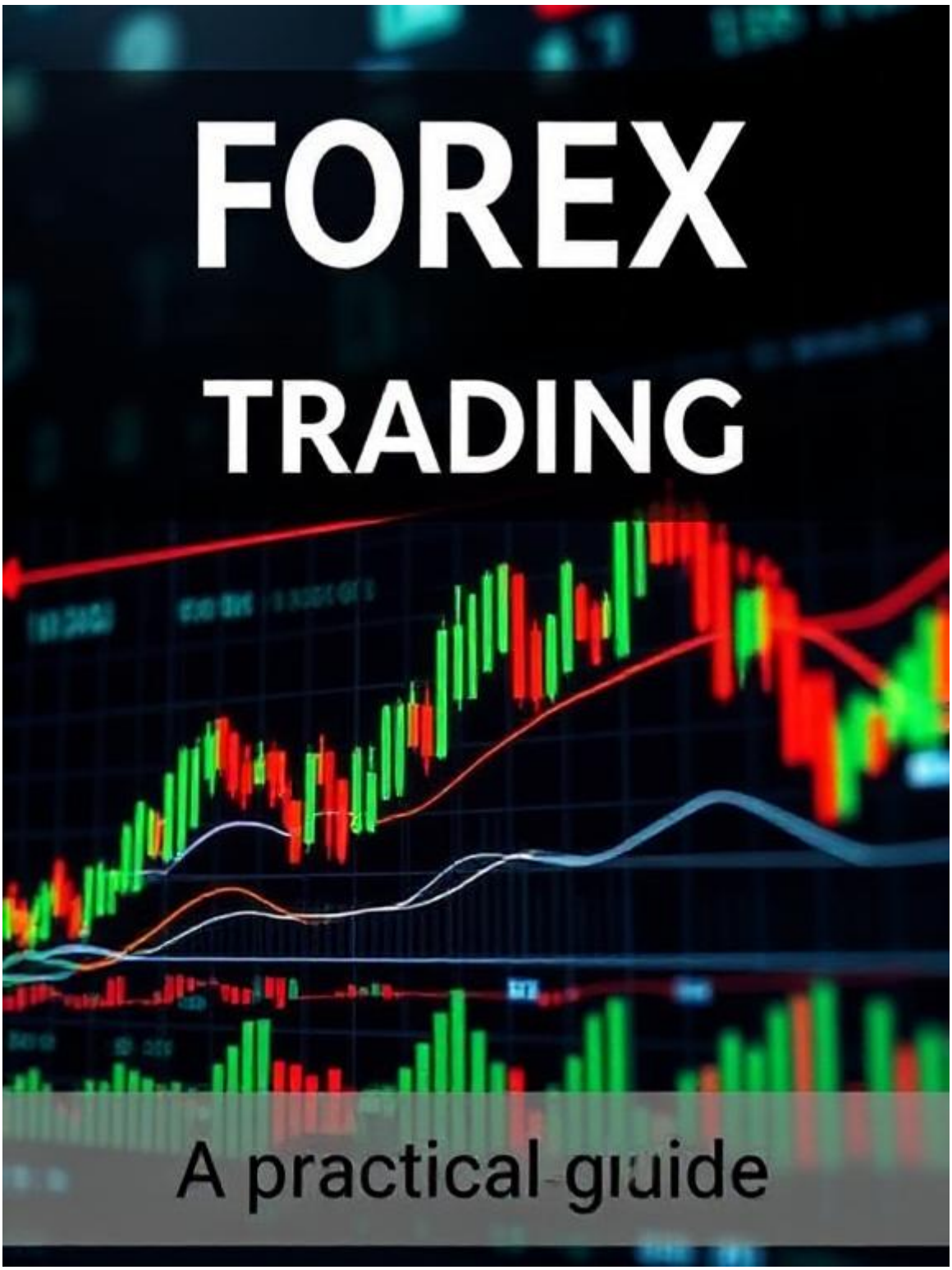

FOREX
TRADING
A practical guide

# Automatically generated PDF

This document is automatically generated PDF version of the EPUB source. For best reading experience the original electronic book (EPUB) is recommended:

github.com/hillsmithbooks/ebook



# Acknowledgement

First of all thank you for taking interest in my writing! I encourage you to send feedback and comments about the book. It is a great motivation for me to continue writing and to incorporate the observations into my future works. In the meantime: enjoy reading and I wish you useful lessons!

Pest County, November 9, 2025

*— Hill Smith*
*hill.smith.books@gmail.com*

# Preface

This book is a practical, hands-on guide to Forex trading aimed at traders building automated systems in Python. It focuses on actionable technical-analysis concepts, common tactical patterns, traps to avoid, and reproducible programmatic strategies using OHLCV data. Material is ordered from fundamentals to advanced topics so you can build progressively: definitions and market structure first, trading mechanics next, then pattern recognition, risk controls, backtesting, implementation, deployment, and advanced techniques.

Though focus is on Forex most principles are also relevant to stock market. If you are not interested in automated trading systems or programming If you're not interested in automated trading systems or programming, you'll probably find most of the chapters useful and informative anyway.

Every chapter covers a different aspect or topic of trading. I strived to make every chapter concise and expressive to cover most aspects and to keep the length of this book moderate.

*If you are new to Forex trading you may want to examine the Glossary chapter first.*

Intended use:

- As a learning roadmap for traders moving from manual to automated strategies.
- As a reference handbook to translate trading ideas into code, backtests, and live rules.
- As a checklist of risks, costs, and operational controls required for safe deployment.

Assumptions and scope:

- Reader has basic Python familiarity and access to OHLCV market data.
- Focus is on intraday-to-short-swing horizons (primary target: ~10-day trades), though many concepts apply across timeframes.
- Emphasis on technical and quantitative approaches; fundamentals and macro commentary are covered where they affect price mechanics and risk.
- Advanced topics include machine learning, co-integration/pairs trading, portfolio optimization, and best practices for robust model development.

How to use the chapters:

- Read sequentially for a structured learning path.
- Use the implementation and backtesting chapters when converting rules into code.
- Treat the sample rulesets and pseudocode as templates to adapt, not turnkey strategies—validate with your own data and risk limits.

Goals and disclaimers:

- Goal: give you the conceptual and practical toolkit to design, test, and operate algorithmic FX strategies on a 10-day horizon.
- Not financial advice. Perform your own due diligence, and use capital controls, paper trading, and staged deployment.

# How to Use This Book

Who this book is for:

- Traders building automated systems in Python with basic programming knowledge.
- Practitioners focused on 4H and 1D timeframes (daily/short-swing, ~10-day horizon).
- Readers who want practical, codable rules, backtesting methodology, and deployment guidance.

How to read (recommended path):

1. Read Chapters 3–6 (Definitions, Market Mechanics, Orders, Timeframes) to ground vocabulary and structure.
2. Read Chapters 7–11 (Fundamentals, Technical Foundations, Patterns, Indicators) to build signal concepts.
3. Read Chapters 12–15 (Risk, Margin, Costs, Psychology) before designing strategies.
4. Study Chapter 16 (LFT vs HFT) to set realistic expectations and infrastructure needs.
5. Read Chapters 17–22 (Automated Bots, Backtesting, Implementation, Deployment, Failures) while translating ideas into code.
6. Read Chapters 23–26 (Market Comparisons, Regulation, 10-day Plan, Advanced Topics) for refinement and production readiness.
7. Use Appendix and sample code when implementing and testing.

How to use the examples and pseudocode:

- Treat examples as templates: change parameters, timezones, data feeds, and broker execution primitives to match your environment.
- Replace hard-coded assumptions (e.g., pip size, lot size, spread values) with values read from your data provider or broker API.
- Use the included pseudocode as a basis for unit tests, backtests, and paper-trading before live deployment.

Data, timeframe, and horizon conventions used in this book:

- Primary timeframes: 4-hour (4H) and daily (1D). Examples and parameter defaults will be oriented to these.
- Trade horizon: typical position duration ~1–10 days; entries and exits optimized for multi-day moves rather than tick-by-tick scalping.
- OHLCV series: candle times follow UTC by default; adjust for broker/server timezone and DST in your code.

Code and reproducibility workflow (practical checklist):

1. Define strategy hypothesis in plain language (trigger, entry, stop, target, sizing).
2. Collect and normalize OHLCV data (UTC, consistent candle close convention).

3. Implement deterministic signal generator and position logic (no randomized steps).
4. Backtest with realistic transaction-cost model (spread, commission, slippage, rollover).
5. Do walk-forward or rolling-window validation; keep out-of-sample data.
6. Paper-trade with the same execution code and broker API.
7. Deploy gradually with size limits, monitoring, and kill-switches.

A short list of files/folders you should keep per strategy:

- data/ (raw and normalized OHLCV, metadata)
- src/
    - signals.py
    - risk.py
    - execution.py
    - backtest.py
- tests/ (unit tests for signal logic, edge cases)
- notebooks/ (exploratory analysis)
- logs/ (trade logs, P&L, alerts)
- config/ (broker credentials, instrument definitions) — keep secrets out of repo

Notation and parameter defaults used in examples:

- ATR refers to 14-period ATR on the same timeframe unless otherwise noted.
- “pip” size uses the instrument convention (e.g., 0.0001 for EURUSD).
- Risk per trade default: 0.5% of account equity (configurable).
- Position sizing examples use fixed-fraction and volatility-adjusted sizing.

How to adapt material to your broker and jurisdiction:

- Map instrument symbols and contract sizes to broker specifics (e.g., OANDA vs Interactive Brokers).
- Confirm commission model, spread reporting, and rollover computations.
- Implement KYC and tax compliance as required.

# Basic Definitions

## What is Forex?

The foreign exchange (Forex, FX) market is the global, decentralized marketplace for trading currencies. Trades are always in pairs (one currency bought, one sold). Forex is primarily OTC (over-the-counter) with liquidity provided by banks, brokers, and electronic venues.

## Currency pair, base vs quote

- Currency pair: A/B (e.g., EUR/USD).
- Base currency: A (first) — unit you buy/sell.
- Quote currency: B (second) — currency used to price the base.
- A price of 1.1200 for EUR/USD means 1 EUR = 1.1200 USD.

## Major, minor, exotic pairs

- Majors: pairs with USD and highly liquid crosses (EUR/USD, USD/JPY, GBP/USD, USD/CHF, AUD/USD, USD/CAD, NZD/USD).
- Minors/crosses: no USD (EUR/GBP, EUR/JPY).
- Exotics: one major currency paired with an emerging market currency (USD/TRY, EUR/TRY) — wider spreads, lower liquidity.

## Pip, pipette, tick

- Pip: typical smallest price increment for quoting (often 0.0001 for 4-decimals pairs; 0.01 for JPY pairs).
- Pipette: one fractional pip (e.g., 0.00001).
- Tick: smallest change in price provided by feed (broker dependent).

## Lots and contract sizes

- Standard lot: 100,000 units of base currency.
- Mini lot: 10,000.
- Micro lot: 1,000.
- Some brokers allow fractional lots (e.g., 0.01). Always confirm contract specification per broker.

## Leverage, margin, and buying power

- Leverage: ratio of notional exposure to account equity (e.g., 50:1).
- Margin: amount of equity required to open/maintain a position (used margin).
- Free margin = equity − used margin. Excessive leverage increases volatility of account equity.

## Market participants and liquidity providers

- Commercial banks and prime brokers (largest liquidity providers).
- Non-bank liquidity providers and ECNs.
- Hedge funds, proprietary trading firms, institutional investors.
- Retail brokers and individual traders (use ECN/market-maker access).
- Corporates (FX hedging), central banks (policy and interventions).

## Market hours and sessions

- Forex runs 24/5: Sunday evening (GMT) open → Friday close.
- Major sessions: Asian (Tokyo/Singapore), European (London), North American (New York).
- Overlap periods (London/New York) typically show highest liquidity and volume.

## Price formation and the order book (OTC vs ECN)

- OTC: prices are quoted by dealers and executed bilaterally; order flow and liquidity are fragmented.
- ECN/STP: electronic networks aggregate orders and present consolidated prices closer to true market supply/demand; may have tighter spreads and deeper visible liquidity.
- Order book depth for FX retail data is often limited compared to centralized exchange order books.

## Bid/Ask and spread

- Bid: price at which market buys base (you sell).
- Ask (offer): price at which market sells base (you buy).
- Spread = Ask − Bid. Primary implicit cost; varies by pair, time, and provider.

## Tick data vs candle (OHLC) data

- Tick data: every quote/trade; required for precise execution/backtesting where entry timing matters.
- OHLC (Open, High, Low, Close) candles: aggregated intervals (e.g., 4H, 1D). Primary data used for 4H/1D strategies and much easier to handle.
- Volume in FX: exchange volume is not centralized; many brokers provide tick/quote count or aggregated executed volume proxies — interpret with caution.

## Liquidity, slippage, and market impact

- Liquidity: how easily a position can be opened/closed without large price movement. Majors have highest liquidity.
- Slippage: difference between expected execution price and actual fill. More common during news/events, low liquidity hours, or large order sizes.

- Market impact: for large orders, your execution pushes price; retail traders typically face minimal impact on majors but must model it for backtests.

## Typical volatility characteristics

- FX volatility varies by pair and session: JPY crosses often have lower pips but can move significantly in percent terms; exotic pairs can be highly erratic.
- Volatility also clusters in time — use ATR or volatility filters when sizing and timing trades.

## Tick conventions, quoting formats, and decimals

- Most FX pairs use 5 decimals (pipette) in retail feeds (EUR/USD 1.12345). JPY pairs often use 3 decimals (e.g., USD/JPY 110.123).
- When coding, normalize pip size and decimal handling per instrument to avoid rounding errors.

## Key takeaways for 4H and 1D trading

- Use 4H and daily candles as primary signals; avoid building strategies that require tick-level timing.
- Base position sizing and stop placement on volatility (ATR) measured on the same timeframe.
- Prefer major pairs for consistent liquidity and tighter spreads.
- Always model spreads, slippage, and rollover in backtests for realistic performance.

# Market Mechanics

## Order types and execution behavior

- Market order: immediate execution at best available price; subject to slippage during fast moves or thin liquidity.
- Limit order: executes at specified price or better; may not fill (partial fills possible).
- Stop order (stop-loss/stop-entry): becomes a market order when stop price is hit (subject to slippage).
- Stop-limit: becomes a limit order when triggered — avoids worst slippage but may miss fills.
- OCO (one-cancels-other): pair stop and target; execution of one cancels the other.
- Time-in-force: GTC (good-til-cancel), IOC (immediate-or-cancel), FOK (fill-or-kill) — broker support varies.

## Partial fills, requotes, and fill quality

- Partial fills: order partially matched due to available liquidity; remainder may be canceled or wait.
- Requotes: broker returns a new price before filling — more common with market-makers during volatile times (less so with ECNs).
- Fill quality: measure in backtests by modeling realistic slippage distributions and rejected orders.

## Slippage modeling

- Historical slippage depends on volatility, liquidity, and order size. For 4H/1D strategies:
    - Use a fixed slippage buffer (e.g., half the spread or X pips) as baseline.
    - Add event-driven slippage: larger slippage during economic releases or illiquid sessions.
    - Model slippage stochastically (e.g., normal with mean = 0, sigma tuned) if you have tick/level data.

## Spread dynamics and spread modeling

- Spreads widen during low liquidity and news; some brokers have variable spreads.
- Backtest spread as:
    - Tick-by-tick mid-price with ask/bid adjustment per trade, or
    - Candle-based: assume opening/closing trade occurs at worst side price (add half-spread to buys, subtract half-spread from sells), or
    - Use historical bid/ask series if available.

## Rollover/swap (overnight financing)

- Rollover: interest differential charged/credited for positions kept past the rollover hour (usually 17:00 NY for many brokers).
- Swap rate = (short rate − long rate)/365 * notional + broker markup.
- For a 10-day horizon, cumulative rollover can be meaningful; include in backtests and P&L calculations.

## Trade lifecycle and position maintenance

1. Signal generation (entry condition).
2. Order submission (limit/market/stop).
3. Execution/fill (partial/full).
4. Trade monitoring (manage stop, trailing stop, scaling).
5. Exit (target, stop, time stop, or manual).
6. Post-trade logging and performance attribution.

## Slippage, latency, and market microstructure effects

- Latency matters less for 4H/1D strategies but still affects order fill timing around session opens and economic releases.
- Use limit orders to improve price if you can tolerate fill probability tradeoffs.
- Monitor and log execution timestamps vs candle close to detect system-induced delays.

## Liquidity constraints and size limits

- Retail accounts often have size limits (lot increments) and different margin rules across brokers.
- Large notional trades may cause partial fills or require working orders across time — not typical for retail-sized 4H/1D systems but plan for scaled sizing.

## Stop placement mechanics and stop hunting

- Stops placed at round numbers or obvious technical levels can be targeted during thin liquidity or by algo flow; mitigate by:
    - Placing stops beyond local volatility (e.g., multiple ATR).
    - Using mental stops + limit exit orders or time-based exits where appropriate.
    - Adding buffer in pip terms to avoid being stopped on noise.

## Hidden costs to account for

- Slippage and spread (explicit in each trade).
- Swap/rollover (for multi-day holds).
- Commission (per trade or per million notional).
- Market data subscription and latency costs.
- Funding/withdrawal fees and taxes.

## Best practices for 4H/1D automated trading

- Align signal generation to candle closes (e.g., submit at candle close market/open rather than during candle) to avoid intra-candle ambiguity.
- Use volatility-adjusted stops (ATR) and position sizing to maintain consistent risk.
- Handle holidays and thin sessions by reducing size or skipping signals.
- Always simulate fills with bid/ask modeling and include swap accruals for overnight positions.

# Orders

This chapter gives practical rules and pseudocode-ready logic for submitting, managing, and reconciling orders for 4H/1D automated FX trading. Focus is on deterministic behavior, realistic fills, and safety.

## Concrete order types and when to use them

- Market Order
  - Use: urgent entry/exit at candle close when fill certainty is required (e.g., end-of-day rebalancing).
  - Risk: slippage during news/low liquidity.
- Limit Order
  - Use: improve entry/exit price; implement patience-based entries (buy dips).
  - Risk: may never fill.
- Stop Order (market on trigger)
  - Use: breakout entries or protective stops.
  - Risk: slippage when triggered; during spikes you may get worse price.
- Stop-Limit
  - Use: reduce worst-case execution price when stop-triggered; avoid when market can gap through.
  - Risk: may not fill, leaving you exposed.
- OCO (One-Cancels-Other)
  - Use: pair profit target and stop; ensure broker supports atomic OCO or implement locally with careful race handling.

## Deterministic execution rules for 4H/1D systems

- Evaluate signals only at candle close times (e.g., at 00:00 UTC daily or every 4th hour for 4H). Use the closed candle's OHLC to compute indicators and decide.
- Choose consistent submission timing:
  - Immediately at candle close: submit orders referencing next available market state.
  - Or delay to next session open if you want to avoid thin overnight fills.
- Prefer limit entries for better average price; fall back to market if time-based entry window expires.

## Suggested safe defaults (configurable)

- Max slippage tolerance: 1–3 pips for majors (adjust per pair). If market order would exceed tolerance, cancel and retry or skip.
- Retry policy: attempt up to N=2 additional submissions spaced by T=1–5 seconds (mostly relevant for low-latency infra; optional for retail).

- Partial fill handling: if partial and remaining volume is below minimum lot, cancel remainder. If partial and you still want exposure, decide whether to scale risk or attempt to fill again.
- Fill-at-open rule for daily: treat next day open fill as market price and model using open price adjusted for spread/slippage.

### Pseudocode: Order submission and confirmation (deterministic)

```
# Inputs: side, size_lots, order_type, price (for limit/stop), max_retry
attempt = 0
while attempt <= max_retry:
    order_id = broker.submit_order(side, size_lots, order_type, price)
    sleep(ack_wait)  # small wait to allow broker ack
    status = broker.get_order_status(order_id)
    if status == 'FILLED':
        record_fill(broker.get_fill(order_id))
        break
    if status in ['PARTIAL_FILLED']:
        record_partial_fill(...)
        if remaining_lots < min_lot:
            broker.cancel(order_id)
            break
        # decide to continue trying or cancel
    if status in ['REJECTED','CANCELLED']:
        handle_reject(status_reason)
        break
    attempt += 1
    # optional backoff before retry
    sleep(retry_backoff)
if attempt > max_retry and status != 'FILLED':
    log("Unfilled order; take fallback action")
```

### Stop/target maintenance logic (local vs broker-managed)

- Broker-managed stops: rely on broker OCO/stop architecture for guaranteed off-exchange stops (but check reliability and requote behavior).
- Locally-managed stops: keep logic in your system; requires actively monitoring price feed and submitting market/limit orders when stop condition is met. Safer for custom behavior but riskier during system outage.
- Recommended: use broker OCO for simplicity and redundancy; mirror stops locally for auditing and reconciling fills.

