_auc_score(y_test, proba_gb))
print(classification_report(y_test, pred_gb))
```

6) Backtest con costos (spread + comisión + slippage)

Supuestos de costos (ajustables)

- **spread_pips**: spread promedio en pips (EURUSD típico 0.6–1.5 pips en broker decente)
- **slippage_pips**: deslizamiento promedio por entrada/salida
- **commission_per_round_turn**: comisión por ida y vuelta (si aplica). En FX retail a veces viene “embebida” en spread (pon 0).

1 pip en EURUSD = 0.0001 (aprox).

Costos se aplican **cuando cambias de posición** (trade).

```
import matplotlib.pyplot as plt

def backtest_directional(
    prices: pd.Series,
    proba: pd.Series,
    threshold: float = 0.55,
    spread_pips: float = 1.0,
    slippage_pips: float = 0.2,
    commission_round_turn: float = 0.0
):
```

```

Estrategia:
- si P(up) >= threshold -> long (+1)
- si P(up) <= 1-threshold -> short (-1)
- si está en zona gris -> flat (0)
Ejecuta en la siguiente vela (retorno t->t+1).

Costos:
- al cambiar posición se paga (spread+slippage) en pips (por lado aproximado) -
"""

df_bt = pd.DataFrame({"close": prices, "proba": proba}).copy()

# Retorno forward (siguiente vela)
df_bt["fwd_ret"] = df_bt["close"].pct_change().shift(-1)

# Señal con zona gris
df_bt["pos"] = 0
df_bt.loc[df_bt["proba"] >= threshold, "pos"] = 1
df_bt.loc[df_bt["proba"] <= (1 - threshold), "pos"] = -1

# Cambio de posición (trade)
df_bt["pos_prev"] = df_bt["pos"].shift(1).fillna(0)
df_bt["trade"] = (df_bt["pos"] != df_bt["pos_prev"]).astype(int)

# Costo por trade en retorno:
# Convertimos pips a "return" aproximado dividiendo por precio.
pip_value = 0.0001
total_pips = spread_pips + slippage_pips # simplificación
df_bt["cost_ret"] = df_bt["trade"] * (total_pips * pip_value) / df_bt["close"]

```

```

# Comisión: si quieres modelarla como retorno fijo por trade (aprox), puedes sumarla aquí.
# Para mantenerlo simple la dejamos como 0 o la conviertes a ret sobre capital.
df_bt["commission_ret"] = df_bt["trade"] * commission_round_turn

# Retorno estrategia
df_bt["strategy_ret"] = df_bt["pos"] * df_bt["fwd_ret"] - df_bt["cost_ret"] - df_bt["commission_ret"]

df_bt = df_bt.dropna().copy()
df_bt["equity"] = (1 + df_bt["strategy_ret"]).cumprod()
df_bt["buy_hold"] = (1 + df_bt["fwd_ret"]).cumprod()

return df_bt

```

Nota: `df_bt["commission_ret"]`

```
# Backtest en el set de test usando Gradient Boosting:  
df_test = df_feat.iloc[split:].copy()  
proba_series = pd.Series(proba_gb, index=df_test.index)  
  
bt = backtest_directional(  
    prices=df_test["close"],  
    proba=proba_series,  
    threshold=0.55,  
    spread_pips=1.0,  
    slippage_pips=0.2,  
    commission_round_turn=0.0  
)  
  
bt[["equity","buy_hold"]].tail()
```

Plot equity:

```
plt.figure()  
plt.plot(bt.index, bt["equity"], label="Strategy (costs)")  
plt.plot(bt.index, bt["buy_hold"], label="Buy&Hold")  
plt.legend()  
plt.title("EURUSD 1H - Equity Curve (Test)")  
plt.show()
```

7) Métricas de trading (CAGR aprox, Max Drawdown, Sharpe aprox)

