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)

Estas features son “clásicas” y seguras para empezar.

```
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

```
split = int(len(df_feat) * 0.8)

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

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

pipe_lr = Pipeline([
    ("scaler", StandardScaler()),
    ("clf", LogisticRegression(max_iter=300))
])

pipe_lr.fit(X_train, y_train)
proba_lr = pipe_lr.predict_proba(X_test)[:, 1]
pred_lr = (proba_lr >= 0.5).astype(int)

print("LogReg AUC:", roc_auc_score(y_test, proba_lr))
```

```
print(classification_report(y_test, pred_lr))
```

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) -> round-turn 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

# 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.