## Order cancellation and race conditions

- When implementing OCO at the client level, there is a race between filling target or stop and canceling the other. Use these practices:
    - On receiving a fill event, immediately issue a cancel for the opposite order and wait for cancel confirmation.
    - If cancel fails, log and escalate; implement idempotent handlers so repeated events don’t create inconsistent state.
    - Use unique client-order-ids to map broker fills to local positions.

## Handling trading halts, market closures, and reconnects

- On reconnect after data or execution outage:
    - Reconcile positions by querying broker account and open orders.
    - Rebuild internal state from broker snapshots; do not trust local memory alone.
- If broker disconnects mid-trade:
    - Assume orders may have been partially filled—reconcile on reconnect.
    - Implement a “fail-safe” that reduces exposure if you detect inability to manage stops (e.g., close a subset of positions where safe).
- For market closures (weekend gaps), avoid opening new positions within X hours of close (e.g., 1 hour before Sunday open) unless strategy explicitly trades weekend gaps.

## Slippage, spread and worst-case fills in code

- When simulating fills in backtests, apply:
    - For limit buy: executed_price = min(limit_price, best_ask_at_fill) — but if modeling worst-case, assume execution at ask + slippage.
    - For market buy: executed_price = ask + modeled_slippage.
    - For sells invert symmetrically.
- Example deterministic adjustment:
    - executed_price = raw_price + sign * (half_spread + fixed_slippage)
    - where half_spread and fixed_slippage are per-instrument.

## Example: safe entry workflow for a breakout (4H)

1. Compute breakout condition at 4H candle close.
2. Place a stop-entry order above breakout level (stop becomes market) sized by volatility-based sizing.
3. Place stop-loss at breakout_level − k * ATR (k e.g., 1.5–2.5).
4. Place profit target at R:R e.g., 1.5–3x expected risk OR use trailing ATR-based exit.
5. If order not filled within next candle, cancel entry and re-evaluate on subsequent close.

### Trade state machine (recommended)

- States: IDLE → SIGNALED → ORDER_SUBMITTED → PARTIAL_FILLED → FILLED → MANAGED → EXIT_SUBMITTED → CLOSED → RECONCILE
- Implement explicit handlers for transitions and idempotency so repeated callbacks/events don’t corrupt state.

### Logging and auditing

- Log every broker interaction: submit, ack, fill event, cancelled, rejection, and timestamps.
- Maintain audit trail linking each P&L line to the signal that generated it and to exact order messages.
- Store raw market snapshot at submit time (best bid/ask, spread) for later analysis.

### Testing order logic

- Unit tests: mock broker responses to validate state machine behavior for fills, partials, rejects.
- Integration tests: paper trading account to verify real-world behavior (OCO support, stop reliability).
- Backtests: include execution model (spread, slippage, partial fills) and validate that simulated fill rates match paper/live fills historically.

### Quick checklist before live trading

- Confirm instrument pip/lot semantics with broker.
- Set min/max lot constraints in execution module.
- Validate timezone handling for candle close vs broker execution times.
- Test stop and OCO behavior on paper account.
- Implement kill switch to close all positions and stop trading on critical failures.

# Timeframes Considerations

## Timeframe choices for 4H and 1D trading

- 4H: captures intermediate swings, multiple signals per week, suitable for scaling in/out within the 10-day horizon.
- 1D (daily): primary filter for trend direction and volatility; aligns with daily risk resets and rollover timing.
- Use 4H for entry/management nuance and 1D for trend/context (multi-timeframe approach).

## Candle conventions and timing

- Use UTC standardized candle closes in your data pipeline. Convert broker timestamps to UTC to avoid DST/locale errors.
- For daily strategies, decide whether “day” means calendar day or trading day relative to broker rollover (many brokers use 17:00 New York). Use consistent convention across data, backtests, and live.

## Multi-timeframe rules (practical pattern)

- Define trend on 1D (e.g., 20-EMA slope or 50 SMA): only take direction-aligned trades on 4H.
- Use 4H to time entries when daily shows high-probability set-ups.
- Example rule: enter long on 4H signal only if daily close > 20-EMA and 1D ATR < threshold.

## Signal cadence and expected trade frequency

- Expect fewer signals than intraday scalping: a 4H/1D system typically yields several trades per pair per month. For a 10-day holding target, expect 1–3 trades per pair per month depending on rules.
- Keep a basket of pairs (3–8 majors/crosses) to increase activity while controlling correlation.

## Stop placement and sizing for multi-day holds

- Use volatility-based stops (ATR on the trading timeframe or daily ATR for 4H entries) to account for normal price movement.
- Common rules:
    - ATR(14, 1D) * 1.5–3.0 for daily-based stop when holding several days.
    - For 4H entries, ATR(14, 4H) * 1.5–2.5.
- Position sizing: risk fixed fraction of equity (e.g., 0.25%–1% per trade) and compute lot size from pip-risk = stop_pips * lot_value_per_pip.

### Time stop (max holding period) for ~10-day horizon

- Implement a time stop to exit stale trades: e.g., if trade duration > 10–14 calendar days without reaching stop/target, exit at market (or tighten stop). This limits regime drift and reduces overnight swap exposure.

### Entry timing relative to sessions

- Avoid initiating new positions right before major session overlaps close or before major economic releases. Prefer entries at:
  - Start of London session (increased liquidity) for EUR/GBP/EURUSD moves.
  - Start of New York session when news flows matter for USD pairs.
- For daily systems, placing orders immediately after daily close (or at next open) reduces intra-day execution ambiguity.

### Handling economic events on 4H/1D strategies

- Major releases can cause multi-day moves and widened spreads; options:
  - Skip new entries if a high-impact event is within X hours (e.g., 4–12 hours) of entry.
  - Reduce position size around event windows.
  - Use event-aware stop widening (increase ATR multiplier temporarily) — be conservative.

### Rollover/swap considerations for multi-day holding

- Compute expected swap per night and aggregate over expected hold duration. If swap cost materially changes edge, adjust strategy or prefer pairs with favorable or negligible swap.
- Carry trades: for positions held multiple days, positive carry can add to edge but introduces interest-rate risk.

### Volatility regimes and adaptive parameters

- Use regime detection (ATR percentile, realized volatility) to adapt:
  - During low volatility: tighten stops and increase position sizing modestly to keep expected edge.
  - During high volatility: widen stops, reduce size, or increase min-volatility filters to avoid noise-driven exits.
- Example adaptive rule: lot_size = base_size * (target_atr / current_atr) capped within [0.5x, 2x].

## Correlation and portfolio construction for multiple pairs

- FX pairs can be highly correlated (EUR/USD and EUR/GBP share EUR exposure). For a 4H/1D portfolio:
  - Limit aggregate directional exposure by currency (e.g., max net EUR exposure).
  - Use equal risk per trade instead of equal notional to control portfolio-level drawdown.
  - Consider hedged pairs or neutral baskets to reduce single-currency concentration.

## Expected P&L profile for 4H/1D, 10-day target strategies

- Longer stops than intraday scalpers → lower win rate but higher average win/loss magnitude.
- Expect wider drawdowns but smoother trade noise; focus on expectancy (avg_win * win_rate − avg_loss * loss_rate) and trade duration distribution.
- Track time-weighted and trade-weighted metrics (e.g., return per day exposed).

## Practical monitoring cadence

- Daily review: reconcile fills, check open positions against rules, monitor economic calendar.
- 4H checks: automated health checks for slippage, spread spikes, and position management triggers.
- Weekly review: performance, parameter drift, recent trades audit.

## Implementation tips specific to 4H/1D

- Backtest and validate using daily and 4H candles aligned to same UTC convention.
- When using 4H signals filtered by daily trend, anchor stop to ATR measured on the daily series to better match holding horizon.
- Prefer robust entry filters (e.g., volatility percentile, momentum) to avoid whipsaw in sideways markets.

## Quick checklist for a 10-day (4H/1D) strategy

- Define timeframe mapping: signal TF = 4H, trend TF = 1D.
- Set ATR-based stop on 1D for multi-day durability, compute lot size from risk %.
- Implement time stop at 10–14 days.
- Model spread, slippage, and rollover in backtest.
- Limit correlated exposure and monitor aggregate currency risk.
- Skip entries near high-impact calendar events or reduce size.

# Fundamental Analysis for Forex

## Why fundamentals matter for 4H/1D (10-day) trading

- Macro events and central-bank moves create multi-day trends or regime shifts; they often drive the moves you catch on a 4H/1D horizon.
- Use fundamentals primarily as a filter (bias/avoid) and for sizing/holding decisions rather than precise entry timing.

## Core macro drivers

- Interest rates and expectations: the dominant long-term driver. Rate differentials influence carry and directional pressure.
- Inflation: influences central bank policy and real rates; unexpected surprises move FX.
- Growth (GDP, PMI, employment): shifts growth expectations and risk sentiment.
- Trade balances and capital flows: intervening factor over medium-term horizons.
- Fiscal policy, political risk, and commodity prices: especially relevant for commodity-currency pairs (AUD, CAD, NZD).

## Central bank policy and forward guidance

- Central bank statements, minutes, and forward guidance move expectations; rate decision dates are high-impact.
- Watch policy-reaction function: how the central bank reacts to inflation vs growth. Changes in guidance can alter trend direction for days to weeks.

## Economic calendar and event typology

- High-impact scheduled events: central bank rate decisions, NFP (US employment), CPI releases, GDP prints.
- Medium/low impact: retail sales, PMI flash, housing data.
- Unscheduled events: geopolitical shocks, natural disasters — model tail risk (wider spreads, large slippage).
- For 4H/1D systems: mark events as ±X hours/days and apply rules (skip entries, hedge, reduce size).

## Practical rules for event handling

- Pre-event: avoid opening new positions within a buffer (e.g., 4–24 hours) for high-impact events unless strategy is event-driven.
- Post-event: allow a cooling window (e.g., wait 1–2 candles on 4H or 1 daily candle) before acting on new signals to avoid whipsaw.

- Event-driven strategies: explicitly model widened spread and slippage; use smaller size or straddle entries with defined stop.

### Macro sentiment and risk-on/risk-off

- Risk assets correlation: USD often strengthens in risk-off; commodity currencies weaken. Monitor VIX proxies or equity futures for global risk bias.
- For 10-day trades, align entries with prevailing risk regime where appropriate (e.g., avoid long AUD in extreme risk-off).

### Data sources and monitoring

- Reliable economic calendar (filtered by pair relevance and impact).
- Central bank feeds and official policy statements.
- Market expectations (swap/implied rates, futures) to gauge priced-in moves.
- News aggregators and sentiment feeds for unscheduled events.

### Incorporating fundamentals into systematic rules

- Bias filter: only take long trades for currency X if interest-rate differential trend is favorable or expected to be stable for your holding period.
- Volatility adjustment: widen stops or reduce size around anticipated macro risk windows.
- Carry overlay: for multi-day holdings, incorporate expected swap into expected return (net of financing).
- Macro regime features: create binary/regime features (hawkish/dovish; growth strong/weak) and use them in strategy selection or weighting.

### Practical examples (rule-level)

- Example 1 — Rate-differential filter:
  - If 3-month forward rate differential for EURUSD > threshold and expected to remain, allow larger position sizes on EUR longs.
- Example 2 — Event blackout:
  - If any H1 (high-impact) event for USD within next 12 hours, skip USD pair entries.
- Example 3 — Post-news confirmation:
  - After CPI release, wait for one daily close post-release; only trade in direction confirmed by daily close and ATR expansion.

### Fundamental signals to code (features)

- Forward rate differential (implied from swaps) normalized to ATR.
- Surprise metric: actual − consensus for key releases, normalized by historical surprise volatility.

- Central bank sentiment score: mapped from meeting minutes and speech sentiment (simple scoring).
- Risk sentiment proxy: equity futures return over last 1–5 days.

## Limitations and cautions

- Fundamentals are noisy on short horizons; market prices often “price in” expectations well ahead.
- Correlation between macro releases and FX moves is variable; always combine with price-action confirmation.
- Avoid overfitting to specific event outcomes; treat fundamental inputs as regime signals, not precise triggers.

## Quick checklist for integrating fundamentals into 4H/1D systems

- Subscribe to a reliable economic calendar and map event impact to pairs.
- Implement event blackout and post-event cooldown rules.
- Compute expected swap and include in backtest P&L for multi-day trades.
- Create simple macro features and test their incremental predictive power in walk-forward validation.

# Technical Analysis Foundations

This chapter covers the core price-action building blocks you'll use to design 4H/1D strategies: support/resistance, trends, chart types, candlestick interpretation, and volume caveats in FX.

## Support and resistance (S/R)

- Definition: S = price area where buyers historically enter; R = area where sellers historically enter. Use zones, not single lines.
- Construction methods:
  - Swing highs/lows on 1D for major zones; refine with 4H for entry precision.
  - Round numbers (psychological levels) and previous session opens/close levels.
  - Moving averages (20/50/200 MA) can act as dynamic S/R.
- Practical rules:
  - Treat S/R as zones (±X pips or ATR fraction) and require price action confirmation before entering.
  - Use higher timeframe S/R to filter 4H signals (only trade breaks/holds aligned with daily S/R).

## Trends, trendlines, and channels

- Trend definition: series of higher highs/higher lows (uptrend) or lower lows/lower highs (downtrend) on 1D.
- Trendlines: connect at least two significant swing points; better validated with a third touch.
- Channels: parallel trendlines capturing price oscillation—use for mean-reversion within channel or breakout strategies at channel breach.
- Practical rules:
  - Prefer trend confirmation on 1D for direction bias; time entries on 4H when price respects trendline or bounces at channel support/resistance.
  - Avoid trading against an established daily trend unless contrarian setup has strong signal and tight risk.

## Chart types and candle basics

- Candlestick advantages: show open, high, low, close; provide intra-candle bias (e.g., long wick indicates rejection).
- Heikin-Ashi: smooths candles to emphasize trend; helpful for trailing exits but should not replace price-level stops.
- Bar/line charts: useful for simplifying structure; line charts (close only) reduce noise for swing identification.

- Practical usage:
    - Use standard candles for entries and wick analysis on 4H; use daily close patterns for trend decisions.

## Candlestick patterns and price action context

- Single-bar: pin bar (long wick rejection), marubozu (strong directional candle).
- Multi-bar: engulfing, inside bar (consolidation), morning/evening star (reversal).
- Context matters: a bullish engulfing at daily support is more meaningful than the same pattern in a trendless zone.
- Coding note: require patterns to meet size thresholds (e.g., body > X% of ATR) to avoid noise.

## Breakouts vs fakeouts

- True breakout: price breaks S/R with follow-through (volume/ATR expansion, next candle closes beyond).
- Fakeout: quick breach followed by reversal; common around round numbers and thin sessions.
- Mitigations:
    - Use volatility filter: require ATR expansion or momentum confirmation post-break.
    - Use retest entries: wait for price to retest breakout level and show support/confirmation on 4H before entering.
    - Use confirmation candle close on daily for higher conviction.

## Moving averages and dynamic filters

- Common MAs: EMA(20), SMA(50), SMA(200). EMA reacts faster; SMA smoother.
- Uses:
    - Trend filter: only trade direction when price above/below chosen MA.
    - Crossover systems: simple but prone to whipsaw—apply volatility/time filters for 4H/1D.
- Practical default: use 20 EMA on daily for trend, 50 SMA as medium-term bias.

## Momentum and breakout confirmation

- Momentum indicators (RSI, MACD) confirm strength of moves; prefer to use them as filters rather than primary triggers.
- Example rule: require RSI(14, daily) > 55 for long trades aligned with trend; avoid long when RSI < 45.

## Volume in Forex: caveats and proxies

- FX has no centralized traded volume; broker volume is a partial proxy and not comparable across providers.

- Use tick-count volume (number of price updates) or broker-provided executed volume cautiously; better to rely on price/volatility signals for order timing.
- When available, volume spikes aligned with breakout can increase confidence, but do not over-weight volume signals.

### Price structure concepts (higher timeframe first)

- Market structure: sequence of swing highs/lows establishes bias. Detect break of structure (BoS) on daily to flip bias.
- Order blocks and liquidity pools: institutional interest zones often near previous consolidation/high-liquidity areas—use as potential targets or stops.
- Fair value and mean reversion zones: extreme deviations measured in ATR or standard deviations can be candidates for mean-reversion trades when confirmed by structure.

### Volatility measures and their use

- ATR (Average True Range): preferred for volatility-adjusted stops and sizing. Use ATR(14, 1D) for daily stop baselines.
- Volatility percentile: avoid strategies when ATR percentile is below a threshold (too quiet) or above (too noisy) depending on your edge.
- Use ATR to convert pip risk to lot sizing: lot = (risk_amount / (stop_pips * pip_value_per_lot)).

### False signals and noise reduction

- Use multi-timeframe confirmation: require daily trend + 4H trigger.
- Require minimum pattern strength relative to ATR (pattern_size > k * ATR).
- Add time filters: avoid trading around market open/close and during major economic events.

### Practical coding tips

- Normalize price series to consistent timezone and tick convention before computing indicators.
- Implement functions to detect swings and label highs/lows with lookback windows (e.g., local_max/min over N bars).
- Encapsulate S/R zone creation: cluster swing levels within X pips into single zone to avoid duplicate signals.
- Make indicator parameters configurable and perform walk-forward tuning rather than optimizing on whole sample.

### Quick checklist for building price-action rules (4H/1D)

- Define higher-timeframe trend rule (daily).
- Identify daily S/R zones and key levels.

- Use 4H for entry: require pattern or retest with ATR-based size/stop.
- Filter signals by momentum or ATR percentile.
- Backtest with spread/slippage and include rollover for multi-day holds.

# Candlestick Patterns

This chapter lists practical candlestick and price-action patterns useful for 4H/1D systems, shows how to code them deterministically, and gives guidance on context, sizing, and filtering.

## General principles for pattern use

- Use patterns as conditional signals, not standalone guarantees. Require higher-timeframe confirmation (daily) and volatility thresholds (ATR).
- Quantify patterns with measurable numeric rules (no visual-only criteria). Include minimum body size relative to ATR and wick ratios.
- Prefer patterns that indicate rejection or momentum (pin bars, engulfing) for entries; use inside bars and ranges for breakout setups with retest logic.

## Single-bar patterns

- Pin Bar (hammer/inverted hammer/shooting star)
  - Definition: small body with long tail/wick opposite direction of expected move.
  - Coding heuristic:
    - body_size <= 0.3 * candle_range
    - tail_len >= 0.6 * candle_range
    - tail_len >= k * ATR (e.g., 0.25 * ATR)
  - Use: reversal or rejection at S/R; enter on break of the candle high (for bullish) or low (for bearish) on 4H with daily support.
- Marubozu
  - Definition: candle with little/no wick — strong momentum.
  - Coding heuristic:
    - body_size >= 0.9 * candle_range
    - body_size >= k * ATR
  - Use: momentum continuation; consider scaling out with trailing ATR.

## Two-bar and three-bar patterns

- Engulfing (bullish/bearish)
  - Definition: second candle body fully engulfs previous candle body and closes in direction of engulf.
  - Coding heuristic:
    - abs(body2) > abs(body1) and body2 closes beyond open of candle1
    - body_size2 >= m * ATR (filter small candles)
  - Use: reversal at daily S/R or trend continuation when aligned with daily momentum.
- Inside Bar (IB)
  - Definition: candle fully within the range of previous candle (mother bar).

    - Coding heuristic:
        - high_IB < high_mother and low_IB > low_mother
        - mother_bar_range >= n * ATR (use larger mother range for relevance)
    - Use: breakout strategy — place stop orders beyond mother bar with confirmation retest on 4H or daily.
- Tweezer Tops/Bottoms
    - Definition: two candles with similar highs (tops) or lows (bottoms), indicating rejection.
    - Coding heuristic:
        - abs(high1 - high2) <= delta_pips or <= pct_of_ATR
    - Use: short-lived reversal signals; require daily confirmation.

## Multi-bar structures and sequences

- Morning/Evening Star
    - Three-bar reversal: large candle, small indecision candle (star), then large opposite candle.
    - Coding heuristic:
        - body1 large (>= q * ATR), body2 small, body3 large opposite and closes into body1
    - Use: higher timeframe reversals; enter on confirmation of third candle close or retest.
- Three White Soldiers / Black Crows
    - Sequence of 3 consecutive strong candles in same direction — strong continuation.
    - Coding heuristic:
        - three consecutive candles with bodies > r * ATR, each closing higher/lower than previous open.
    - Use: trend-following entries with ATR-based stop.