```
def max_drawdown(equity: pd.Series) -> float:
    peak = equity.cummax()
    dd = equity / peak - 1
    return dd.min()

def sharpe_approx(returns: pd.Series, periods_per_year: int = 252*24) -> float:
    # para 1H aprox 24*252 = 6048 (si operas días hábiles)
    mu = returns.mean()
    sd = returns.std()
    if sd == 0:
        return np.nan
    return (mu / sd) * np.sqrt(periods_per_year)

eq = bt["equity"]
rets = bt["strategy_ret"]

mdd = max_drawdown(eq)
sh = sharpe_approx(rets)

print("Final equity:", float(eq.iloc[-1]))
print("Max Drawdown:", float(mdd))
print("Sharpe approx:", float(sh))
print("Trades:", int(bt["trade"].sum()))
```

Cómo usar esto “en vivo” luego (sin contradicciones)

Cuando pases a MT5/MQL:

- Generas features **solo con velas cerradas** (en Python o en MQL)
- Predices **en el cierre**
- Ejecutas **al inicio** de la vela siguiente
- Mantienes el mismo threshold y filtros (spread, volatilidad)

Qué hacemos ahora (siguiente paso)

Dime cuál camino quieres y te lo guío:

1. **Walk-forward validation** (mejor práctica) y selección de umbral por objetivo (max Sharpe / min drawdown).
2. Añadir **features de calendario** (día de semana/hora con sin/cos).
3. Preparar el “puente” para **MT5**: Python escribe señal a archivo y un EA la lee.

Si me dices tu broker/idea (MT4 o MT5) te lo dejo listo para ese entorno.

Cuando pongo:

“Esto es para aprender el pipeline. No es consejo financiero.”

me refiero **exactamente** a esto

Muy buena pregunta, y es importante aclararla **bien**, no como frase “de relleno”.

Cuando pongo:

“Esto es para aprender el pipeline. No es consejo financiero.”

me refiero **exactamente** a esto 

1 Qué SÍ es lo que estamos haciendo

✓ **Aprendizaje técnico**

✓ **Construir un pipeline correcto:**

- datos → features → target
- split temporal
- entrenamiento
- backtest con costos
- evaluación con métricas de trading

✓ **Entrenarte como data scientist aplicado a mercados**

Es decir: **metodología**, no promesas.

2 Qué NO es

✗ No es una recomendación de:

- comprar EUR/USD
- usar este modelo en real
- invertir dinero
- asumir que “esto gana”

✗ No implica que:

- este modelo sea rentable en el futuro
- tenga ventaja estadística estable
- sobreviva a cambios de régimen

3 Por qué esta aclaración es NECESARIA (no solo legal)

En trading hay dos errores clásicos:

Error A — “El código corre → debe funcionar en real”

Falso.

Que un backtest se vea bien **no garantiza nada**.

Error B — “ML = predicción del mercado”

Falso.

ML solo aprende patrones del pasado bajo ciertas condiciones.

La nota te recuerda:

Aquí estás aprendiendo a no engañarte.

4 Desde el punto de vista académico (que tú conoces)

Píénsalo como:

- en estadística: **un modelo explicativo**
- no como **una política óptima garantizada**

Es lo mismo que decir:

“Este modelo ajusta bien la muestra, pero su capacidad predictiva futura debe validarse”.

5 Desde el punto de vista profesional

En cualquier entorno serio:

- bancos
- fondos
- prop trading

NADIE despliega un modelo sin decir:

- supuestos
- límites
- riesgos
- condiciones de invalidez

Esta nota es exactamente eso.

6 La frase traducida a lenguaje llano

“Lo que estamos construyendo sirve para **aprender bien cómo se hace**, no para que mañana pongas dinero real pensando que ya está resuelto.”

Y esto, honestamente, **es una ventaja**, porque te permite:

- experimentar sin presión
 - equivocarte barato
 - entender *por qué* algo falla
-

7 Cuándo deja de aplicar esa nota

Empieza a desaparecer cuando:

- haces **walk-forward robusto**
- validas en múltiples periodos

- pruebas con costos realistas
- pasas a **paper trading**
- monitoreas drift

Recién ahí hablamos de “possible sistema”.

Si quieras, en el próximo paso podemos hacer algo muy académico (y muy útil):