## Breakout and retest patterns

- Breakout + Retest (preferred for false-break reduction)
    - Rule:
        - Price closes beyond S/R on 4H or daily (breakout).
        - Wait for price to retest level and form confirmation candle (pin bar, engulfing) on retest.
        - Enter on confirmation candle close or on breakout continuation with ATR filter.
    - Coding heuristic: require retest candle to have body opposite side of breakout and body size > s * ATR.
- False breakout detection
    - If breakout occurs but next candle closes back inside range, treat as fakeout; consider contrarian entry if supported by momentum divergence.

## Pattern filters and numeric thresholds (defaults)

- Minimum body size: >= 0.3 * ATR (to avoid noise) for significance.
- Wick/body ratio: use thresholds like tail >= 0.6 * candle_range for pin bars.
- Confirmation candle: require close beyond a reference by at least t pips or > 0.25 * ATR.
- Use lookback to avoid nearby conflicting levels (no trade if within X pips of daily open/close).

## Coding examples (pseudocode snippets)

- Detect bullish engulfing (4H):

```
is_bull_engulf = (close[t] > open[t]) and \
                 (close[t-1] < open[t-1]) and \
                 (open[t] < close[t-1]) and \
                 (close[t] > open[t-1]) and \
                 (abs(close[t]-open[t]) >= 0.3*ATR)
```

- Pin bar bullish detection (4H):

```
candle_range = high[t] - low[t]
body = abs(close[t] - open[t])
upper_wick = high[t] - max(open[t], close[t])
lower_wick = min(open[t], close[t]) - low[t]
is_bull_pin = (lower_wick >= 0.6*candle_range) and (body <= 0.3*candle_rang
```

## Trade execution templates using patterns

- Retest pin bar long:
    1. Identify daily support zone.
    2. On 4H, detect bullish pin bar whose wick touches support zone.
    3. Place entry limit at pin bar high (or market on close), stop at support − 0.5*ATR, size by risk %.
    4. Target 1.5–3x risk or use trailing ATR exit.
- Engulfing breakout continuation:
    1. Detect breakout above daily resistance (daily close > resistance).
    2. On 4H an engulfing candle forms in breakout direction with body > 0.5*ATR.
    3. Enter market on close, stop at daily support or ATR multiple.

## Handling ambiguous or conflicting patterns

- When multiple pattern signals conflict (e.g., bullish pin bar but bearish engulfing same candle set), prioritize higher-timeframe signal or the pattern with stronger numeric measures (larger body relative to ATR).
- If pattern occurs within a tight range and ATR low, require additional confirmation (momentum indicator or volume proxy).

## Backtesting tips for patterns

- Use out-of-sample breaks and walk-forward testing to validate patterns. Pattern performance decays if overfitted to a specific lookback.
- Ensure no lookahead: compute pattern at candle close and only use data up to that timestamp for decision.
- Include transaction costs and slippage—many pattern edges vanish after realistic costs.
- Check pattern frequency: rare patterns may have high expectancy but insufficient trade count; balance statistical significance against edge.

## Practical parameter tuning (defaults for 4H/1D)

- ATR period: 14 on same timeframe (4H) or daily ATR for stops.
- Min body threshold: 0.3 * ATR.
- Pin bar tail threshold: 0.6 * candle_range.
- Max distance from S/R for pattern acceptance: 0.5 * ATR.

## Example sanity checks after backtesting

- Verify pattern hit rate and average return per trade.
- Check time-to-exit distribution and max drawdown caused by pattern trades.
- Analyze equity curve stability across market regimes and sessions.

# Structural Setups

Chapter summary: provides deterministic, codable definitions for common chart patterns, entry/exit/stop/target rules, ATR-relative thresholds, pseudocode and backtest setup (including spread/slippage/rollover). All rules assume candle-close decisioning on UTC 4H or 1D bars and default parameters: ATR(14), risk per trade = 0.5% equity, candle time = UTC, deterministic limit entries with fallback to market if not filled within N candles.

Table of contents:



Overview and pattern-use guidance:

- Use patterns as structural context + trade trigger; combine with trend / volatility filter (ATR, MA) and volume where available.
- For 4H/1D systems aimed at ~10-day horizons: prefer patterns that complete within 3–20 bars (4H) or 2–14 bars (1D).
- Require a minimum structural amplitude relative to ATR to avoid micro-noise signals.
- Deterministic rules: detect pattern at candle close, compute entry as limit at breakout/counterlevel, if not filled within max_wait_candles then cancel or use market entry at next candle open (configurable).

Pattern detection conventions and variables:

- Use high/low/close/open arrays indexed by increasing time.
- ATR = ATR(14) computed on same timeframe.
- candle_range = high - low
- body_size = abs(close - open)

- pivot detection: use n=3 or n=5 swing pivots (configurable). A pivot high at index i if high[i] > high[i-k:i) and high[i] >= high((i+1):(i+k+1)] with k=2 (5-bar pivot) unless otherwise noted.
- All thresholds expressed relative to ATR (default) or absolute ticks/pips if preferred.
- Min amplitude rule: pattern height >= 1.2 * ATR (configurable to 1.5 for stricter).
- Time limits: max bars between key pattern points to prevent multi-month structures (max_span = 30 for 4H by default, 120 for 1D by default — adjust for strategy).

1. Head & Shoulders (H&S) and Inverse H&S

Definition (codable):

- Three consecutive pivot highs (H&S) or pivot lows (Inverse) with middle peak (head) higher (lower) than shoulders.
- Neckline: linear fit across the two shoulder troughs/peaks; breakout is close crossing neckline. Detection thresholds
- Left_shoulder, head, right_shoulder pivots must satisfy: head_height >= shoulders_height + 0.6*ATR
- Shoulder heights within symmetry: abs(LS - RS) <= 0.6 * ATR
- Pattern height (head_peak - neckline_level) >= 1.2 * ATR
- Max bars between LS and RS <= max_span

Entry/Stop/Target (codable):

- Entry: place sell (H&S) or buy (Inverse) limit at breakout close below/above neckline on candle close. Prefer conservative entry: require close beyond neckline by gap = 0.1*ATR.
- Stop: above last shoulder by stop_padding = 0.6 * ATR (H&S) or below last shoulder for inverse.
- Target: measured move = head_peak - neckline -> project from breakout point. Take-profit levels: TP1 = 0.6*measured_move, TP2 = measured_move.
- Trail: move stop to breakeven after 0.6*measured_move; then use ATR(14)*1.0 trailing step.

Example parameters (defaults): - ATR_min_height = 1.2 * ATR - symmetry_tol = 0.6 * ATR - breakout_margin = 0.1 * ATR - stop_padding = 0.6 * ATR

1. Double Top / Double Bottom

Definition: Two pivots near same level separated by trough/peak; neckline = intervening trough/peak.

Detection thresholds:

- Peak separation: bars_between_peaks <= max_span
- Peak level difference: abs(P1 - P2) <= 0.6 * ATR
- Intervening trough depth >= 0.6 * ATR (ensures structure)

Entry/Stop/Target:

- Entry: on close below trough (double top) or above peak (double bottom) by breakout_margin = 0.1 * ATR.
- Stop: above the nearer peak/two-peak high + stop_padding = 0.6*ATR.
- Target: measured_move = average(P1,P2) - trough -> project measured_move from breakout; TP1 = 0.6*measured_move, TP2 = measured_move.
- Fail-safe: if price revisits peaks within X bars (X=3) after breakout, exit.

1. Triangles (Symmetrical / Ascending / Descending)

Definition: Formed by converging trendlines connecting at least two highs and two lows (pivots).

Detection thresholds:

- Convergence slope: distance between lines at left > distance at right by at least 20% of left distance (i.e., narrowing).
- Min height at left: height_left >= 1.2*ATR
- Max_span between first pivot and last pivot <= max_span

Entry/Stop/Target:

- Entry: breakout beyond upper/lower trendline confirmed by close beyond breakout_margin = 0.12 * ATR or close outside by X% of current range (configurable). Prefer breakout retest entry: wait for pullback to broken trendline within N bars (N=3) and take limit at retest price.
- Stop: opposite trendline or stop_padding = 0.8*ATR
- Target: measured_move = height_at_left; TP = measured_move projected from breakout. Use TP1=0.6*measured_move, TP2=measured_move. Notes: distinguish symmetrical (break either way) vs ascending (bias up) vs descending (bias down). Use MA direction filter (e.g., 21 EMA slope) to bias trade direction.

1. Wedges (Rising & Falling)

Definition: Converging lines but both slope same direction; typically reversal pattern.

Detection thresholds:

- Both trendlines slope same direction and converge; height_left >= 1.2*ATR
- Duration: typically shorter than triangles; max_span smaller (e.g., 20 4H bars, 40 1D bars) Entry/Stop/Target
- Entry: breakout against wedge slope (e.g., rising wedge breaks down). Confirm close beyond breakout_margin = 0.12*ATR.
- Stop: above last high for rising wedge / below last low for falling wedge + 0.6*ATR
- Target: measured_move = height_at_left.

1. Flags & Pennants

Definition:

- Flagpole: sharp move (impulse) followed by small rectangle/triangle consolidation (flag/pennant). Detection thresholds
- Flagpole length: initial impulse move >= 2.0 * ATR * N_impulse where N_impulse roughly bars in impulse (configurable). Simpler: impulse amplitude >= 2.0 * ATR.
- Flag body height <= 0.6 * flagpole length and <= 1.0 * ATR
- Duration: flag/pennant lasts 1–8 bars (4H) or 1–6 bars (1D)

Entry/Stop/Target:

- Entry: breakout in direction of flagpole beyond breakout_margin = 0.08 * ATR.
- Stop: below opposite side of flag by 0.6*ATR
- Target: project flagpole length from breakout; TP1=0.6*pole, TP2=pole.

1. Channels (Up / Down)

Definition:

- Parallel trendlines with at least 3 touches on each side (ideal), consistent slope. Detection thresholds
- Channel width >= 1.2 * ATR and minor deviations allowed within 0.4*ATR.
- Min touches: left/mid/right touches count >= 2 on each side.

Entry/Stop/Target: - Range trading: buy near lower trendline when price within buffer = 0.2*channel_width of line; sell near upper line. - Breakout trading: if price closes beyond channel by breakout_margin = 0.12*ATR with volume/volatility confirmation, trade breakout with target = channel_width projected. - Stops: opposite line ± stop_padding = 0.6*ATR.

1. Rectangles / Ranges

Definition: Horizontal support/resistance band with repeated bounces.

Detection thresholds:

- Range height >= 1.2*ATR
- At least 3 touches on both SR lines within max_span

Entry/Stop/Target:

- Buy at support boundary + small offset; sell at resistance boundary - offset.
- Stop: beyond boundary by 0.6*ATR
- Target: mid-range or opposite boundary; prefer partial fills at mid and final at opposite boundary.

Entry / Stop / Target templates (codable) - All entries are processed at candle close: 1. Detect pattern at close t. 2. Calculate entry_price (limit) per pattern rule. 3. Place limit order with ttl = max_wait_candles (default 3 for 4H, 2 for 1D). If not filled by ttl, either: - Cancel and abort OR - Place market order at next open (strategy param). - Position sizing:

risk_amount = equity * risk_per_trade (default 0.005). Position_size = risk_amount / (stop_distance_in_price * contract_size_conversion). For FX use notional sizing: lots = risk_amount / (stop_pips * pip_value). - Slippage model: assume slippage = k * ATR (default k=0.05) or slippage in pips; apply when market orders executed or when limit orders are filled via market movement. - Spread model: model spread as bid/ask difference in backtest; entry/exit price adjusted to worst side (long: buy at ask; short: sell at bid) using spread_at_time (fixed or time-varying). - Rollover/swap: include nightly rollover cost prorated per day; subtract from PnL on rollover_date or daily.

Pseudocode: generic pattern detection + trade lifecycle (Python-style)

```
# Inputs: ohlcv DataFrame with columns [open, high, low, close, volume], ti
# Config: ATR_len=14, risk_per_trade=0.005, max_wait_candles = 3 (4H), brea
atr = ATR(df['high'], df['low'], df['close'], length=14)

for idx in range(min_index, len(df)):
    window = df.iloc[idx-window_size:idx+1]
    # example: detect head and shoulders at idx
    if detect_head_shoulders(window, atr[idx]):
        neckline = compute_neckline(window)
        breakout_price = neckline - breakout_margin*atr[idx]  # for H&S sho
        entry = {'side':'sell', 'price': breakout_price, 'ttl': max_wait_ca
        submit_limit(entry)
    process_existing_orders_and_positions(idx, df, atr)
```

Example: H&S detect function (simplified)

```
def detect_head_shoulders(window, atr_now):
    pivots = find_pivots(window)  # returns indices of pivot highs/lows
    for triplet in possible_triplets(pivots):
        LS, H, RS = triplet
        if (H.level - max(LS.level, RS.level)) >= 0.6*atr_now and abs(LS.le
            if (H.level - neckline_level(window, LS, RS)) >= 1.2*atr_now:
                return True, compute_measures(...)
    return False, None
```

## Backtest setup & execution realism

- Data:
  - Use tick-level if available for best realism. If only OHLCV, use bar-slicing with assumed uniform intra-bar distribution OR simulate intrabar high/low sequences (cautious approach).
  - Use UTC candles, align broker server time (apply timezone conversion if broker uses different zone).
- Resampling:
  - For 4H: aggregate 1H or 15m ticks properly. For 1D: use daily close ignoring holidays for FX (FX trades 24/5).

- Spread modeling:
  - Model spread as time-varying or fixed per instrument. For backtests use per-bar spread in pips, e.g., EURUSD 0.6 pip (0.00006) baseline. Entry price for long = mid + spread/2; exit for long = mid - spread/2 when selling.
- Slippage:
  - Two components: slippage on limit fills when price gaps through limit (adverse/positive) and slippage on market orders. Model as random normal centered at 0 with stdev = slippage_factor * ATR or deterministic adverse slippage = k*spread* + *m*ATR.
  - Conservative default: slippage = max(0.2*pip, 0.05*ATR) adverse to trader.
- Partial fills and liquidity:
  - For larger notional sizes, model partial fills or refusals; include max_fill_pct per bar. If not filled, cancel remainder or scale-in.
- Rollover/swap:
  - Apply daily swap interest cost for held positions using instrument swap rates. In absence of real rates, use conservative estimate (e.g., 0.00001 * notional per day) or obtain historical swap from broker.
- Commission:
  - Model fixed commission per side or per million notional.
- Execution order handling:
  - Deterministic limit entry with ttl. When breakout occurs intrabar, treat as filled at breakout price adjusted for spread and slippage. If close only check used, use open of next candle for market fills.
- Walk-Forward and OOS:
  - Use rolling walk-forward: train/parameter-select on in-sample (e.g., 70% window), test on next out-of-sample segment (30%), then roll forward by step (e.g., monthly for 1D, weekly for 4H).
  - Avoid using future bars in feature calculations. Use only up-to-index data for indicators.
- Survivorship & sample selection:
  - FX currency pairs are generally survivorship-free; still ensure consistent instrument list and correct handling of discontinued pairs.
- Lookahead / leakage pitfalls:
  - When computing pivots and confirming pattern, only use data up to that candle. Do not peek at future close beyond detection candle.

## Walk-forward, statistical testing & significance

- Minimum sample size: require at least N occurrences (default N>=30) of pattern backtested to consider statistical claims; else treat as anecdotal and combine with bootstrap.
- Use bootstrapping or block-bootstrap on trade sequences to estimate p-values of performance metrics (Sharpe, CAGR).
- Control multiple-hypothesis testing: apply Bonferroni or Benjamini-Hochberg when testing many patterns/parameters.

- Outlier removal: winsorize extreme trade returns before interpreting metrics; but report both raw and winsorized results.

## Robustness controls & filtering

- Volatility filter: require ATR to be within acceptable range (not extremely low/high) relative to lookback median (e.g., ATR_now between 0.5*ATR_median and 3.0*ATR_median) to avoid microstructure noise or desensitization.
- Trend filter: use 21 EMA slope or ADX threshold to bias trades (e.g., only take triangle breakouts in direction of EMA slope > 0).
- Volume/participation: for venues where volume data is meaningful, require above-average volume on breakout (e.g., breakout_volume >= 1.2 * vol_ma21). For FX, use tick count as proxy if available.
- Minimum pattern amplitude: enforce pattern height >= H_min = 1.2*ATR; raise to 1.5*ATR in stricter modes.
- Confirmation retest: optionally wait for a pullback to broken trendline (retest) within N bars and use retest limit entry for higher win rate but slower entries.

## Sample backtest parameters (default)

- Timeframe: 4H or 1D; history: min 5 years for daily, min 2 years for 4H.
- Initial equity: $100,000 (for position sizing examples)
- Risk per trade: 0.5% equity
- ATR_len: 14
- Max_wait_candles: 3 for 4H; 2 for 1D
- Spread model: base_spread in pips per instrument (configurable)
- Slippage: adverse_slippage = max(0.05*ATR, fixed 0.5 pip)
- Commission: $2 per round trip per 100k notional (example)
- Rollover: per-day swap cost using broker data or conservative estimate.

## Example 4H trade (H&S short) — worked example

- Instrument: EURUSD 4H; ATR(14)=0.0020 (20 pips) at detection.
- Detected H&S: head_peak=1.1200, neckline=1.1100, right_shoulder=1.1175.
- Pattern height = 100 pips = 5.0*ATR → valid.
- Entry: limit sell at neckline - 0.1*ATR = 1.1080 (neckline 1.1100 - 2 pips)
- Stop: shoulder_high + 0.6*ATR = 1.1175 + 12 pips = 1.1287 (adjust units)
- Measured move = 1.1200 - 1.1100 = 100 pips. TP1 = 60 pips (1.1020), TP2 = 100 pips (1.0980).
- Position sizing: equity $100k, risk 0.5% = $500. Stop distance = entry - stop = ~10.7*? convert to pips and compute lot size using pip_value.
- Execution: include spread (0.6 pip) on entry and slippage 1pip worst-case when market order fills; adjust actual entry and SL accordingly in backtest.

## Common failure modes for pattern-based systems

- Overfitting to visual patterns: fix parameters with walk-forward, prefer coarse-grained thresholds.
- False breakouts: mitigate with retest

# Chart Patterns

## Overview

This chapter defines common chart patterns and structural setups used for 4H and 1D automated Forex trading, provides codable detection rules, prescriptive entry/stop/target rules using ATR(14) defaults, execution-model notes (spread/slippage), and short pandas-style pseudocode for detection. Emphasis: deterministic rules evaluated at candle close (UTC), ATR-relative thresholds, and backtest realism (bid/ask, slippage, rollover).

## Pattern taxonomy and use cases

- Structural patterns (trend structure): higher-highs/lows, lower-highs/lows, supply/demand zones, swing pivots.
- Continuation patterns: flags, pennants, rectangles.
- Reversal patterns: head & shoulders, double top/bottom, rising/falling wedges.
- Consolidation/breakout patterns: triangles (symmetrical, ascending, descending), ranges.
- Micro-structures for entries: pullbacks to moving averages, trendline touches, retests of breakout candles.
- False-break setups and liquidity sweeps (stop hunts): model explicitly as distinct trade types with tighter rules.

Use cases by timeframe: - 4H: better for swing entries within multi-week moves — captures intra-10-day swings, sensitive to intra-day session flows. - 1D: better for higher-confidence structural breaks and trend-following over multi-day moves (~10-day horizon).

## Common pattern definitions and codable attributes

For each pattern below: detection inputs = OHLCV candles (UTC), ATR14, lookback windows. All tests run at candle close.

Table: pattern → key attributes (use in code as booleans/numeric tests)

- Head & Shoulders (H&S)
  - Attributes: three peaks with central peak highest; left/right shoulders within X% of each other (e.g., 0.6–1.1 of head drop), neckline slope (flat/descending/ascending) fit by linear regression on pivot lows.
  - Lookback: 20–60 candles (4H) | 40–120 candles (1D).
  - Confirmation: break and close below neckline (for bearish) with volume/vol proxy confirmation if available.
- Double Top / Double Bottom
  - Attributes: two peaks/troughs separated by 3–20 candles; peak heights within tolerance (±0.5*ATR or ±1–2%).
  - Confirmation: close below (double top) / above (double bottom) intervening valley/ridge.

- Triangles (symmetrical/ascending/descending)
  - Attributes: converging trendlines from at least two swing highs and lows; slope of lines computed by linear regression; height = vertical distance at pattern start.
  - Confirmation: breakout candle closing beyond trendline + volume proxy or ATR expansion.
- Flags / Pennants
  - Attributes: sharp preceding pole (move ≥ k*ATR, e.g., 3–5*ATR), consolidation of small range (body ≤ 0.6*ATR) for n candles (2–10).
  - Confirmation: breakout in direction of pole with close above/below consolidation extremes.
- Rectangles / Ranges
  - Attributes: horizontal support/resistance with ≥2 touches each side; range height compared to ATR.
  - Trade: trade edges (buy support/sell resistance) or breakout beyond range (wait for close).
- Wedges (rising/falling)
  - Attributes: converging trendlines but sloped opposite to expected breakout direction; often reversal signals after extended moves.
- Breakout with retest (preferred)
  - Attributes: initial close beyond level, followed by pullback that respects level (bounce within 1–2 ATR).
  - Trade: enter on retest candle close reversal signal (e.g., bullish engulf, pin, or close above retest high).
- Liquidity sweep / false break
  - Attributes: quick spike beyond obvious S/R followed by close back inside; candlestick tail length ≥ 0.8*range and > threshold relative to ATR.
  - Trade: fade spike with confirmation (reversal candle close), tight stop beyond tail.

## Codable detection rules — templates

Notes: - Use pivot detection by local maxima/minima over window p (e.g., p=3 or 5 candles) to find swings. - Use ATR14 (default) for size normalization. - Use tolerance params as multiples of ATR or percentage.

Pseudocode (conceptual; replace with concrete pandas code in implementation chapter):

Detect pivot:

```
is_pivot_high(i, p):
  return high[i] == max(high[i-p : i+p+1])

is_pivot_low(i, p):
  return low[i] == min(low[i-p : i+p+1])
```

Head & Shoulders detection (bearish):

```
find_pivots(lookback):
  highs = list of pivot highs indices
  lows  = list of pivot lows indices

for consecutive triples (hL, hH, hR) in highs:
  cond1 = high[hH] > high[hL] and high[hH] > high[hR]
  cond2 = abs(high[hL] - high[hR]) <= max(0.6*ATR, ATR*0.5)
  neck_low = min(low between hL and hR)
  neckline_slope = linear_regression_slope on lows around pivots
  if cond1 and cond2:
    pattern = record(head= hH, left=hL, right=hR, neckline=neck_low, slope=
```

Triangle breakout:

```
fit_trendline(points_highs)
fit_trendline(points_lows)
if lines_converge and pattern_duration between min/max:
  if close[-1] > upper_trendline_at_time or close[-1] < lower_trendline_at_
    signal breakout direction
```

Flag:

```
if prior_move >= 3*ATR and consolidation_range <= 0.6*ATR and candles_in_fl
  if breakout candle close beyond consolidation extreme:
    signal continuation
```

Retest entry (generic):

```
if breakout_detected at t0:
  wait for pullback: price returns to level within 2*ATR
  if reversal_candle_close in direction of breakout and volume/ATR expansio
    enter at close (prefer limit at retest high/low)
```

## Entry, stop, and target rules (prescriptive, codable)

Defaults: - ATR = ATR(14) on chosen timeframe (4H or 1D). - Risk per trade = 0.5% equity (configurable). - Entry evaluated at candle close; prefer limit entries at favorable price levels with fallback to market-plus-slippage. - Model spread by adjusting entry/stop/target to bid/ask depending on side.

Generic mapping: - Stop distance: use ATR multiples depending on pattern volatility: - Tight (false-break, liquidity-fade): 0.6–1.0 * ATR - Standard (breakout w/ retest): 1.0–1.5 * ATR - Structural reversal (H&S, double top): 1.5–3.0 * ATR - Target(s): - Single target: pattern height projection (triangle height, flag pole) or 1.5–3.0 * stop distance (risk:reward 1:1.5–1:3). - Multiple targets / scale-out: first at 1*stop_distance (1:1), second at 2*stop_distance (1:2), remainder trailed by ATR or moving average. - Position sizing: compute size by risk_per_trade / (stop_distance_in_pips * pip_value). Use default 0.5% equity.

Example: 4H triangle breakout long - ATR14 = 40 pips. - Stop = 1.25 * ATR = 50 pips. - Risk per trade = 0.5% of $100,000 = $500. - Pip value (per standard lot) = $10 → position_size = $500 / (50 pips * $10) = 1.0 lot. - Entry = close on breakout candle. If using limit at breakout price and not filled, fallback to market + slippage (modeled).

Execution adjustments (spread/slippage): - For buy entries, convert price to ask for entry and stop levels (ask->bid for stop/target as needed). - Add modeled slippage: e.g., slippage = Normal(mu=0, sigma = 0.2*ATR) or fixed ticks (configurable). Apply slippage as adverse price move on fills. - Ensure stops are placed beyond spread: stop_distance_effective = stop_distance + spread/2.

Example numeric rules (codable): - Enter long when breakout candle close > upper_trendline and close > previous n highs. - Place stop = min(neckline, breakout_level - stop_distance). - Place take-profit1 = entry + 1.0 * (entry - stop) - Place take-profit2 = entry + 2.0 * (entry - stop) - If using trailing stop: trail by ATRe * 0.8 after TP1 reached.

## Execution realism and backtest modeling for patterns

Model these explicitly in backtests: - Bid/Ask modeling: reconstruct ask from mid or OHLC depending on data. For FX, prefer tick or 1-min bid/ask if available. If only mid OHLC provided, for buys assume entry price = close + spread/2 (and vice versa). - Spread: use instrument-specific average spread per timeframe or time-of-day table (session-aware). - Slippage: model as random or deterministic adverse adjustment on fills. Example: slippage ~ Normal(μ=0, σ=0.1*ATR) signed towards worse fill. - Execution priority: limit entries modeled with fill probability dependent on limit aggressiveness; fallback to market after N candles if not filled. - Rollover/swap: for holding multi-day, model overnight rollover charges by instrument, direction, and broker rates. - Partial fills & lot discretization: round to available contract sizes; model minimum lot and partial fills.

Backtest cadence for pattern-based strategies: - Evaluate pattern detection at candle close. - Generate orders deterministically (limit/market) and simulate fills in next candle(s) using bid/ask and slippage assumptions. - If entry depends on intrabar price (e.g., breakout during candle), use higher-frequency data or conservative assumption: require close beyond level (close-based rule) to avoid lookahead.

## Parameter choices and robustness

- Lookbacks: 4H patterns — 20–60 candles (roughly 5–15 days); 1D patterns — 40–120 candles (roughly 2–6 months).
- Pivot window p: 3–5 for 4H; 5–7 for 1D.
- ATR multipliers: use defaults above but run sensitivity analysis: test multipliers in ranges (stop: 0.6–3.0, TP multiplier: 1.0–3.0).
- Avoid overfitting: prefer coarse parameter grids, walk-forward validation, and out-of-sample periods.
- Use slippage/spread stress tests: add +50% spread and +mean slippage to ensure edge persists.

## Pattern-specific worked example (4H breakout with retest)

Context: EUR/USD, 4H candles (UTC), ATR14 = 0.0040 (40 pips), equity = $50,000, risk = 0.5% = $250.

Detection: - Symmetrical triangle formed over last 12 candles; upper trendline broken with candle close at 1.1200.

Trade rules: 1. Entry: place limit buy at retest high when price revisits breakout level within next 5 candles and produces bullish reversal close above retest high. 2. Stop: 1.25 * ATR = 50 pips below entry. 3. Size: position_size = risk / (stop_pips * pip_value). If pip_value = $10 (per standard lot), size = 250 / (50 * 10) = 0.5 lot. 4. Targets: TP1 = entry + 1.0*stop_pips (50 pips), TP2 = entry + 2.0*stop_pips (100 pips). Scale out 50% at TP1. 5. Execution modeling: assume spread 1.2 pips, slippage = 0.5*pip on entry. Model fills accordingly; adjust stop for spread.

Backtest implementation notes: - Mark entry price = observed fill = retest_close + spread/2 + slippage. - If not filled within 5 candles, cancel and abort trade. - Charge rollover per broker for days held.

## Practical detection checklist (to implement)

- Normalize sizes to ATR before thresholds.
- Detect pivots with deterministic window.
- Fit trendlines with least-squares to pivot coordinates.
- Validate pattern duration and minimum size (height >= 0.8*ATR* min_candles).
- Use close-based breakout confirmation to avoid intrabar lookahead unless high-frequency data used.
- Implement entry limit with configurable timeout and market fallback.
- Simulate spread/slippage/rollover in backtest and stress-test.

## Common pitfalls and mitigations

- Overfitting visual patterns — mitigate with parameter grid search & walk-forward testing.
- Using intrabar breakouts with only OHLC daily/4H data — require close confirmation or higher-frequency data.
- Ignoring spread on small-range patterns — always compare pattern height to spread and ATR.
- Failing to model skipped fills and partial fills — model order lifecycle and cancellation logic.
- Survivorship bias in pattern success rates — ensure instrument sample integrity.

## Recommended exercises (codable)

1. Implement a pivot detector (p=3) and list last 20 pivots on 4H EUR/USD.
2. Code a triangle detector: fit trendlines to last 20 pivots, detect convergence, and mark breakout candles.

3. Backtest a triangle-breakout-with-retest strategy for 4H on a currency pair using ATR-based stops and two TP levels; include spread=1.5 pips, slippage=0.5 pips, and report win rate, CAGR, max drawdown.
4. Run sensitivity: vary stop ATR multiplier from 0.8 to 2.5 and plot Sharpe ratio surface.

## Summary — implementation-ready checklist

- Use UTC candles, ATR(14) normalization.
- Detect pivots, fit trendlines, measure pattern height.
- Use close-based confirmations; prefer retest entries.
- Size by ATR-based stops and default 0.5% risk.
- Model spread, slippage, rollover, and fill mechanics in backtests.
- Validate with walk-forward and out-of-sample testing.

# Structural Setups

Chapter summary: provides deterministic, codable definitions for common chart patterns, entry/exit/stop/target rules, ATR-relative thresholds, pseudocode and backtest setup (including spread/slippage/rollover). All rules assume candle-close decisioning on UTC 4H or 1D bars and default parameters: ATR(14), risk per trade = 0.5% equity, candle time = UTC, deterministic limit entries with fallback to market if not filled within N candles.

Table of contents:



Overview and pattern-use guidance:

- Use patterns as structural context + trade trigger; combine with trend / volatility filter (ATR, MA) and volume where available.
- For 4H/1D systems aimed at ~10-day horizons: prefer patterns that complete within 3–20 bars (4H) or 2–14 bars (1D).
- Require a minimum structural amplitude relative to ATR to avoid micro-noise signals.
- Deterministic rules: detect pattern at candle close, compute entry as limit at breakout/counterlevel, if not filled within max_wait_candles then cancel or use market entry at next candle open (configurable).

Pattern detection conventions and variables:

- Use high/low/close/open arrays indexed by increasing time.
- ATR = ATR(14) computed on same timeframe.
- candle_range = high - low
- body_size = abs(close - open)

- pivot detection: use n=3 or n=5 swing pivots (configurable). A pivot high at index i if high[i] > high[i-k:i) and high[i] >= high((i+1):(i+k+1)] with k=2 (5-bar pivot) unless otherwise noted.
- All thresholds expressed relative to ATR (default) or absolute ticks/pips if preferred.
- Min amplitude rule: pattern height >= 1.2 * ATR (configurable to 1.5 for stricter).
- Time limits: max bars between key pattern points to prevent multi-month structures (max_span = 30 for 4H by default, 120 for 1D by default — adjust for strategy).

1. Head & Shoulders (H&S) and Inverse H&S

Definition (codable):

- Three consecutive pivot highs (H&S) or pivot lows (Inverse) with middle peak (head) higher (lower) than shoulders.
- Neckline: linear fit across the two shoulder troughs/peaks; breakout is close crossing neckline. Detection thresholds
- Left_shoulder, head, right_shoulder pivots must satisfy: head_height >= shoulders_height + 0.6*ATR
- Shoulder heights within symmetry: abs(LS - RS) <= 0.6 * ATR
- Pattern height (head_peak - neckline_level) >= 1.2 * ATR
- Max bars between LS and RS <= max_span

Entry/Stop/Target (codable):

- Entry: place sell (H&S) or buy (Inverse) limit at breakout close below/above neckline on candle close. Prefer conservative entry: require close beyond neckline by gap = 0.1*ATR.
- Stop: above last shoulder by stop_padding = 0.6 * ATR (H&S) or below last shoulder for inverse.
- Target: measured move = head_peak - neckline -> project from breakout point. Take-profit levels: TP1 = 0.6*measured_move, TP2 = measured_move.
- Trail: move stop to breakeven after 0.6*measured_move; then use ATR(14)*1.0 trailing step.

Example parameters (defaults): - ATR_min_height = 1.2 * ATR - symmetry_tol = 0.6 * ATR - breakout_margin = 0.1 * ATR - stop_padding = 0.6 * ATR

1. Double Top / Double Bottom

Definition: Two pivots near same level separated by trough/peak; neckline = intervening trough/peak.

Detection thresholds:

- Peak separation: bars_between_peaks <= max_span
- Peak level difference: abs(P1 - P2) <= 0.6 * ATR
- Intervening trough depth >= 0.6 * ATR (ensures structure)

Entry/Stop/Target:

- Entry: on close below trough (double top) or above peak (double bottom) by breakout_margin = 0.1 * ATR.
- Stop: above the nearer peak/two-peak high + stop_padding = 0.6*ATR.
- Target: measured_move = average(P1,P2) - trough -> project measured_move from breakout; TP1 = 0.6*measured_move, TP2 = measured_move.
- Fail-safe: if price revisits peaks within X bars (X=3) after breakout, exit.

1. Triangles (Symmetrical / Ascending / Descending)

Definition: Formed by converging trendlines connecting at least two highs and two lows (pivots).

Detection thresholds:

- Convergence slope: distance between lines at left > distance at right by at least 20% of left distance (i.e., narrowing).
- Min height at left: height_left >= 1.2*ATR
- Max_span between first pivot and last pivot <= max_span

Entry/Stop/Target:

- Entry: breakout beyond upper/lower trendline confirmed by close beyond breakout_margin = 0.12 * ATR or close outside by X% of current range (configurable). Prefer breakout retest entry: wait for pullback to broken trendline within N bars (N=3) and take limit at retest price.
- Stop: opposite trendline or stop_padding = 0.8*ATR
- Target: measured_move = height_at_left; TP = measured_move projected from breakout. Use TP1=0.6*measured_move, TP2=measured_move. Notes: distinguish symmetrical (break either way) vs ascending (bias up) vs descending (bias down). Use MA direction filter (e.g., 21 EMA slope) to bias trade direction.

1. Wedges (Rising & Falling)

Definition: Converging lines but both slope same direction; typically reversal pattern.

Detection thresholds:

- Both trendlines slope same direction and converge; height_left >= 1.2*ATR
- Duration: typically shorter than triangles; max_span smaller (e.g., 20 4H bars, 40 1D bars) Entry/Stop/Target
- Entry: breakout against wedge slope (e.g., rising wedge breaks down). Confirm close beyond breakout_margin = 0.12*ATR.
- Stop: above last high for rising wedge / below last low for falling wedge + 0.6*ATR
- Target: measured_move = height_at_left.

1. Flags & Pennants

Definition:

- Flagpole: sharp move (impulse) followed by small rectangle/triangle consolidation (flag/pennant). Detection thresholds
- Flagpole length: initial impulse move >= 2.0 * ATR * N_impulse where N_impulse roughly bars in impulse (configurable). Simpler: impulse amplitude >= 2.0 * ATR.
- Flag body height <= 0.6 * flagpole length and <= 1.0 * ATR
- Duration: flag/pennant lasts 1–8 bars (4H) or 1–6 bars (1D)

Entry/Stop/Target:

- Entry: breakout in direction of flagpole beyond breakout_margin = 0.08 * ATR.
- Stop: below opposite side of flag by 0.6*ATR
- Target: project flagpole length from breakout; TP1=0.6*pole, TP2=pole.

1. Channels (Up / Down)

Definition:

- Parallel trendlines with at least 3 touches on each side (ideal), consistent slope. Detection thresholds
- Channel width >= 1.2 * ATR and minor deviations allowed within 0.4*ATR.
- Min touches: left/mid/right touches count >= 2 on each side.

Entry/Stop/Target: - Range trading: buy near lower trendline when price within buffer = 0.2*channel_width of line; sell near upper line. - Breakout trading: if price closes beyond channel by breakout_margin = 0.12*ATR with volume/volatility confirmation, trade breakout with target = channel_width projected. - Stops: opposite line ± stop_padding = 0.6*ATR.

1. Rectangles / Ranges

Definition: Horizontal support/resistance band with repeated bounces.

Detection thresholds:

- Range height >= 1.2*ATR
- At least 3 touches on both SR lines within max_span

Entry/Stop/Target:

- Buy at support boundary + small offset; sell at resistance boundary - offset.
- Stop: beyond boundary by 0.6*ATR
- Target: mid-range or opposite boundary; prefer partial fills at mid and final at opposite boundary.

Entry / Stop / Target templates (codable) - All entries are processed at candle close: 1. Detect pattern at close t. 2. Calculate entry_price (limit) per pattern rule. 3. Place limit order with ttl = max_wait_candles (default 3 for 4H, 2 for 1D). If not filled by ttl, either: - Cancel and abort OR - Place market order at next open (strategy param). - Position sizing:

risk_amount = equity * risk_per_trade (default 0.005). Position_size = risk_amount / (stop_distance_in_price * contract_size_conversion). For FX use notional sizing: lots = risk_amount / (stop_pips * pip_value). - Slippage model: assume slippage = k * ATR (default k=0.05) or slippage in pips; apply when market orders executed or when limit orders are filled via market movement. - Spread model: model spread as bid/ask difference in backtest; entry/exit price adjusted to worst side (long: buy at ask; short: sell at bid) using spread_at_time (fixed or time-varying). - Rollover/swap: include nightly rollover cost prorated per day; subtract from PnL on rollover_date or daily.

Pseudocode: generic pattern detection + trade lifecycle (Python-style)

```
# Inputs: ohlcv DataFrame with columns [open, high, low, close, volume], ti
# Config: ATR_len=14, risk_per_trade=0.005, max_wait_candles = 3 (4H), brea
atr = ATR(df['high'], df['low'], df['close'], length=14)

for idx in range(min_index, len(df)):
    window = df.iloc[idx-window_size:idx+1]
    # example: detect head and shoulders at idx
    if detect_head_shoulders(window, atr[idx]):
        neckline = compute_neckline(window)
        breakout_price = neckline - breakout_margin*atr[idx]  # for H&S sho
        entry = {'side':'sell', 'price': breakout_price, 'ttl': max_wait_ca
        submit_limit(entry)
    process_existing_orders_and_positions(idx, df, atr)
```

Example: H&S detect function (simplified)

```
def detect_head_shoulders(window, atr_now):
    pivots = find_pivots(window)  # returns indices of pivot highs/lows
    for triplet in possible_triplets(pivots):
        LS, H, RS = triplet
        if (H.level - max(LS.level, RS.level)) >= 0.6*atr_now and abs(LS.le
            if (H.level - neckline_level(window, LS, RS)) >= 1.2*atr_now:
                return True, compute_measures(...)
    return False, None
```

## Backtest setup & execution realism

- Data:
  - Use tick-level if available for best realism. If only OHLCV, use bar-slicing with assumed uniform intra-bar distribution OR simulate intrabar high/low sequences (cautious approach).
  - Use UTC candles, align broker server time (apply timezone conversion if broker uses different zone).
- Resampling:
  - For 4H: aggregate 1H or 15m ticks properly. For 1D: use daily close ignoring holidays for FX (FX trades 24/5).

- Spread modeling:
  - Model spread as time-varying or fixed per instrument. For backtests use per-bar spread in pips, e.g., EURUSD 0.6 pip (0.00006) baseline. Entry price for long = mid + spread/2; exit for long = mid - spread/2 when selling.
- Slippage:
  - Two components: slippage on limit fills when price gaps through limit (adverse/positive) and slippage on market orders. Model as random normal centered at 0 with stdev = slippage_factor * ATR or deterministic adverse slippage = k*spread* + *m*ATR.
  - Conservative default: slippage = max(0.2*pip, 0.05*ATR) adverse to trader.
- Partial fills and liquidity:
  - For larger notional sizes, model partial fills or refusals; include max_fill_pct per bar. If not filled, cancel remainder or scale-in.
- Rollover/swap:
  - Apply daily swap interest cost for held positions using instrument swap rates. In absence of real rates, use conservative estimate (e.g., 0.00001 * notional per day) or obtain historical swap from broker.
- Commission:
  - Model fixed commission per side or per million notional.
- Execution order handling:
  - Deterministic limit entry with ttl. When breakout occurs intrabar, treat as filled at breakout price adjusted for spread and slippage. If close only check used, use open of next candle for market fills.
- Walk-Forward and OOS:
  - Use rolling walk-forward: train/parameter-select on in-sample (e.g., 70% window), test on next out-of-sample segment (30%), then roll forward by step (e.g., monthly for 1D, weekly for 4H).
  - Avoid using future bars in feature calculations. Use only up-to-index data for indicators.
- Survivorship & sample selection:
  - FX currency pairs are generally survivorship-free; still ensure consistent instrument list and correct handling of discontinued pairs.
- Lookahead / leakage pitfalls:
  - When computing pivots and confirming pattern, only use data up to that candle. Do not peek at future close beyond detection candle.