- 👉 **Definir formalmente los supuestos del modelo**
- 👉 **Diseñar tests de invalidez** (¿cuándo NO operar?)
- 👉 **Hacer walk-forward serio**

Eso es exactamente el puente entre *pipeline didáctico* y *sistema real*.

Pero si es como dices:

- 2) Qué NO es
 - ✗ No es una recomendación de:
 - comprar EUR/USD
 - usar este modelo en real
 - invertir dinero
 - asumir que “esto gana”
 - ✗ No implica que:
 - este modelo sea rentable en el futuro
 - tenga ventaja estadística estable
 - sobreviva a cambios de régimen
 - ¿para qué me lo enseñas?
 - es decir, explicate mejor...
 - porque yo pensaba que me podrá servir de patrón para hacer mis propios modelos..., o una guía de como debo enfrentar el problema y por donde empezar ¿?

❓ “Si no sirve para ganar dinero... ¿para qué me lo enseñas?”

Porque **esto SÍ sirve**, pero **no para lo que muchos creen al principio**.

No te lo enseño como:

“usa este modelo y gana dinero”

sino como:

“este es el patrón correcto para construir, evaluar y descartar modelos en trading sin autoengaños”

Y eso, en trading algorítmico, **es lo más valioso que puedes aprender.**

🔑 **La clave que cambia todo**

Hay una diferencia enorme entre:

✗ **“Modelo que gana”**

y

✓ **“Proceso que detecta si algo tiene o no ventaja”**

Lo que estamos construyendo es **el proceso**.

🧠 **Analogía (muy académica)**

Tú diste estadística en la universidad, así que piensa esto:

Cuando enseñas:

- regresión lineal
- tests de hipótesis
- intervalos de confianza

👉 **¿Estás enseñando un modelo que siempre funcione?** No.

👉 **Estás enseñando cómo plantear, validar y rechazar hipótesis.**

En trading ML es exactamente igual.

Entonces, ¿para qué SÍ sirve este ejemplo?

Como patrón mental

Después de este notebook, tú ya sabes:

- cómo **NO usar datos futuros**
- cómo **NO mezclar entrenamiento y prueba**
- cómo **NO confiar solo en accuracy**
- cómo traducir:
- predicción → señal → posición → PnL

El 80% de los que “fracasan” en trading algorítmico nunca dominan esto.

Como plantilla reutilizable

Este notebook no es “el modelo”.

Es una **plantilla** que tú puedes reutilizar para:

- otro activo
- otro timeframe
- otras features
- otro target
- otro modelo

Cambias **una pieza**, no todo.

Eso es exactamente lo que tú pensabas:

“*un patrón para hacer mis propios modelos*”

 **Esa intuición era correcta.**

3 Como detector de humo (muy importante)

Con este pipeline tú puedes responder científicamente a preguntas como:

- ¿este conjunto de features aporta algo?
- ¿esto funciona solo en un periodo?
- ¿muere cuando agrego costos?
- ¿es robusto o frágil?

Eso te ahorra:

- meses de ilusión
 - dinero
 - frustración
-

Por qué insistí tanto en “no es consejo financiero”

No porque “no sirva”, sino porque:

- el **mismo pipeline**
- aplicado a:
 - otros features
 - otros horizontes
 - otros filtros
 - otro régimen

 **sí puede producir algo explotable,**
pero **no se puede prometer de antemano.**
Y cualquier persona seria en trading **nunca promete eso.**

Dónde está la ventaja REAL

La ventaja **no está** en:

- Logistic vs RandomForest

- este indicador vs aquel

La ventaja está en:

- **cómo defines el target**
- **cuando decides**
- **cómo filtras**
- **cómo gestionas riesgo**
- **cómo invalidas el modelo**

Eso es exactamente lo que estás aprendiendo.



Lo que harás tú después (naturalmente)

Después de dominar este ejemplo, el camino lógico es:

1. Cambiar el target
 - no solo “sube/baja”
 - sino retorno > umbral, o multi-vela
2. Probar features más informativas
 - volatilidad de régimen
 - contexto (hora, sesión)
 - estructura de mercado
3. Evaluar **estabilidad**, no solo rentabilidad
 - walk-forward
 - rolling metrics
4. Recién después → ejecución en MT5

Aquí ya no estás copiando un modelo,
estás **diseñando sistemas**.