## Walk-forward, statistical testing & significance

- Minimum sample size: require at least N occurrences (default N>=30) of pattern backtested to consider statistical claims; else treat as anecdotal and combine with bootstrap.
- Use bootstrapping or block-bootstrap on trade sequences to estimate p-values of performance metrics (Sharpe, CAGR).
- Control multiple-hypothesis testing: apply Bonferroni or Benjamini-Hochberg when testing many patterns/parameters.

- Outlier removal: winsorize extreme trade returns before interpreting metrics; but report both raw and winsorized results.

## Robustness controls & filtering

- Volatility filter: require ATR to be within acceptable range (not extremely low/high) relative to lookback median (e.g., ATR_now between 0.5*ATR_median and 3.0*ATR_median) to avoid microstructure noise or desensitization.
- Trend filter: use 21 EMA slope or ADX threshold to bias trades (e.g., only take triangle breakouts in direction of EMA slope > 0).
- Volume/participation: for venues where volume data is meaningful, require above-average volume on breakout (e.g., breakout_volume >= 1.2 * vol_ma21). For FX, use tick count as proxy if available.
- Minimum pattern amplitude: enforce pattern height >= H_min = 1.2*ATR; raise to 1.5*ATR in stricter modes.
- Confirmation retest: optionally wait for a pullback to broken trendline (retest) within N bars and use retest limit entry for higher win rate but slower entries.

## Sample backtest parameters (default)

- Timeframe: 4H or 1D; history: min 5 years for daily, min 2 years for 4H.
- Initial equity: $100,000 (for position sizing examples)
- Risk per trade: 0.5% equity
- ATR_len: 14
- Max_wait_candles: 3 for 4H; 2 for 1D
- Spread model: base_spread in pips per instrument (configurable)
- Slippage: adverse_slippage = max(0.05*ATR, fixed 0.5 pip)
- Commission: $2 per round trip per 100k notional (example)
- Rollover: per-day swap cost using broker data or conservative estimate.

## Example 4H trade (H&S short) — worked example

- Instrument: EURUSD 4H; ATR(14)=0.0020 (20 pips) at detection.
- Detected H&S: head_peak=1.1200, neckline=1.1100, right_shoulder=1.1175.
- Pattern height = 100 pips = 5.0*ATR → valid.
- Entry: limit sell at neckline - 0.1*ATR = 1.1080 (neckline 1.1100 - 2 pips)
- Stop: shoulder_high + 0.6*ATR = 1.1175 + 12 pips = 1.1287 (adjust units)
- Measured move = 1.1200 - 1.1100 = 100 pips. TP1 = 60 pips (1.1020), TP2 = 100 pips (1.0980).
- Position sizing: equity $100k, risk 0.5% = $500. Stop distance = entry - stop = ~10.7*? convert to pips and compute lot size using pip_value.
- Execution: include spread (0.6 pip) on entry and slippage 1pip worst-case when market order fills; adjust actual entry and SL accordingly in backtest.

### Common failure modes for pattern-based systems

- Overfitting to visual patterns: fix parameters with walk-forward, prefer coarse-grained thresholds.
- False breakouts: mitigate with retest

# Indicators and Oscillators

## Overview

Practical, codable definitions of commonly used indicators and oscillators for 4H and 1D Forex automated trading. Focus: deterministic, non-ambiguous formulas; default parameters and ATR-normalized thresholds; signal construction (entry/exit) suitable for ~10-day horizons; implementation hints (pandas/numpy), lookahead avoidance, and execution/backtest realism (spread/slippage, resampling). Include recommended parameter grids for robustness testing.

## General rules for indicator use (applied to all)

- Compute on UTC candles; avoid mixing timezones.
- Use close prices for non-lagging trend indicators; for oscillators use close or typical price ((H+L+C)/3) consistently.
- Calculate indicators on the same timeframe used for signaling (4H or 1D). If combining multi-timeframe signals, compute higher timeframe indicators independently from resampled data.
- Avoid using future data: all indicator values must be computed up to index t for decisions at t (i.e., use .shift(0) values only).
- Normalize thresholds to ATR when threshold depends on price movement magnitude.
- Always include lookback/warm-up period equal to max indicator length before using signals.
- Use vectorized implementations in pandas/numpy for speed; keep code deterministic.

## Moving Averages (MA) & Trend Filters

Purpose: identify trend direction, smooth price, dynamic support/resistance.

Common types and defaults: - SMA (Simple MA): windows: 50 (4H ≈ 200h), 100 (4H), 200 (4H) — for 1D use 20/50/200. - EMA (Exponential MA): defaults: 20, 50 for signal generation (reactive). - WMA/HLMA: optional for reducing lag.

Signals (codable): - Trend filter: price > MA_k → bullish trend; price < MA_k → bearish. - Crossovers: fast_EMA(20) crosses above slow_EMA(50) on close → buy; cross below → sell. - MA slope: slope = (MA[t] - MA[t-n]) / n. Use slope threshold relative to ATR (slope_abs > 0.02*ATR_per_period) for confirming trend strength.

Implementation note (pandas example):

```
ema_fast = price.ewm(span=20, adjust=False).mean()
ema_slow = price.ewm(span=50, adjust=False).mean()
cross_up = (ema_fast > ema_slow) & (ema_fast.shift(1) <= ema_slow.shift(1))
```

Pitfalls: - Whipsaws in choppy markets — combine with volatility filter (ATR) or use breakout confirmation.

### Average True Range (ATR) — volatility measure (default ATR(14))

- Default: ATR(14) on the used timeframe.
- Use for stop sizing, position sizing, volatility filters (e.g., skip signals if ATR < min_ATR or pattern height < k*ATR).
- Compute using Wilder’s smoothing or simple rolling mean; be consistent.

Pandas example:

```
tr1 = high - low
tr2 = (high - close.shift()).abs()
tr3 = (low - close.shift()).abs()
tr = pd.concat([tr1, tr2, tr3], axis=1).max(axis=1)
atr = tr.ewm(alpha=1/14, adjust=False).mean()
```

### Momentum indicators

- Rate of Change (ROC): window 9–21 (4H: 9; 1D: 14). Signal: ROC crossing zero or extreme percentiles.
- Momentum via derivative: price.diff(n) normalized by ATR*n.

Signal example: - Buy when ROC(14) > 0 and ROC rising; sell when ROC < 0 and falling.

### RSI (Relative Strength Index)

- Default: RSI(14).
- Use for mean-reversion and divergence detection. For 4H/1D prefer 14; test 7–21 for sensitivity.
- Thresholds: overbought = 70, oversold = 30. For automated trades use ATR-normalized triggers: only act when RSI extreme coincides with price beyond +1*ATR in direction of reversal.

Pandas implementation (Smoothed):

```
delta = close.diff()
up = delta.clip(lower=0)
down = -delta.clip(upper=0)
avg_up = up.ewm(alpha=1/14, adjust=False).mean()
avg_down = down.ewm(alpha=1/14, adjust=False).mean()
rs = avg_up / avg_down
rsi = 100 - (100 / (1 + rs))
```

Signal construction: - Mean-reversion entry: RSI < 30 AND price within X*ATR of identified support → buy. - Trend-confirm entry: RSI crosses above 50 AND EMA20 > EMA50.

### Stochastic Oscillator

- Default: %K=14, %D=3.
- Use for short-term pullback timing in 4H; less helpful on 1D for multi-day swings unless combined with higher-timeframe trend filter.
- Signal: %K crossing above %D from <20 for bullish entry, with ATR-confirmation.

### MACD (Moving Average Convergence Divergence)

- Default: fast=12, slow=26, signal=9 (standard).
- Use for trend direction and momentum divergence.
- Codable signals: MACD histogram turning positive + MACD > 0 for trend-following entries; divergence between MACD and price for reversals (requires robust implementation and guard against false positives).

Implementation:

```
ema12 = close.ewm(span=12, adjust=False).mean()
ema26 = close.ewm(span=26, adjust=False).mean()
macd = ema12 - ema26
signal = macd.ewm(span=9, adjust=False).mean()
hist = macd - signal
```

### Bollinger Bands

- Default: 20-period SMA, 2.0 standard deviations.
- Use for volatility breakouts and mean reversion.
- Signal ideas:
  - Squeeze → expect breakout; measure bandwidth = (upper - lower) / SMA. If bandwidth < threshold (e.g., 0.6*ATR/SMA) then mark consolidation.
  - Breakout: close > upper → momentum buy if volume/ATR expands.
  - Mean-reversion: price touches lower band with RSI < 30 → potential buy.

Compute:

```
sma20 = close.rolling(20).mean()
std20 = close.rolling(20).std()
upper = sma20 + 2*std20
lower = sma20 - 2*std20
bandwidth = (upper - lower) / sma20
```

### Volume/Proxy-volume (for FX use tick/quote volume or range as proxy)

- FX often lacks true volume; use tick counts, broker-provided volume, or range/ATR * number_of_ticks as proxies.
- Use for breakout confirmation: breakout with ATR expansion or proxy-volume spike is more reliable.

Signal: Breakout candle close beyond resistance AND candle_range > k*ATR and tick_volume > mean_tick_volume* 1.5.

### Composite signals and filters

- Always combine a trend filter + trigger + volatility filter for 4H/1D:
  - Trend filter: EMA20>EMA50 (trend direction)
  - Trigger: price breakout above recent high or RSI crossing threshold
  - Volatility filter: ATR > min_ATR and candle_range > 0.6*ATR
- Example combined buy rule (codable):

```
if ema20 > ema50 and close > recent_high and atr > atr_min and close > uppe
  generate_long_signal
```

### Constructing entry/exit rules from indicators

- Entry on confirmation: require indicator + price confirmation within same candle or close-based confirmation next candle.
- Stop placement: use ATR-based stop as primary; if indicator provides a natural level (e.g., lower Bollinger band, swing low), use the more conservative.
- Exits:
  - Fixed TP using ATR multiple or pattern projection.
  - Indicator-based exit: RSI crossing back through threshold (e.g., RSI > 70 then close below 65).
  - Trailing exit: MA slope turning negative or price closing below EMA20.

### Parameter defaults & sensitivity grids

Default sets (start here; test via grid/walk-forward): - ATR: 14 - EMA windows: 20 (fast), 50 (slow) - RSI: 14 - MACD: 12/26/9 - Bollinger: 20, 2.0 - Stochastic: 14, 3 - ROC: 14

Suggested grid ranges for robustness testing: - EMA_fast: [10, 20, 30] - EMA_slow: [40, 50, 100] - ATR_periods: [10, 14, 21] - RSI_period: [7, 14, 21] - Stop ATR multiplier: [0.6, 1.0, 1.5, 2.0]

### Implementation notes — pandas tips & avoiding lookahead

- Compute all rolling/ewm with min_periods equal to length; discard initial NaNs.
- For crossover detection use shifted boolean arrays:

```
cross_up = (fast > slow) & (fast.shift(1) <= slow.shift(1))
```

- When combining indicators across timeframes, resample high-frequency data to the target timeframe explicitly and compute indicators on resampled series.
- For backtesting, generate signals at candle close, create orders for next candle open or place limit orders at computed levels; never use future candle values.

## Execution and backtest realism (indicator-specific)

- Signal slippage: indicators can trigger many false breakouts; model execution slippage as function of signal rate (more signals → worse practical fill rates).
- Time-of-day effects: compute indicator behavior per session (e.g., London open) and include session filters.
- Spread sensitivity: for narrow indicator thresholds (e.g., small breakout of 5 pips) compare threshold to spread + slippage; skip signals where edge < transaction cost.

## Example strategies built from indicators (short sketches)

1. Trend-follow EMA crossover (4H)
    - Entry: EMA20 crosses above EMA50 on close, ATR > 0.8*ATR_mean.
    - Stop: 1.25*ATR below entry.
    - TP: 2*stop; trail by ATR*0.8 after first TP.
    - Size: 0.5% risk.
2. Bollinger breakout with volatility confirmation (1D)
    - Entry: close > upper_band AND bandwidth > 0.8*historical_bandwidth AND range > 1.2*ATR.
    - Stop: SMA20 or 1.5*ATR below entry (whichever lower).
    - TP: 1.5–3.0*ATR multiples.
3. RSI mean-reversion (4H) (short horizon)
    - Trend filter: EMA50 down
    - Entry: RSI < 30 AND close touches lower Bollinger band
    - Stop: 1.0*ATR above entry (for shorts invert)
    - TP: 1.5*stop

Each example must be backtested with spread/slippage/rollover modeled and parameter sensitivity tested.

## Common pitfalls and mitigations

- Over-optimizing indicator windows — use coarse grids and walk-forward.
- Chaining many indicators increases complexity and overfitting risk—prefer minimal effective combos (trend filter + trigger + volatility).
- Ignoring indicator latency — EMA and SMA lag; adjust stop/TP accordingly.
- Using indicators not robust to regime change — include regime filters (ATR, VIX-like proxies) and retrain parameter selection periodically.

## Practical exercises (codable)

1. Implement EMA20/EMA50 crossover for 4H EUR/USD with ATR-based stops; backtest with spread=1.5 pips, slippage normal(0, 0.2*ATR).
2. Code Bollinger squeeze detector and log every squeeze event; measure subsequent 10-day return distribution.

3. Create a composite filter: EMA trend + RSI trigger + ATR threshold; run walk-forward optimization over 3-year data and report out-of-sample performance.

## Summary — quick reference table

- ATR(14): volatility/stops/sizing (default)
- EMA20/50: trend filter & crossovers (default)
- RSI14: mean-reversion/divergence
- MACD 12/26/9: momentum/trend strength
- Bollinger 20/2: volatility squeeze/breakouts
- Stochastic 14/3: short-term entry timing
- ROC14: momentum confirmation

Use these with ATR normalization, close-based confirmations, spread/slippage modeling, and walk-forward validation.

# Risk Management and Position Sizing

## Overview

Concrete, codable risk-management and sizing rules for automated Forex trading on 4H and 1D timeframes. Includes position-sizing formulas, portfolio risk limits, margin-aware sizing, ATR-based stops, walk-forward parameter guidance, trade lifecycle risk controls (kill switches, max concurrent trades), and Python-ready examples. Defaults: ATR(14), risk per trade = 0.5% equity, candle timezone = UTC.

## Core principles (decisive, codable)

- Fixed fractional risk per trade: risk_amount = equity * risk_pct (default risk_pct = 0.005).
- Risk measured in money per trade = position_size * stop_distance_in_price * pip_value.
- Stop must be defined before sizing. Always compute stop_distance in quote units (pips or price).
- Account for spread and slippage when computing effective stop distance.
- Respect margin/leverage constraints; compute margin required per position and cap position by available margin.
- Enforce portfolio-level caps: max_drawdown_limit, max_daily_loss, max_open_positions.
- Use ATR(14) for stop sizing and volatility-normalized thresholds.
- Implement deterministic, idempotent sizing routines in code (pure functions given equity, ATR, instrument params).

## Position sizing formulas (codable)

Inputs:

- equity: current account equity (USD)
- risk_pct: fraction of equity to risk per trade (default 0.005)
- stop_pips: stop distance in pips (or price units)
- pip_value_per_lot: USD per pip per standard lot for the instrument
- lot_step, min_lot, max_lot: broker constraints
- margin_rate: required margin fraction (e.g., 1% for 100:1 leverage → 0.01) OR use margin_per_lot

Compute risk_money:

- risk_money = equity * risk_pct

Compute lot_size (standard lots):

- raw_lots = risk_money / (stop_pips * pip_value_per_lot)

- Apply lot discretization:
  - lots = floor(raw_lots / lot_step) * lot_step
  - if lots < min_lot → lots = 0 (do not trade)
  - lots = min(lots, max_lot)

Margin check:

- required_margin = lots * margin_per_lot (or notional * margin_rate)
- if required_margin > free_margin → reduce lots or skip trade

Return:

- final lots, required_margin, expected_risk_money (lots * stop_pips * pip_value_per_lot)

Example (codable):

```
def compute_lots(equity, risk_pct, stop_pips, pip_value, lot_step, min_lot,
    risk_money = equity * risk_pct
    raw_lots = risk_money / (stop_pips * pip_value)
    lots = math.floor(raw_lots / lot_step) * lot_step
    if lots < min_lot:
        return 0, 0
    lots = min(lots, max_lot)
    required_margin = lots * margin_per_lot
    if required_margin > free_margin:
        # reduce lots to fit margin
        max_lots_by_margin = math.floor(free_margin / margin_per_lot / lot_
        lots = min(lots, max_lots_by_margin)
        if lots < min_lot:
            return 0, 0
    expected_risk = lots * stop_pips * pip_value
    return lots, required_margin, expected_risk
```

## Stop distance rules (ATR-based defaults)

- ATR = ATR(14) in price units for current timeframe.
- Stop distance defaults by strategy type:
  - Tight (scalp/fade): 0.6 * ATR
  - Standard (breakout/retest): 1.0–1.5 * ATR
  - Structural reversal: 1.5–3.0 * ATR
- Always add half-spread and slippage buffer to stop_distance_effective:
  - stop_distance_effective = stop_distance + (spread/2) + slippage_buffer
- Enforce a minimum stop in pips to avoid micro-stops (e.g., min_stop_pips = max(3 * tick_size, 0.2 * ATR)).

Codable:

```
stop_pips = max(min_stop_pips, atr * stop_multiplier)
stop_effective = stop_pips + spread/2 + expected_slippage_pips
```

### Risk per trade and portfolio limits

Defaults and recommended hard limits (codable and configurable):

- Default risk_per_trade = 0.5% equity.
- Max risk per trade = 2.0% (hard cap).
- Max portfolio risk (sum of worst-case losses if all open trades hit stops): max_portfolio_risk_pct (e.g., 5%).
- Max concurrent open trades: 3–10 depending on account size and correlation controls.
- Max daily loss (kill switch): e.g., 3% equity — disable new trades for the day when exceeded.
- Max drawdown hard stop: e.g., 15% equity — trigger manual review/stop trading.

Implement checking:

- before opening trade, compute potential_portfolio_risk = current_open_risk + new_trade_risk
- if potential_portfolio_risk > equity * max_portfolio_risk_pct → reduce or skip trade.

### Margin-aware sizing and leverage

- Compute margin requirement per lot precisely using instrument contract size and broker margin rules.
- For FX standard lot (100,000 units): notional = lots * 100,000 * quote_currency_conversion_rate
- margin_required = notional * margin_rate
- Free_margin = equity - used_margin
- When leverage is tight, reduce lot size to fit margin limits rather than increasing stop distance.

Codable example (simplified for USD-base or USD-quote pairs):

```
notional = lots * contract_size
required_margin = notional * margin_rate
```

- For cross-currency pairs convert notional into account currency using current FX rates.

### Scaling in/out and pyramiding rules

- Allowed only if risk budgeting accounts for full aggregated exposure and ensures worst-case risk per tranche capped.
- Example pyramid rule (codable):
    - Max pyramids = 2

    - Add size = previous_size * 0.5
    - Additional entry only after price moves further in favor by ATR * 0.8 and new stop is trailed to breakeven+buffer
    - Recompute total worst-case risk after each add; do not exceed portfolio caps.

### Stop management and trailing rules

- Static stop: fixed ATR-based distance.
- Break-even move: move stop to entry + spread/2 after price moves favorably by X*ATR (e.g., 1.0*ATR) but only if doing so does not remove meaningful profit potential.
- ATR-trailing stop: trail stop at N * ATR behind highest favorable price (N default 0.8).
- Time-based stop: if trade hasn't achieved N*ATR adverse or favorable move in M days, exit (avoid stale trades).
- Always model transaction costs when adjusting stops (spread when stop becomes market order).

Codable trailing:

```
if peak_price - entry_price >= atr * breakeven_trigger:
    new_stop = max(current_stop, entry_price + spread/2)
if peak_price - entry_price >= atr * trail_trigger:
    new_stop = peak_price - trail_multiplier * atr
```

### Position correlation and portfolio risk aggregation

- Compute correlation matrix across instruments using returns on chosen timeframe.
- Adjust portfolio risk for correlation using approximate formula:
    - portfolio_risk ≈ sqrt(sum_i sum_j w_i w_j cov_ij)
    - where w_i = expected_percent_risk_i / equity in dollar terms normalized
- Simpler heuristic: cap exposure to highly correlated positions (e.g., max two positions in same currency pair family).
- Codable rule: disallow new long on EURUSD if EUR crosses already have combined notional above threshold.

### Drawdown control and recovery mode

- Track peak equity and current drawdown % = (peak_equity - current_equity)/peak_equity.
- When drawdown exceeds soft limit (e.g., 10%), reduce risk_pct by half. At hard limit (e.g., 15%), stop new trading and require manual review.
- Codable state machine: NORMAL → REDUCED_RISK → STOP_TRADING (based on drawdown thresholds).

### Order sizing with discrete contracts, pip value, and currency conversions

- Pip_value_per_lot depends on pair and account currency. Compute pip value precisely:
  - For USD-quoted pairs (e.g., EUR/USD), pip_value ≈ 10 USD per standard lot.
  - For USD-base pairs or non-USD accounts convert via FX rate.
- Always compute pip_value dynamically using current FX rates when account currency ≠ quote currency.

Codable calculation:

```
if quote_currency == account_currency:
    pip_value = contract_size * pip_in_price_units / 1.0
else:
    pip_value = contract_size * pip_in_price_units * fx_rate_quote_to_accou
```

### Practical Python snippet (pandas-friendly, vectorizable)

```
# inputs: equity, risk_pct, atr, stop_mult, spread_pips, slippage_pips, pip
stop_pips = max(min_stop_pips, atr * stop_mult)
stop_effective = stop_pips + spread_pips/2 + slippage_pips
risk_money = equity * risk_pct
raw_lots = risk_money / (stop_effective * pip_value)
lots = (np.floor(raw_lots / lot_step) * lot_step).clip(min=min_lot, max=max
required_margin = lots * margin_per_lot
if required_margin > free_margin:
    lots = (np.floor((free_margin / margin_per_lot) / lot_step) * lot_step)
    if lots < min_lot: lots = 0
```

(Implement as pure function returning zero lots if trade impossible.)

### Stress testing and robustness checks

Run these tests on all sizing rules:

- Spread shock: multiply spread by 2x and 5x; measure effect on win rate & expectancy.
- Volatility shock: double ATR and recompute stops/sizes; check margin usage.
- Batch risk test: open N worst-case trades and verify portfolio_risk <= max_portfolio_risk_pct.
- Monte Carlo of slippage and fills to assess tail risk.

### Automated risk controls and kill switches (codable state machine)

- Hard kill switches (immediate stop accepting any new trades):
  - max_daily_loss exceeded
  - equity drawdown > hard_limit

    - system_health_flag = false
- Soft limits (reduce risk_pct or stop new entries):
    - rolling 7-day loss > threshold
    - too many consecutive losing trades (e.g., 7)
- Implement idempotent transition functions:

```
if drawdown_pct > hard_limit: state = 'STOP'
elif drawdown_pct > soft_limit: state = 'REDUCED_RISK'
else: state = 'NORMAL'
```

### Examples (numeric)

Example A — 4H trade sizing:

- equity = $50,000; risk_pct = 0.005 → risk_money = $250
- ATR = 0.0040 (40 pips); stop_mult = 1.25 → stop_pips = 50
- spread = 1.2 pips; slippage = 0.5 pips → stop_effective = 50 + 0.6 + 0.5 ≈ 51.1 pips
- pip_value = $10 → raw_lots = 250 / (51.1 * 10) ≈ 0.489 → round down to 0.4 (lot_step 0.1)
- required_margin, check free_margin before placing order.

Example B — margin-limited account:

- If free_margin insufficient for computed lots, recompute lots by free_margin / margin_per_lot with same lot_step constraint; if below min_lot abort trade.

### Implementation checklist (codable)

- Implement pure functions for:
    - pip_value calculation
    - stop_distance computation with spread/slippage
    - lot sizing with discretization and margin checks
    - portfolio risk aggregation
    - state machine for risk modes (NORMAL/REDUCED/STOP)
- Log every sizing decision with inputs and outputs for reconciliation.
- Unit tests:
    - test edge cases (very small ATR, huge ATR, zero free margin)
    - test discrete rounding and margin constraints
    - test state transitions for kill switches

### Common mistakes and mitigations

- Using notional-based sizing without stops → can’t measure risk. Always compute stop first.
- Ignoring spread/micro-stops → stops too tight and frequently hit. Add spread buffer.
- Not accounting for correlated positions → underestimation of portfolio tail risk. Implement simple correlation caps.

- Overly complex pyramiding rules that blow up risk. Limit pyramiding and require explicit worst-case aggregation.

## Exercises (codable)

1. Implement compute_lots() function handling currency conversion, pip value, discrete lot sizes, and margin checks; unit-test with USD-account on EURUSD and USDJPY.
2. Simulate 1000 randomized trades with ATR-distributed stops and compute max concurrent margin and worst-case portfolio drawdown; verify limits enforced.
3. Implement state machine that halves risk_pct when drawdown > 10% and stops trading at 15%; backtest historical equity curve to validate behavior.

## Summary — prescriptive defaults

- ATR(14) for stops; default stop_mult 1.0–1.5 for standard trades.
- Default risk_per_trade = 0.5% (0.005), max risk_per_trade = 2.0%.
- Max portfolio worst-case risk = 5% (configurable).
- Max concurrent trades = 5 (adjust for account size and correlation).
- Kill switches: daily_loss = 3%, hard_drawdown = 15%.

Implement these as configurable parameters, enforce deterministically in code, and validate with margin-aware backtests and stress tests.

# Margin Management and Leverage

## Overview

Concrete, codable guidance for managing margin and leverage for automated Forex trading on 4H and 1D timeframes. Covers margin calculations, instrument-specific contract math, cross-currency conversions, intraday vs overnight margin considerations, margin-related risks (margin calls, forced liquidation), leverage sizing rules, stress tests, and Python-ready formulas. Defaults: UTC candles, ATR(14) for volatility sizing, risk per trade = 0.5%.

## Key concepts (decisive definitions)

- Notional:
  - contract value = lots × contract_size × price.
  - Standard FX lot contract_size = 100,000 base currency units.
- Margin requirement:
  - margin = notional × margin_rate (broker-specified).
  - margin_rate = 1 / leverage (e.g., 1% for 100:1).
- Free margin: free_margin = equity − used_margin.
- Used margin / required margin: sum of margins for all open positions.
- Maintenance margin: broker-specific threshold below which positions may be liquidated.
- Leverage: ratio of notional exposure to equity. Effective leverage = total_notional / equity.
- Margin utilization: used_margin / equity.
- Margin call / liquidation: triggered when equity falls to or below maintenance threshold.

## Exact contract math (codable formulas)

Inputs per instrument:

- lots
- contract_size (e.g., 100,000)
- price (current mid/ask/bid depending on side)
- margin_rate (e.g., 0.01 for 100:1)
- account_currency and conversion FX rate (if needed)

Compute notional in base currency:

- notional_base = lots × contract_size

Compute notional in account currency:

```
if account_currency == quote_currency:
  notional_account = notional_base × price
else if account_currency == base_currency:
  notional_account = notional_base
else:
  notional_account = notional_base × price × fx_quote_to_account_rate
```

Compute required margin in account currency:

- margin_required = notional_account × margin_rate

Compute used/total margin for portfolio:

- used_margin = sum(margin_required_i for open positions)

Compute free_margin:

- free_margin = equity − used_margin

Compute effective leverage:

- effective_leverage = (sum(notional_account_i) / equity)

Codable snippet:

```
def compute_margin(lots, contract_size, price, margin_rate, fx_quote_to_acc
    notional_base = lots * contract_size
    notional_account = notional_base * price * fx_quote_to_acc
    margin_required = notional_account * margin_rate
    return notional_account, margin_required
```

### Pip value, stop distance, and margin interactions

- Position sizing depends on pip value; pip value depends on pair and account currency. Always compute pip_value dynamically (see Chapter 12).
- When margin is constrained, prefer reducing lots rather than widening stops; widening stops increases expected loss per trade.
- Before placing an order, check required_margin ≤ free_margin_threshold (e.g., free_margin_threshold = 95% of free_margin to leave buffer).

Codable pre-trade check:

```
notional, required_margin = compute_margin(lots, ...)
if required_margin > free_margin * allowed_margin_usage_fraction: # e.g., 0
    reduce_lots_or_abort()
```

## Leverage policy (rules to implement)

- Conservative target effective leverage: 5–20× (i.e., 5–20 times equity). Default target: ≤ 10× for retail automated strategies.
- Hard cap: maximum effective leverage configurable (e.g., 50×) to prevent catastrophic exposure.
- Use margin utilization cap: e.g., used_margin ≤ equity × 0.25 (25%) as default (adjust for account size).
- Enforce per-instrument exposure caps: e.g., no more than X% equity in any single currency pair notional (e.g., 50% notional cap).

Codable enforcement:

```
if effective_leverage > max_leverage: abort_trade()
if used_margin / equity > margin_utilization_cap: abort_trade()
```

## Intraday vs Overnight margin rules

- Brokers may require higher overnight margin (margin_rate increases) or charge rollover swaps that can materially change P&L. Implement time-aware margin modeling:
  - Use session-based margin_rate table keyed by instrument and timestamp.
  - For trades opened near rollover window (e.g., 23:00–01:00 server time), compute potential overnight swap cost and higher margin requirement.
- For backtests and sizing, model margin_rate as function of hold_time:
  - intraday_margin_rate = base_margin_rate
  - overnight_margin_rate = base_margin_rate * overnight_multiplier (e.g., 1.0–3.0 depending on broker)
- Avoid opening trades that will temporarily exceed margin limits after scheduled rollover or corporate events.

Codable approach:

```
margin_rate_at_open = margin_rate_table.get(timestamp)
estimate_swap_costs = days_held * swap_rate_per_day * notional_account
```

## Margin calls and liquidation modeling

- Simulate liquidation by defining maintenance_margin_pct. If equity ≤ used_margin × maintenance_factor, assume forced close of positions (in worst case, all positions).
- Conservative modeling for backtests: apply forced liquidation at maintenance threshold with slippage penalty and loss of partial positions.
- Implement soft stop behavior in live bot: prefer orderly reductions (close smallest positions or highest-risk) when equity approaches threshold rather than immediate mass close, but ensure deterministic rules for automation.

Codable liquidation test:

```
if equity <= used_margin * maintenance_threshold:
    trigger_liquidation_sequence()
```

Liquidation sequence example:

- Sort open positions by risk (dollar risk) descending
- Close positions until used_margin < equity * safe_margin_factor
- Apply slippage and spread to closure prices

## Leverage stress-testing and scenario analysis

Simulate these scenarios: - Price shock: simulate 5σ adverse move on top 3 correlated positions; compute margin_required and resulting equity. - Spread widening: increase spread by ×2–5 and recompute stops and margin usage. - Rollover spike: apply higher overnight margin and swap rates and evaluate effect on used_margin and expected carry P&L. - Multiple margin-using trades: open worst-case set of trades (per portfolio caps) and compute maximum drawdown if stops hit sequentially.

Codable Monte Carlo:

- Randomize price moves using empirical return distribution, apply slippage and spread distributions, compute margin and equity trajectories, flag runs causing liquidation.

## Practical margin-aware sizing patterns

- Margin-first sizing: given desired risk_money and stop_distance, compute raw_lots then check margin; if margin insufficient, scale down lots to margin-limited size.
- Margin-first conservative approach: limit target lots to free_margin × margin_utilization_cap / margin_per_lot before computing final lot from risk.
- Reserve buffer: require a free_margin_buffer (e.g., 10–20% of equity) to account for intraday move and avoid execution failures.

Pseudocode:

```
max_lots_by_margin = floor((free_margin * allowed_usage) / margin_per_lot)
lots_by_risk = floor((risk_money / (stop_effective * pip_value)) / lot_step
lots = min(max_lots_by_margin, lots_by_risk)
```

## Cross-currency and conversion edge cases

- For non-USD account currencies, convert notional and margin using up-to-date FX rates; be careful with crossed pairs where account currency appears neither as base nor quote.
- Example: account in USD trading EUR/JPY:
    - notional_account = lots * 100,000 * price_EURJPY * FX_JPY_to_USD (or compute via EURUSD & USDJPY combinations)

- Use live quote conversion or recent mid rates; in backtests use historical conversion rates aligned to candle timestamp.

Codable conversion:

```
if account_currency not in (base, quote):
    use cross_rate = get_fx_rate(quote -> account_currency) at timestamp
    notional_account = lots * contract_size * price * cross_rate
```

## Leverage and margin policy for automation (recommended defaults)

- Default leverage offered by broker ≠ leverage used. Programmatic policy:
  - target_effective_leverage = 8–10×
  - max_effective_leverage = 20×
  - margin_utilization_cap = 30% (used_margin ≤ 30% equity)
  - minimum_free_margin_buffer = 10% equity
- Hard stops: if used_margin / equity > 50% → stop opening new trades and reduce risk dynamically.

## Practical examples (numeric)

Example A — USD account, EURUSD long:

- equity = $50,000
- margin_rate = 0.01 (100:1)
- price = 1.1200
- lots = 1.0 → notional_base = 100,000
- notional_account = 100,000 × 1.1200 = $112,000
- margin_required = 112,000 × 0.01 = $1,120
- used_margin if this is only trade = $1,120
- effective_leverage = 112,000 / 50,000 = 2.24×

Example B — cross-currency conversion (account USD, trading EURJPY):

- price_EURJPY = 140.00
- lots = 1.0 → notional_base = 100,000 EUR
- convert EUR to USD: need EURUSD rate at timestamp, say 1.1200 → notional_account = 100,000 × 1.1200 = $112,000
- margin_required = 112,000 × margin_rate

## Operational rules for live automation (codable checklist)

- Maintain up-to-date margin_rate table per instrument and timestamp.
- Compute margin-required before sending orders; abort if margin constraints violated.
- Reserve free_margin_buffer and do not consume all free margin.
- Implement pre-trade simulations for worst-case stop hit scenario and ensure portfolio worst-case loss ≤ max_portfolio_risk_pct.

- Tie risk-state machine (from Chapter 12) to margin checks: reduced risk when used_margin / equity > threshold.
- Log margin calculations and decisions for audit.

### Failed-trade and emergency procedures

- If order rejected due to insufficient margin: do not retry indefinitely; reduce lot size deterministically and retry once.
- If margin call imminent: implement automatic de-risking:
    - close newest/highest-risk positions first or
    - scale down positions proportionally to restore required margin.
- If forced liquidation occurs: record event, pause trading, and send alerts for manual review.

Codable rule:

```
if order_rejected_margin:
    lots = max(floor(lots * retry_scale, lot_step), 0)
    if lots >= min_lot: retry_once()
```

### Stress test checklist (codable)

Run periodically and before deployment: - Price shock tests: -5%/-10% on top correlated positions, check resulting liquidations. - Spread shock tests: ×2 and ×5 - Rollover spike: double overnight margin and recompute positions - Correlation concentration: open worst-case correlated basket per policy and test margin_utilization

Record pass/fail and store in system health logs.

### Common mistakes and mitigations

- Confusing broker leverage offered with safe operational leverage — use conservative internal leverage policy.
- Forgetting currency conversion for margin — always convert notional to account currency.
- Over-consuming margin leaving no buffer for spikes — enforce free_margin_buffer.
- Assuming margin stays constant across session/overnight — use time-aware margin tables.

### Implementation notes (Python)

- Keep margin and contract-size metadata in a configuration table keyed by instrument.
- Provide pure functions:
    - compute_notional_and_margin(lots, price, instrument, account_currency, timestamp)
    - adjust_lots_for_margin(target_lots, free_margin, margin_per_lot, lot_step)

- Maintain live state of used_margin and free_margin in engine; recompute after fills and mark-to-market on each candle.
- Test edge cases: very small accounts, large notional instruments (crosses), and instruments with odd contract sizes.

## Exercises (codable)

1. Implement compute_margin() and test for EUR/USD, USD/JPY, EUR/JPY with account in USD using historical FX rates.
2. Simulate a portfolio of 5 positions with given lots and prices; compute used_margin, free_margin, effective_leverage, and test policy enforcement when equity drops 10%.
3. Run a Monte Carlo with random adverse moves and spread widening; count runs causing forced liquidation under current policy.

## Summary — prescriptive defaults

- Maintain conservative operational leverage (target 8–10×, hard cap 20×).
- Enforce margin_utilization_cap ≈ 30% and free_margin_buffer ≈ 10% equity.
- Compute margin per-instrument with up-to-date FX conversions.
- Model time-of-day / overnight margin_rate and swap costs.
- Implement deterministic de-risking procedures and logging for all margin events.

# Costs of Trading

## Overview

Precise, codable modeling of all trading costs for automated Forex systems on 4H and 1D timeframes. Include spread, explicit commissions, slippage, rollover/swap (carry), financing, market impact approximations, latencies, and discrete-lot effects. Provide formulas, defaults, stress tests, and backtest integration guidance so P&L and edge estimates are realistic. Defaults: UTC candles, ATR(14) for volatility normalization.

## Cost categories (decisive list)

- Spread (bid/ask differential) — implicit cost on every round-trip.
- Commission — explicit per-lot or per-side fees charged by broker.
- Slippage — execution deviation from intended fill price (adverse or favorable).
- Rollover / Swap / Financing — daily carry cost for positions held overnight.
- Market impact (for large orders) — price move caused by order size relative to market liquidity.
- Latency-related costs — missed fills, worse fills due to delayed order placement/cancellation.
- Taxes/fees and exchange-specific charges — less common for FX in retail but include where applicable.

Model all these in backtests and live P&L accounting.

## Spread modeling

- Use instrument- and time-of-day-dependent spread tables. Example: EURUSD spread smaller during London session, wider overnight.
- If only mid-price OHLC available, convert to bid/ask:
    - ask = mid + spread/2; bid = mid − spread/2
- For buy entries: assume fill at ask; for sell entries: fill at bid.
- For stops and limit orders: evaluate against bid/ask appropriately (stop for long triggers at bid; short stop triggers at ask).
- Represent spread as deterministic or stochastic (e.g., base_spread × (1 + noise)), and stress-test with multiplied spreads (×2, ×5).

Codable:

```
spread_pips = spread_table[instrument][hour_of_day]
ask = mid + spread_pips/2 * pip_value_in_price_units
bid = mid - spread_pips/2 * pip_value_in_price_units
```

## Commission handling

- Commission types:
  - Per-side per-lot fixed (e.g., $3 per side per standard lot)
  - Per-round-trip flat fee
  - Spread-inclusive (no explicit commission)
- Model commissions as immediate cash cost on fill.
- Apply commission to both entry and exit.
- For performance metrics, subtract commissions from trade P&L to compute net returns.

Codable:

```
commission_entry = lots * commission_per_lot_per_side
commission_exit = lots * commission_per_lot_per_side
net_pnl = gross_pnl - commission_entry - commission_exit - other_costs
```

## Slippage modeling

- Slippage sources: queue latency, partial fills, price movement between signal generation and order arrival.
- Model slippage as:
  - deterministic fixed ticks (e.g., 0.5 pip)
  - stochastic variable: Slippage ~ Normal(mu=adverse_bias, sigma = k * ATR) truncated to realistic bounds
  - fill-probability model for limit orders: probability of fill p = f(aggressiveness, liquidity); if not filled within timeout, fallback to market with slippage.
- Conservative default: slippage = max(0, Normal(μ=0.2*pip, σ=0.5*pip)) adverse on average.
- For large orders relative to liquidity, model slippage scaling with order_size / average_daily_volume_metric.

Codable limit-fill logic:

```
if order_type == 'limit':
    fill_prob = sigmoid(aggressiveness)  # e.g., based on depth
    if random() < fill_prob:
        fill_price = limit_price + slippage_sample
    else:
        after_timeout: fill_price = market_price + market_slippage_sample
```

## Rollover / Swap / Financing

- Swap formula depends on interest-rate differential and broker markup:
  - swap = notional_base × (rate_base - rate_quote) × day_fraction − broker_markup

    - brokers often provide per-lot overnight swap rates; use historical swap table aligned to timestamp.
- Apply swap daily at broker’s rollover time (or on weekends include triple swap on Wednesday depending on broker).
- For multi-day 4H/1D trades, include swap in expected P&L and in position holding cost calculations; for short (~10-day) horizon swap can be material for crosses with high rate differentials.

Codable:

```
swap_per_day = lots * swap_rate_per_lot_per_day  # from broker table
total_swap = swap_per_day * days_held
```

Backtest: subtract swap per open day from daily P&L; ensure triple-swap rules applied historically.

### Market impact approximations

- For retail FX often negligible for typical lot sizes, but include model for large accounts:
    - impact = impact_coefficient × (order_notional / ADV)^impact_power
    - typical power = 0.5; coefficient calibrated from liquidity assumptions.
- Model price slippage during aggressive market entry as sum of execution slippage + market impact.
- Use conservative cap: if order_size > liquidity_threshold → scale down lots or use iceberg/two-step entry.

Codable:

```
impact = impact_coeff * (notional_account / adv) ** 0.5
expected_fill_price = intended_price + sign * impact
```

### Latency and missed fill costs

- Latency cost arises from delay between signal generation and order placement.
- Measure system latency (ms) and model expected price movement during that latency using short-term volatility estimate:
    - expected_move = sqrt(latency_seconds / period_seconds) * ATR_per_period
- For 4H/1D systems, latency usually less material, but for limit entries near tight levels latency can cause misses.
- Implement timeouts for limit orders and re-evaluation logic for stale signals.