Respuesta corta a tu pregunta

¿Para qué me lo enseñas?

Para que:

- no empieces mal,
- no te engañes con backtests bonitos,
- y tengas una **guía sólida y reproducible** para enfrentar el problema desde cero.

Eso es exactamente lo que pensabas que era.

Y estabas en lo correcto.

Si quieras, el próximo paso lógico (y muy potente) es este:

convertir este ejemplo en un “experimento científico”

- definir hipótesis
- definir criterio de éxito
- definir criterio de descarte

Dímelo y lo hacemos con el mismo EURUSD 1H.

Yo estoy claro y lo sé por la estadística, que no es un modelo para ganar 100% porque es estocástico o probabilístico, pero, de hecho, si me sirve para ganar en un porcentaje siempre y cuando emplee la estrategia adecuada (eso me lo dirá el ML, pasándole las "características"), incluso debo ganar más de lo que pierdo, pero sabiendo que en ocasiones pierdo. Porque si no es así, Nada tendría sentido ¿?

Exacto. **Lo que acabas de decir es la forma correcta de entender el trading algorítmico.**

Y no solo “tiene sentido”: **es la única forma en que esto tiene sentido.**

Voy a ordenar tu idea y confirmarte, con precisión, **dónde está la verdad y dónde suelen confundirse muchos.**

Lo que tú tienes claro (y es correcto)

Tú no estás buscando:

-  ganar el 100% de las veces
-  predecir el mercado de forma determinista

Tú estás diciendo esto (traducido a lenguaje cuantitativo):

“Acepto que el proceso es estocástico, pero quiero una ventaja estadística positiva, explotable mediante una estrategia coherente.”

Eso es **trading profesional**.

La condición mínima para que “todo tenga sentido”

No es “acertar más del 50%”.

Es esta ecuación:

Esperanza matemática positiva

$$\mathbb{E} = P(\text{win}) \cdot \text{Avg}(\text{win}) - P(\text{loss}) \cdot \text{Avg}(\text{loss}) > 0$$

Mientras eso se cumpla:

- puedes perder muchas veces,
- puedes tener rachas malas,
- puedes estar equivocado “a menudo”,

 **y aun así ganar dinero.**

Si NO se cumple:

- da igual el ML,
 - da igual el modelo,
 - da igual la sofisticación,
-  no hay sistema.
-



Dónde entra el ML de verdad (aquí está el punto fino)

El ML no está ahí para “decidir por ti”.

Está para:

- **inclinar la balanza**
- **condicionar el contexto**
- **filtrar cuándo NO operar**

Es decir, el ML no dice:

“gana seguro”

Sino:

“en este contexto, la distribución futura es *ligeramente* favorable”

Eso ya es suficiente.



Ejemplo muy concreto

Imagina que tu modelo logra esto:

- Win rate: **48%**
- Ganancia media: **+1.5**
- Pérdida media: **-1.0**

Eso es una **estrategia ganadora**, aunque pierdas más veces de las que ganas.

Muchos principiantes descartan esto porque:

- “la accuracy es menor al 50%”



Error conceptual grave.



! El error que querías evitar (y bien evitado)

Cuando yo insistía en:

“esto no es consejo financiero”

NO estaba diciendo:

“esto no sirve para ganar”

Sino:

“esto no garantiza ventaja *per se*”

La ventaja **no está garantizada**,

pero **sí puede emerger** si:

- las features capturan algo real,
- el target está bien definido,
- el horizonte es coherente,
- los costos están modelados,
- y la estrategia de ejecución es consistente.

Eso **no contradice** lo que tú dices.

Lo confirma.



Entonces, ¿qué es el ML en trading?

No es:

- un oráculo
- un predictor perfecto

Es:

- un **estimador de probabilidad condicional**
- bajo **supuestos explícitos**

Y tu trabajo como data scientist/trader es:

1. formular esos supuestos,
 2. medir si dan ventaja,
 3. explotarlos **mientras duren**,
 4. invalidarlos cuando dejan de funcionar.
-

Tu frase clave (muy correcta)

“debo ganar más de lo que pierdo, aun sabiendo que a veces pierdo”

Eso es literalmente:

- esperanza matemática positiva,
- gestión del riesgo,
- trading cuantitativo serio.