Codable expected latency slippage:

```
latency_slippage_pips = sigma_per_second * sqrt(latency_seconds)  # sigma d
```

### Combining costs into per-trade cost model

Net P&L per trade = gross_price_move_in_pips × pip_value × lots − (spread_cost + commission_total + slippage_cost + swap_total + market_impact_cost)

Where:

- spread_cost = spread_pips × pip_value × lots (round-trip includes spread on entry & exit if using mid-based fills)
- commission_total = commission_entry + commission_exit
- slippage_cost = slippage_entry + slippage_exit (modeled)
- swap_total = sum(overnight_swap_per_day)
- market_impact_cost = modeled impact as above

Codable function signature:

```
def compute_trade_net_pnl(entry_price, exit_price, lots, pip_value, spread,
    gross_pips = (exit_price - entry_price) / pip_size * sign
    gross_pnl = gross_pips * pip_value * lots
    costs = (spread_cost + commission + slippage_entry + slippage_exit + sw
    return gross_pnl - costs
```

### Backtest integration best practices

- Integrate all costs at the time of simulated fills; do not apply them post-hoc as a flat percentage.
- Use realistic fill logic:
    - For market orders: fill at next candle’s bid/ask with slippage sample.
    - For limit orders: apply fill probability and possible partial fill across next N candles.
- Keep cost parameters time-series-aware: spreads, swap rates, and liquidity change over time and by session.
- Run cost sensitivity and stress tests:
    - baseline, spread*2, spread*5, slippage*2, swap shocks.
- Report results both gross and net of all costs.

### Stress testing costs

- Run scenarios:
    - Spread widening events (e.g., news): randomly choose N% of trades to have spread ×k.
    - Slippage tail events: add heavy-tailed slippage distribution for a fraction of fills.
    - Carry shock: sudden interest rate moves changing swap rates.
    - Combined microstructure shock: spread ×5, slippage heavy-tail, and temporary liquidity drop for correlated pairs.
- Measure effect on expectancy, Sharpe, and worst drawdown. Ensure strategy still robust or reduce frequency/stop tightening.

## Recording costs and reconciliation

- Record every cost item per trade in logs: spread, commission, slippage_entry, slippage_exit, swap_total, impact_cost.
- Reconcile live broker statements daily: verify realized commissions and swaps match modeled values within tolerance; adjust model parameters if persistent bias observed.
- Maintain versioning of cost models with timestamps used for backtests.

## Practical default parameters (starting points)

- Spread: use historical average by instrument & hour; if unknown use conservative defaults:
  - Major pairs (EURUSD, USDJPY): 0.6–1.5 pips (4H/1D)
  - Minor/exotics: 2–10+ pips
- Commission: $3–7 per side per standard lot (or 0 if spread-inclusive)
- Slippage: mean adverse slippage 0.2–0.7 pips; sigma 0.5–2.0 pips depending on liquidity
- Swap: use broker table; if unknown estimate from interest differential (small for majors, larger for exotic carries)
- Market impact: impact_coeff calibrated; set to 0.01 for small retail, higher for institutional-size orders

Always stress-test with multiples of these defaults.

## Example computation (numeric)

Trade: EURUSD long, 0.5 lot, entry 1.1200, exit 1.1250 (50 pips move), pip_value = $5 (per 0.5 lot):

- gross = 50 pips * $5 = $250
- spread = 1.2 pips → round-trip spread cost ≈ 1.2 * $5 = $6
- commission = $3 per side * 0.5 lot proportion → $3 total (assume $3 round-trip for simplicity)
- slippage_entry + exit = 0.5 + 0.5 pips = 1.0 pip → $5
- swap_total for 3 days = $2
- net_pnl = 250 − (6 + 3 + 5 + 2) = $234

## Implementation checklist (codable)

- Maintain time-indexed tables for spread_by_hour, swap_rates, commission_specs, liquidity_metrics.
- Implement fill simulator that consumes order requests and returns fills with:
  - fill_price, filled_lots, slippage, commission charged, timestamp, reason (limit/market/partial)
- Include flags for stressed fills and events (news spread widening).
- Log cost breakdown per trade for analysis and reconciliation.

- Include unit tests for cost model edge cases (zero spread, extremely high spread, partial fills).

## Common mistakes and mitigations

- Applying flat cost per trade post-hoc — instead compute costs at fill time.
- Ignoring time-of-day spread variation — use hour/session sensitive spreads.
- Forgetting swap costs for multi-day holds — include per-day swap charges.
- Using optimistic slippage assumptions — calibrate from live fills and be conservative in backtests.

## Exercises (codable)

1. Build a fill simulator that models spread (hourly), slippage (stochastic), and commission; run on historical 4H EURUSD signals and compare gross vs net P&L.
2. Implement swap calculation for a basket of currency pairs using historical central bank rates; calculate total swap over a 10-day hold.
3. Stress test strategy under spread*5 and heavy-tail slippage for 1,000 randomized trades; produce summary metrics (expectancy, max drawdown).

## Summary — prescriptive defaults

- Model spread per instrument and session; convert mid to bid/ask before fills.
- Apply commissions at fills, model slippage stochastically with adverse bias, include swap per day for overnight holds.
- Integrate market impact and latency models for large orders and tight limit entries.
- Stress-test cost multipliers and use conservative cost assumptions in sizing and expected-edge calculations.
- Log and reconcile costs daily to calibrate models.

# Trading Psychology

## Overview

Practical, codable controls and operational procedures to prevent human behavioral errors when running automated Forex systems on 4H and 1D timeframes. Focus: operator decision biases, monitoring/alerting, rule-enforcing automation, post-trade review discipline, and embedding safeguards in code to minimize human-caused drift. Provide prescriptive actions, state-machine controls, and checklist items.

## Core problem statement (decisive)

Automated systems still require human operators. Most losses stem not from model math but from human interventions (overriding rules, changing parameters impulsively, misinterpreting drawdown). Embed procedural and technical safeguards so behaviorally-driven errors are minimized and auditable.

## Common behavioral pitfalls and deterministic mitigations

- Overtrading / revenge trading
  - Pitfall: increasing position size after losses or forcing more trades to recover.
  - Mitigation: hard cap on trades per day and per-loss-sequence; automated halting rule:
    - if consecutive_losses >= N_loss_limit → set state = REDUCED_RISK or PAUSE_TRADING for M hours/days.
  - Log and require manual review before state cleared.
- Parameter tinkering / curve-fitting
  - Pitfall: changing parameters after seeing recent performance (data fishing).
  - Mitigation: enforce parameter-change policy:
    - Only allow parameter changes if A/B testing/walk-forward shows improvement with at least X% out-of-sample improvement and Y months of forward live-sim testing.
    - Parameter changes must be versioned, flagged, and require two-person approval in logs.
- Ignoring risk rules
  - Pitfall: overriding risk_pct, stop sizes, or margin checks.
  - Mitigation: make risk parameters immutable without multi-step approval; require passphrase or offline approval to change critical limits.
- Overconfidence after wins
  - Pitfall: increasing risk after runs of wins.
  - Mitigation: enforce a policy that risk_pct can only be increased after at least Z consistent months of positive risk-adjusted returns and re-validation.
- Premature shutdown or reactivation
  - Pitfall: turning system off during drawdown, losing long-run edge.

    - Mitigation: require documented incident report and checklist before pausing or restarting live trading; automated alarms plus mandatory review steps.
- Selective memory (survivorship bias)
    - Pitfall: cherry-picking only recent winning trades as evidence.
    - Mitigation: require system reports to show gross & net, full trade history, and stress-test results before decisions.
- Confirmation bias and narrative fallacy
    - Pitfall: crafting stories to justify changes.
    - Mitigation: rely on quantitative triggers and objective metrics to allow rule changes (no free-text override without recorded rationale).

## Codable governance & controls

- Immutable configuration with versioning
    - Store config in version-controlled repository (git) with timestamped releases used by live system.
    - Automatic deployment only from tagged commits after automated tests pass.
- Multi-tier approval for critical changes
    - Changes to risk_pct, max_drawdown, margin caps, or kill-switch thresholds require two independent signatures (or two API tokens).
- Lockable parameters and admin roles
    - Implement roles: operator, reviewer, admin. Only admin can change certain parameters; reviewer must sign off.
- Audit logging (mandatory)
    - Log every parameter change, manual order, manual cancel, system pause/resume with user ID, timestamp, rationale.
    - Keep immutable append-only logs for reconciliation.
- Automated behavioral guards (code)
    - Hard-stop rules as state machine (NORMAL → REDUCED_RISK → PAUSE → STOP) triggered by quantitative events (drawdown, daily loss, consecutive losses).
    - Example:

```
if daily_loss_pct > DAILY_LOSS_LIMIT: state = 'PAUSE'; notify_team(
if consecutive_losses >= LOSS_SEQ_LIMIT: state = 'REDUCED_RISK'
if global_drawdown >= HARD_LIMIT: state = 'STOP_TRADING'
```

    - Ensure manual override requires multi-step confirmation and is logged.

## Monitoring, alerts, and dashboards

- Real-time metrics to monitor (minimum set)
    - equity, used_margin, free_margin, open_positions_count, open_risk_$, realized_P&L_today, unrealized_P&L, consecutive_wins/losses, drawdown_from_peak, latency_ms, failed_orders_count.

- Alerting tiers
    - Info: daily summary email
    - Warning: threshold exceeded (e.g., used_margin > 25% equity)
    - Critical: immediate alert requiring acknowledgment (e.g., equity drawdown > soft_limit, failed order > x)
- Alert delivery: multiple channels — SMTP, Slack/Teams, SMS for critical. Require human acknowledgment and automated escalation if unacknowledged.
- Dashboard features: immutable recent events list, parameter version, last deploy hash, current state (NORMAL/PAUSE/STOP), and link to audit logs.

## Reporting discipline and post-trade review

- Daily automated report must include:
    - trades executed (gross & net cost breakdown), P&L by strategy, realized vs expected slippage, fill rates, number of cancelled/partial fills, system health metrics.
- Weekly review checklist:
    - Verify configuration unchanged in week
    - Run automated cost-reconciliation (modeled vs actual)
    - Review any manual intervention logs
- Monthly strategic review:
    - Walk-forward performance review, stress-test recalibration, parameter-change proposals with supporting OOS tests.
- Template for incident report: time, symptom, immediate action taken, root-cause hypothesis, mitigation, follow-up actions, approval.

## Psychological safety and team processes

- Prevent single-actor critical decisions:
    - Major changes require at least two approvers.
    - No single person should be able to pause and resume trading in one action without a review window (e.g., can pause but resume requires second sign-off).
- Blameless postmortems:
    - After critical incidents, run structured blameless postmortems with factual timelines and code/log artifacts.
- Rotating on-call duties and documented runbooks:
    - On-call person follows runbook for common incidents; runbook scripted and versioned.

## Embedding human constraints in automation (codable examples)

- Immutable risk guard:

```
@requires_two_approvals
def change_risk_pct(new_pct):
```

```
    if new_pct > max_allowed or new_pct < min_allowed: reject()
    record_change()
```

- Manual override audit:

```
def manual_close(position_id, user_id, reason):
    require_reason(reason)
    append_audit_log(user_id, reason, position_snapshot)
    execute_close(position_id)
```

- Auto-lock after manual intervention:
    - Any manual trade or parameter change triggers automatic system lock until weekly review or second approver unlocks.

### Behavioral metrics to track (quantitative)

- Intervention rate = manual_actions / automated_trades
- Decision latency = time between alert and human acknowledgment
- Parameter-change frequency per month
- Reversion ratio = fraction of manual overrides reversed within 7 days
- Trade-execution failure rate pre/post manual interventions

Use these to quantify and reduce human-created risk.

### Training and checklists for operators

- Pre-deployment checklist:
    - Verify latest config tag deployed
    - Run smoke tests on paper account for 24–72 hours
    - Check margin model and pip values
    - Confirm alerts and contacts configured
- Incident checklist:
    - Pause new entries
    - Notify team and stakeholders
    - Gather logs and last-100 events
    - If trading resumes, do so from a known-good tagged config and document reasons
- Re-activation checklist (after pause):
    - Confirm root cause addressed
    - Run 24h paper simulation with same market conditions if possible
    - Obtain required approvals documented in audit log

### Practical anti-bias rules (codable policies)

- No-increase-after-loss policy: prevent increasing risk_pct after N consecutive losses; implemented automatically.

- Cooldown timer after manual override: after any manual change, disable new trades for T minutes/hours.
- Require evidence for parameter change: automated report that must include OOS backtest and walk-forward results with p-values or robust metrics.

### Example state-machine for behavioral control (simple)

States: NORMAL, REDUCED_RISK, PAUSE_TRADING, STOP_TRADING

Transitions:

- NORMAL → REDUCED_RISK when consecutive_losses >= 4 or drawdown > 7%
- REDUCED_RISK → PAUSE_TRADING when daily_loss_pct > 3% or drawdown > 10%
- PAUSE_TRADING → NORMAL only after manual review and documented approval
- Any state → STOP_TRADING when drawdown >= 15% or critical system failure

All transitions logged with timestamp, cause, and actor (system or human).

### Exercises and operational drills

- Run a simulated incident drill monthly: inject failure (e.g., large drawdown or halted market feed) into paper system and practice runbook steps.
- Compute intervention_rate for the past 6 months and target reduction of 50% by automating manual tasks and improving thresholds.
- Implement a two-person approval workflow in configuration management and test that a single operator cannot resume trading alone.

### Summary — prescriptive controls to implement now

- Immutable versioned configs; code-deployed only from tagged commits.
- Two-person approval for critical parameter changes.
- Automated kill switches for daily loss, drawdown, consecutive losses.
- Mandatory audit logs for all manual actions.
- Cooldown and lockout after manual interventions.
- Real-time alerts and required acknowledgments plus runbooks.
- Regular drills and blinded backtests to reduce operator bias.

Implement these as code-first, auditable controls; behavioral discipline must be enforced by system design, not left to operator goodwill.

# Low-Frequency vs High-Frequency Trading

## Overview

This chapter contrasts low-frequency (LF) and high-frequency (HF) approaches, focusing on implications for automated Forex trading on 4H and 1D timeframes (primary audience). It covers strategy design, data needs, execution realism, infrastructure, risk management, backtesting requirements, and recommended practices for building reliable automated systems.

## 1. Definitions and scope

- **Low-Frequency Trading (LF):**
  - Trade rate: minutes to days between decisions (for this book: 4H and 1D candles, ~one trade per several days to weeks).
  - Strategy horizon: short-swing (~10-day) to multi-week.
  - Typical instruments: major/minor FX pairs; execution via retail/professional brokers.
  - Primary constraints: correct signal generation, robust position sizing, realistic transaction cost modeling (spread, slippage, rollover).
- **High-Frequency Trading (HF):**
  - Trade rate: sub-second to seconds/minutes; very high order throughput.
  - Strategy horizon: intraday, tick-level microstructure.
  - Requires colocation, direct market access (DMA), and specialized market data.
  - Not the focus of this book; HF is materially different in data, infrastructure, and regulatory needs.

## 2. Why LF is appropriate for 4H/1D, and HF isn’t

- 4H/1D signals aggregate noise, making structural patterns and trend signals more meaningful; latency of seconds is irrelevant.
- LF simplifies requirement set: historical OHLCV suffices, lower data storage, simpler execution models.
- HF requires tick-level data, order book depth, ultra-low-latency execution, and sophisticated market-making/latency arbitrage logic—out of scope for traders targeting 4H/1D horizons.

## 3. Data requirements

- LF (4H/1D):
  - Primary: cleaned OHLCV at 4H and daily UTC alignments.
  - Required fields: open, high, low, close, volume (and ideally bid/ask or mid+spread history).

    - Time span: multi-year for strategy discovery; prefer ≥3 years for robust backtests across regimes.
    - Data quality: fill missing candles, handle DST/holiday gaps, adjust for rollovers/ symbol continuity.
- HF:
    - Tick/level-2 order book, message timestamps, trades, quotes.
    - Impractical for 4H/1D goals.

## 4. Execution modeling: LF vs HF differences

- LF must model:
    - **Spread:** use either fixed pip spread per pair (from broker historical averages) or a time-varying spread series if available.
    - **Slippage:** model as percent of ATR or fixed pip distribution; for limit orders include probability of fill and adverse selection.
    - **Rollover/swaps:** apply overnight funding for positions held at Rollover time (typically 5pm New York / broker specific); include both positive and negative swaps.
    - **Order types:** deterministic candle-close decisioning with preferred limit entries and fallback to market orders at close (see Chapter 5).
- HF requires microstructure-aware costs: queue position, fill probability, maker/taker fees—different modelling.

Practical LF execution model (defaults):

- Decision tick: at candle close (UTC).
- Entry attempt: place limit at previous close ± entry offset; if not filled within next candle, convert to market at open of next candle.
- Slippage: sample from distribution proportional to ATR(14) (e.g., Normal(0, 0.1*ATR) clipped to sensible bounds) or apply fixed adverse slippage (e.g., 0.5–1.5 pips for majors).
- Spread: apply bid/ask half-spread to entry and exit prices (use historical median spread per pair).
- Rollover: apply daily swap rate pro rata for number of calendar days open.

## 5. Infrastructure and operational differences

- LF infrastructure (recommended, minimal):
    - Single server or cloud VM running scheduler (cron/airflow) at daily/4H cadence.
    - Data pipeline: periodic fetch + local store (parquet), daily resample, integrity checks.
    - Backtester: vectorized or event-driven (pandas/numpy OK).
    - Broker integration: REST/WebSocket for orders and position sync; idempotent state machine for lifecycle.
    - Monitoring: daily PnL reports, alerts for stale data, execution mismatches, large drawdowns.

- HF infrastructure (not required here) adds:
    - Co-location, kernel bypass, specialized languages (C++), complex risk engines.

## 6. Risk, margin, and capital considerations

- LF position sizing uses ATR, account equity, and percent risk per trade (default: 0.5% equity).
- Margin: account for initial margin and overnight requirements; simulate worst-case margin calls using stressed price moves.
- Diversification: limit correlated pair exposures; model portfolio margin if broker supports.
- HF uses different capital and margin dynamics (frequent intraday churn).

## 7. Backtesting and evaluation differences

- LF backtests:
    - Use OHLCV bars; ensure no lookahead (decisions based on closed bar data).
    - Model spread/slippage/rollover per trade.
    - Use walk-forward and out-of-sample splits, cross-validation across market regimes (bull/bear/flat).
    - Enough sample size: for 1 trade per 10 days, simulate multiple years to obtain statistical significance.
- HF backtests require tick-level replay, order book simulation, queue dynamics—complex and out of scope.

Practical LF backtest checklist:

- Align candle timezone to UTC.
- Use ATR(14) as volatility baseline.
- Default risk per trade: 0.5% equity.
- Deterministic rule execution at candle close.
- Include realistic transaction costs: spread + slippage + commission + rollover.
- Perform walk-forward (rolling) optimization for parameters and hold one or more years as out-of-sample.

## 8. Strategy design implications

- LF favors:
    - Trend and structural setups (breakouts, trend continuations, channel plays).
    - Mean-reversion on higher timeframe cycles (over weeks).
    - Event-driven swings (macro news, FOMC) with pre/post event handling and guardrails.
- HF favors:
    - Market-making, arbitrage, latency-sensitive microstructure plays—irrelevant for 4H/1D.

- For LF design, prefer:
    - Rules that produce fewer trades with higher information ratio.
    - Robust filters (ATR thresholds, minimum volatility, time-of-day filters).
    - Explicit trade durability assumptions (how long to keep a stop or target active).

## 9. Operational safety and deployment

- LF safety measures (practical checklist):
    - Paper trading phase: run identical logic against live prices for 30–90 days before real capital.
    - Kill switches: daily max loss (e.g., 3% equity), maximum concurrent trades, market-open blackout after major releases.
    - Watchdog: automatic pause on stale data, large execution slippage, or connectivity loss.
    - Reconciliation: log every order/quote/position; daily reconcile broker statements with internal PnL.
    - Alerting: immediate alerts for fills, large unexpected PnL moves, risk breaches.
- HF safety requires additional low-latency circuit breakers and regulatory controls.

## 10. Costs and business model

- LF cost drivers:
    - Spread and slippage dominate per trade; commissions less relevant for Forex with spread-based pricing.
    - Swap/rollover can materially affect multi-day holds and must be included in profitability models.
    - Hosting and compute costs are modest.
- HF cost drivers: exchange fees, connectivity, colocation costs—irrelevant to 4H/1D focus.