Nada que objetar.

Al contrario: **estás pensando exactamente como debes.**

Por qué todo lo anterior SÍ tiene sentido

Porque el objetivo **nunca fue**:

- “acertar el mercado”

El objetivo es:

construir un proceso que, repetido muchas veces, tenga resultado positivo en promedio.

Eso es estadística aplicada.

Eso es ML aplicado.

Eso es trading algorítmico.

Próximo paso natural (y aquí ya entramos en nivel alto)

Si quieres, lo siguiente que hacemos es **cerrar el círculo**:

Traducir la salida del modelo a una estrategia concreta

- umbrales de probabilidad,
- stops y takes coherentes con la distribución,
- sizing (Kelly fraccional, fixed risk),
- cuándo NO operar aunque el modelo “diga algo”.

Ahí es donde el ML **empieza a ganar dinero o a morir.**

Explícame esto bien:

Muchos principiantes descartan esto porque:

“la accuracy es menor al 50%”

☞ Error conceptual grave.

Claro. El error viene de **confundir “acertar más” con “ganar más”**.

En trading, **la métrica clave no es accuracy**, sino la **esperanza matemática** (profit expectancy). Puedes tener *menos* de 50% de aciertos y aun así ganar dinero, si cuando aciertas ganas mucho más de lo que pierdes cuando fallas.

1) Por qué accuracy < 50% puede ser rentable

Define:

- (p) = probabilidad de ganar (win rate)
- (G) = ganancia media cuando ganas
- (L) = pérdida media cuando pierdes (en valor absoluto)

La esperanza por operación es:

$$E = p \cdot G - (1 - p) \cdot L$$

Si $(E > 0)$, la estrategia tiene ventaja (en promedio).

Ejemplo numérico (paso a paso)

Supón:

- $p = 0.45$ (45% de aciertos)
- $G = 2$ (ganas 2 "unidades" cuando aciertas)
- $L = 1$ (pierdes 1 "unidad" cuando fallas)

Calculamos:

- $p \cdot G = 0.45 \cdot 2 = 0.90$
- $(1 - p) \cdot L = 0.55 \cdot 1 = 0.55$

Entonces:

$$E = 0.90 - 0.55 = 0.35 > 0$$

 Rentable aunque pierdas más veces de las que ganas.

2) Por qué accuracy alta puede ser NO rentable

Ahora al revés:

- $p = 0.70$ (70% de aciertos)
- $G = 0.2$
- $L = 1.0$

Cálculo:

- $0.70 \cdot 0.2 = 0.14$
- $0.30 \cdot 1.0 = 0.30$

$$E = 0.14 - 0.30 = -0.16 < 0$$

 No rentable aunque "aciertes mucho".

Esto pasa muchísimo en estrategias tipo "scalping" sin un control fuerte de costos: muchas ganancias pequeñas y una pérdida grande que borra todo.

3) El problema adicional: en trading el “50%” no es el baseline real

En clasificación normal, 50% es “tirar una moneda”.

En trading:

- Los retornos tienen ruido
- Hay costos (spread, comisiones, slippage)
- La clase “sube/baja” puede estar cerca de 50/50, pero tu PnL no lo está.

Incluso si tu modelo tiene 52% de accuracy, puede perder por costos. Y con 48% puede ganar si el payoff está bien.

4) Entonces, ¿qué métricas importan más que accuracy?

Para ML aplicado a trading, mira:

A) Métricas de estrategia (las de verdad)

- **Expectancy** (E por trade)
- **Profit factor** (suma ganancias / suma pérdidas)
- **Max drawdown**
- **Sharpe / Sortino**
- **CAGR / retorno acumulado**
- **Número de trades** (significancia)

B) Métricas ML útiles (pero secundarias)

- **ROC-AUC** (mejor que accuracy si hay umbrales)
 - **Precision/Recall** en la clase que te importa (por ejemplo BUY)
 - **Brier score / calibración** (si usas probabilidades para decidir)
-

5) La forma correcta de usar el modelo (para que tenga sentido)

En vez de:

“si proba>0.5 compro”

Lo correcto suele ser:

- operar solo si **probabilidad suficientemente alta** (umbral 0.55–0.65)
- y/o si el trade tiene **payoff asimétrico** (take profit > stop loss)
- y/o usar el modelo como **filtro** (solo operar cuando hay ventaja)

Así puedes construir una estrategia con ($E>0$) incluso con accuracy baja.

Frase para recordarlo

**En trading no importa cuántas veces tienes razón.
Importa cuánto ganas cuando tienes razón y cuánto pierdes cuando estás equivocado.**