## 11. Performance evaluation and expected metrics for LF systems

- Key LF metrics:
    - Annualized return, annualized volatility, Sharpe ratio (use risk-free appropriate for FX), max drawdown, CAGR.
    - Trade frequency, win rate, avg win/loss, payoff ratio, expectancy per trade.
    - Execution-specific: average spread cost per trade, average slippage per trade, swap contribution.
- Statistical caution: low trade counts yield noisy Sharpe estimates; use bootstrapping and multi-period validation.

## 12. Practical examples (generic)

- Example 1 — Trend-following LF (4H):
    - Data: 4H OHLCV, ATR(14).

    - Entry: long when price closes above 20-4H EMA and ATR(14) > ATR_min; use limit at close.
    - Stop: ATR(14)*2 below entry.
    - Target: trailing ATR(14)*1.5 or exit on cross below 20 EMA.
    - Execution model: apply median spread for pair, slippage = 0.2*ATR on fills, include rollovers nightly.
- Example 2 — Mean-reversion LF (1D, ~10-day holding):
    - Data: daily OHLCV.
    - Entry: short when close is > 2.5*ATR above 20-day SMA and RSI(14) > 75.
    - Stop: 1.5*ATR above entry; target: mean reversion to SMA.
    - Risk: 0.5% equity per trade; size via ATR.

(These examples are generic — adapt to your broker specifics in implementation.)

## 13. When to consider moving to HF

- Only consider HF if:
    - You need strategies that exploit sub-second inefficiencies.
    - You can invest in hardware, data, and regulatory compliance.
    - You understand market microstructure and have low latency execution needs.
- For most retail/prosumer algorithmic Forex traders targeting 4H/1D, HF is unnecessary and costly.

## 14. Quick decision checklist for design choice

- If your decision interval is 4H/1D and you target multi-day swings → Low-Frequency.
- If you require sub-second latency, order book queueing, or market-making → High-Frequency.
- For LF: prioritize robust execution models, realistic costs (spread/slippage/rollover), walk-forward tests, and disciplined risk controls.

## 15. End-of-chapter implementation checklist

- Use OHLCV bars (UTC) as authoritative data.
- Implement spread, slippage, and rollover in backtest engine.
- Use ATR(14) defaults for stops and sizing; default risk 0.5% equity.
- Execute decisions on candle close; prefer limit entries with deterministic fallbacks.
- Run paper trading for 30–90 days; enable kill switches and daily reconciliation.
- Perform walk-forward validation and bootstrap metrics when trade counts are low.

References for further reading:

- Microstructure and market making texts (HF context).
- Practical algorithmic trading books focusing on daily/4H systematic trading.
- Broker documentation on swaps/rollover and spreads.

# Automated Trading Bots

## Purpose and scope

This chapter provides pragmatic, codable design principles for building automated Forex trading bots for 4H and 1D timeframes (short-swing, ~10-day horizon). It covers system architecture, deterministic rule design, data pipelines, execution modeling (spread/slippage/rollover), state management, logging/reconciliation, testing, and safety. Examples and pseudocode are generic and Python-friendly.

## High-level architecture

Components:

- Data Layer: ingestion, cleaning, resampling, storage (parquet/feather).
- Strategy Engine: deterministic rule evaluation, signal generation.
- Risk & Sizing Module: position size, margin checks, portfolio limits.
- Execution Engine: order construction, pre-trade checks, broker API interface, simulated fills.
- State & Persistence: durable state machine for orders/trades/positions (idempotent).
- Backtester & Simulator: identical logic run on historical data with cost models.
- Monitoring & Ops: logging, alerts, health checks, dashboards.
- Deployment / Runner: scheduler (cron/Airflow), containerization (Docker), CI.

Recommended minimal tech stack:

- Python 3.10+, pandas, numpy, pyarrow (parquet), requests/websocket client, a lightweight job scheduler, and a remote logging/alerting channel.

## Design principles (core rules)

- Deterministic decisioning
  - Always evaluate entry/exit rules at a deterministic time: candle close in UTC.
  - No randomness in live decisioning; randomness only in simulation experiments when testing distributions.
- Single source of truth
  - One canonical data store (resampled OHLCV) used by backtests and live trading.
  - Version data and record data provenance (timestamps, raw source ID).
- Idempotent operations
  - Re-running the same candle decision must not create duplicate orders.
  - Use durable order IDs and persisted last-processed timestamp.
- Separation of concerns
  - Strategy code returns trade intents; a separate execution module converts intents to orders and handles fills/retries.

- Deterministic order-handling policy
    - Prefer limit entries at specified prices; if not filled by a deterministic time, fallback to market as defined.
    - Fully codify order fallbacks and partial fill behavior.
- Explicit execution realism
    - Model spread, slippage, commission, and rollover in both backtest and paper/live estimations.
    - Use ATR(14) as slippage/stop sizing baseline.
- Fail-safe defaults
    - Use conservative defaults: default risk 0.5% equity per trade, ATR(14) for stops, max daily loss threshold.
    - Implement hard kill switches that stop trading if breached.
- Reconciliation and auditability
    - Log every incoming tick/bar, decision, order, fill, and state change with timestamps.
    - Daily reconcile broker fills and balances with internal ledger.
- Testing first
    - Unit tests for rule logic, integration tests for broker adapter, end-to-end paper trading for at least 30–90 days.

## Data pipeline: ingestion to signal

Steps:

1. Fetch raw data (broker or data vendor) in UTC.
2. Normalize symbols and timestamps; drop duplicate bars.
3. Fill missing bars using conservative rules: mark gaps and do not interpolate trades across long holiday gaps. For short single missing bars, forward/backfill cautiously and flag.
4. Resample to required timeframes (4H, 1D) using:
    - O = first valid price, H = max, L = min, C = last, V = sum(volume).
5. Compute indicators (ATR(14), EMA/SMA, RSI, etc.) with stable rolling windows.
6. Save cleaned bars to versioned parquet store.

Implementation notes:

- Cache computations and store feature columns to speed backtests.
- Always store the source raw file checksum and ingestion timestamp.

## Strategy engine: deterministic rules and signals

Structure:

- Input: closed bar at time t, current positions, account equity, open orders.
- Output: zero or more trade intents (side, notional or units, entry type limit/market, price, stop, target, expiration).
- Rules must be pure functions: given same input, always same output.

Signal pseudocode (generic):

```
def evaluate_strategy(bar, indicators, position, account):
    signals = []
    if long_condition(bar, indicators) and not position.is_long():
        qty = size_for_risk(account.equity, ATR=indicators.atr)
        intent = TradeIntent(side='buy', qty=qty, entry_type='limit', price
        signals.append(intent)
    if exit_condition(bar, indicators, position):
        intent = TradeIntent(side='sell', qty=position.size, entry_type='ma
        signals.append(intent)
    return signals
```

Key constraints:

- Use ATR(14) default for stops and sizing.
- Risk per trade default: 0.5% of equity.
- Expiry for limit orders: next bar close; if not filled, convert to market at open of following bar.

### Risk & sizing module (codable)

Position sizing steps:

1. Compute dollar risk per trade = equity * risk_pct (default 0.005).
2. Stop distance = max(ATR(14)*stop_atr_mult, min_stop_pips) — default stop_atr_mult = 2.
3. Units = floor(dollar_risk / (stop_distance * pip_value)).
4. Check margin: required_margin = units * contract_size / leverage; reject if insufficient.

Pseudocode:

```
def size_for_risk(equity, ATR, stop_mult=2, risk_pct=0.005, pip_value=10):
    dollar_risk = equity * risk_pct
    stop_pips = ATR * stop_mult
    units = floor(dollar_risk / (stop_pips * pip_value))
    return max(units, 0)
```

Notes:

- pip_value depends on pair and lot sizing; make it configurable.
- If units == 0, do not place trade.

## Execution engine: order lifecycle and realism

Order lifecycle (state machine):

- CREATED -> SENT -> PARTIALLY_FILLED / FILLED / REJECTED -> CANCELLED -> CLOSED
- Persist all transitions with timestamps and broker IDs.

Order handling policy (deterministic):

- For entries:
  1. At candle close, compute intent.
  2. Place a limit order at specified price (often close or entry offset).
  3. If limit not filled by next candle open, cancel and place market order at next open (or skip if policy says never cross spread).
- For exits:
  - Prefer placing stops as OCO (one cancels other) with limit profit if defined.
  - Implement time-based stop loss (e.g., if target not hit within N bars, exit or trail).

Execution realism:

- Spread: apply half-spread to pricing depending on side (buy uses ask, sell uses bid). Use median spread per pair for backtests.
- Slippage model:
  - Deterministic: slippage = k_slip * ATR (k_slip default 0.2).
  - Stochastic: sample from Normal(0, sigma=0.1*ATR) clipped.
- Fill probability for limits: implement fill chance based on how aggressive limit is relative to next bar high/low.
- Rollover/swap: compute daily funding for cross-currency pairs; apply when position spans rollover timestamp.

Example fill simulation pseudocode:

```
def simulate_fill(limit_price, next_bar, side, spread, ATR, slip_k=0.2):
    ask = next_bar.open + spread/2
    bid = next_bar.open - spread/2
    if side == 'buy':
        if limit_price >= next_bar.low and limit_price <= next_bar.high:
            executed_price = min(limit_price, ask) + slippage(ATR, slip_k)
            return executed_price
    # if not filled, handle fallback per order policy
```

## Backtester parity: ensure live-sim equivalence

- Use same strategy, sizing, and order handling code for backtest and live (shared modules).

- Backtest must simulate spread, slippage, partial fills, commission, and rollover exactly as live engine will.
- Decision timestamps must match live decision points (candle close UTC).

Backtest checklist:

- No lookahead: only use closed bar data for signals.
- Transaction cost modeling: apply at order entry/exit.
- Slippage & fill model: documented defaults and configurable seeds for reproducibility.
- Walk-forward and out-of-sample splits.
- Monte-Carlo bootstrapping for low trade counts.

## Testing strategy and CI

Tests to implement: - Unit tests - Indicator computations, sizing calculations, stop/target math. - Edge cases: zero ATR, missing bars, tiny equity. - Integration tests - Strategy -> intent generation on historical sample. - Execution simulator: verify fills against synthetic bars. - End-to-end paper trading - Run live adapter in simulated mode for 30–90 days. - Reproducibility - Deterministic seeds for stochastic parts in testing; seed can’t be used in production decisions. - CI pipeline - Run unit tests, static analysis (flake8/mypy), and integration smoke tests on commits.

## Logging, monitoring, and reconciliation

Minimum logs (persisted):

- Incoming bar timestamp and OHLCV.
- All computed indicators.
- Strategy decisions (intent objects).
- All orders sent, status updates, fills with fill price and fill time.
- Account snapshots: equity, margin, positions at daily cadence.

Monitoring:

- Health checks: last processed timestamp, connectivity to broker and data sources.
- Alerts:
    - Fill/adverse slippage > threshold (e.g., >1.5*expected slippage).
    - Daily loss > X% (e.g., 3%).
    - Stale data > N bars.
- Dashboards: PnL over time, drawdown, open positions, trade list.

Reconciliation:

- Daily reconcile internal trades and PnL with broker statements.
- Keep both broker and internal ledgers and run diff reports; log discrepancy investigations.

### Safety features and kill switches

Mandatory safety controls:

- Max daily loss (hard): pause trading for 24 hours if breached.
- Max consecutive losing trades threshold: configurable.
- Max open positions: limit concurrent exposure.
- Leverage guard: if margin usage exceeds threshold, stop new trades.
- Stale data guard: halt on data gaps beyond limit.
- Manual emergency stop: immediate kill that rejects all new orders and attempts to flatten positions.

Implementation tip:

- Implement a “circuit breaker” service: a process that evaluates risk rules and can update a central flag (persisted) read by the execution engine.

### Idempotent state machine and persistence

State machine properties:

- Durable storage: use a small transactional DB (SQLite for single-node, Postgres for distributed) to persist orders, trade history, and last_processed_bar.
- Idempotency: store unique keys combining candle_time + strategy_id to avoid duplicate intents.
- Recovery path: on restart, load last_processed_bar and resume from next bar; reconcile open orders/positions with broker.

State schema minimal example (table-like):

- orders(order_id, strategy_id, local_id, broker_id, status, side, qty, price, created_at, updated_at)
- trades(trade_id, order_id, price, qty, timestamp)
- positions(symbol, qty, avg_price, updated_at)
- metadata(key, value)

### Deployment & operations

Deployment checklist:

- Containerize strategy and runner (Docker).
- Use environment variables for secrets and config; never hardcode API keys.
- Use scheduled runs aligned to candle close + small buffer (e.g., 5s after close) to allow data propagation.
- Use separate environments: dev/backtest -> paper -> live. Promote via CI.
- Use feature flags to toggle strategies or risk parameters without redeploy.

Operational practices:

- Run paper trading for 30–90 days matching equity and latency to live.
- Start small in live (fractional capital) and scale up after passing performance and operational checks.
- Maintain runbooks: procedures for restoring service, manual flattening, and emergency communications.

### Example: minimal bot flow

```
# main loop (runs once per candle close)
def main_loop():
    bar = data_store.get_closed_bar(symbol, timeframe, t0)
    indicators = compute_indicators(bar_history)
    account = broker.get_account_snapshot()
    position = broker.get_position(symbol)
    intents = strategy.evaluate(bar, indicators, position, account)
    for intent in intents:
        if risk.check(intent, account, position):
            order = execution.place_order(intent)
            db.save_order(order)
    reconcile_and_log()
```

Details:

- risk.check includes sizing and margin verification.
- execution.place_order handles conversion of intent to limit/market order and persists status.
- reconcile_and_log performs reconciliation and emits alerts if needed.

### Execution modeling defaults

Defaults (configurable):

- Candle timezone: UTC.
- Decision time: 5 seconds after candle close to ensure data updated.
- ATR(14) for volatility baseline.
- Stop multiplier: 2.0 * ATR.
- Risk per trade: 0.5% equity.
- Limit entry expiry: 1 bar.
- Slippage model: deterministic 0.2 * ATR for fills, stochastic option for testing.
- Spread: use historical median spread per pair; apply half-spread to entry/exit.

Codable config example (JSON-style):

```
{
  "timeframe": "4H",
```

```
  "decision_delay_sec": 5,
  "atr_period": 14,
  "stop_atr_mult": 2.0,
  "risk_pct": 0.005,
  "limit_expiry_bars": 1,
  "slippage_k": 0.2,
  "spread_model": "median_from_history"
}
```

### Practical pitfalls and how to avoid them

- Divergent backtest/live behavior:
  - Cause: inconsistent cost models or different data.
  - Fix: share execution code between backtest and live; same spread/slippage logic.
- Duplicate orders on restart:
  - Cause: non-idempotent processing.
  - Fix: persist last_processed_bar and use unique keys.
- Silent failures (stale data):
  - Cause: network or API changes.
  - Fix: health checks and alerts for stale bars.
- Underestimating rollover impact:
  - Cause: ignoring swaps on multi-day positions.
  - Fix: include swap modeling; test across different rollover regimes.
- Overfitting in development:
  - Cause: too many parameters tuned on limited data.
  - Fix: prefer simple, robust rules; use walk-forward and OOS testing.

### Example code snippets (generic Python)

Indicator and ATR (pandas):

```
import pandas as pd
def atr(df, period=14):
    high_low = df['high'] - df['low']
    high_close = (df['high'] - df['close'].shift()).abs()
    low_close = (df['low'] - df['close'].shift()).abs()
    tr = pd.concat([high_low, high_close, low_close], axis=1).max(axis=1)
    return tr.rolling(period, min_periods=period).mean()
```

Sizing:

```
def size_for_risk(equity, atr, stop_mult=2.0, risk_pct=0.005, pip_value=10)
    dollar_risk = equity * risk_pct
    stop_pips = atr * stop_mult
    if stop_pips <= 0: return 0
```

```
    units = int(dollar_risk / (stop_pips * pip_value))
    return max(units, 0)
```

Limit order handling (simplified):

```
def place_limit_or_market(broker, intent, expiry_bars=1):
    order = broker.place_limit(intent.symbol, intent.qty, intent.price)
    saved = db.save_order(order)
    # on next candle, if order still open and expiry reached:
    if db.order_age_in_bars(order) > expiry_bars:
        broker.cancel(order.broker_id)
        market = broker.place_market(intent.symbol, intent.qty)
        db.save_order(market)
```

## End-of-chapter implementation checklist

- Centralized UTC OHLCV store (parquet) with versioning.
- Deterministic candle-close decisioning (delay 5s).
- Shared codebase for strategy, sizing, and execution used by backtest and live.
- ATR(14) as default volatility measure; default risk 0.5% equity.
- Limit entry with 1-bar expiry and fallback to market; model spread/slippage/rollover in both sims and live estimation.
- Idempotent state machine persisted to durable DB.
- Unit tests, integration tests, and 30–90 day paper trading.
- Daily reconciliation and automated alerts for anomalies.
- Kill switches: max daily loss, max open positions, stale data guard.

## Short example workflow to launch a bot (stepwise)

1. Implement strategy as pure function (inputs: closed bar, indicators, positions).
2. Implement sizing and risk checks.
3. Implement execution adapter (limit + fallback logic) and simulate fills.
4. Run backtests including full cost models; perform walk-forward validation.
5. Run paper trading for 30–90 days; monitor reconciliation and slippage.
6. If acceptable, deploy live with limited capital and strict kill switches.
7. Scale incrementally and keep monitoring and reconciling daily.

# Backtesting with Historical OHLCV Data

## Purpose

This chapter presents a practical, codable, and reproducible approach to backtesting automated Forex strategies for 4H and 1D timeframes. It covers data acquisition and cleaning, resampling, timezone handling (UTC), realistic transaction-cost and execution modeling (spread, slippage, rollover), backtest engine design (event vs vectorized), validation (walk-forward/out-of-sample), common pitfalls (lookahead, survivorship), and reproducible experiment practices. Python-friendly pseudocode and short code snippets are included.

## Goals and assumptions

- Target horizon: 4H and 1D bars; decisions at candle close (UTC).
- Trades are short-swing (~10 days typical holding), but backtests should cover multiple years (≥3) for regime robustness.
- Defaults:
    - ATR(14) for volatility and stop sizing.
    - Risk per trade = 0.5% equity (configurable).
    - Limit entries with 1-bar expiry fallback to market.
- Backtest must model spread, slippage, commission (if any), and rollover (swap) precisely and identically to live/paper execution logic.

## Data requirements and acquisition

Minimum fields per bar:

- timestamp (UTC, close time)
- open, high, low, close
- volume (if available)

Optional but recommended:

- bid_ask spread history or per-bar average spread
- mid price series if source provides
- instrument contract size and pip value
- historical swap rates (long/short) per day

Data sources:

- Broker historical REST or WebSocket dumps
- Commercial vendors (TickData, HistData) or aggregated providers
- For robustness, prefer broker or exchange data where you will execute (aligns tick conventions and rollovers)

Data span:

- Minimum 3 years for LF strategies; longer for multi-regime tests.

## Data cleaning and normalization

Steps (codable):

1. Convert timestamps to UTC and round to bar close time.
2. Remove duplicate bars (keep first/validated one); log duplicates count.
3. Detect and mark gaps:
    - Short gap (1 bar): fill conservatively (forward fill close, high=max(prev_close, next_high), low=min(prev_close, next_low)) and flag.
    - Long gaps (> N bars, e.g., >5 for daily): do not interpolate; start a new contiguous segment; strategies should ignore trades crossing large gaps.
4. Normalize symbols and contract conventions (e.g., EURUSD vs EUR/USD).
5. Adjust for corporate actions not applicable to FX, but handle broker tick-size and quoting changes.
6. Validate OHLC invariants: high >= max(open,close), low <= min(open,close). Fix anomalies by marking and optionally reconstructing from tick data if available.
7. Save cleaned data as versioned parquet/feather with provenance metadata (source, ingest_time, checksum).

Code snippet (pandas-style):

```
df['timestamp'] = pd.to_datetime(df['timestamp'], utc=True)
df = df.drop_duplicates(subset='timestamp', keep='first').sort_values('time
# detect gaps
df['delta'] = df['timestamp'].diff() / pd.Timedelta('1H')  # for 4H set to
gaps = df[df['delta'] > expected_delta_hours * 1.5]
```

## Resampling and timezone alignment

- Use UTC as canonical timezone.
- For 4H: resample by 4H bins aligned to UTC midnight (00:00, 04:00, …).
- For daily: use calendar day in UTC; clarify whether daily close is 00:00 or broker's 17:00 NY rollover — default to UTC close and document mapping to broker rollover used for swap calculation.
- Resample rules:
    - open = first valid price in bin
    - high = max
    - low = min
    - close = last valid price
    - volume = sum (if missing, leave NaN)
- Keep original higher-frequency data for more realistic fill modeling if available.

Resampling snippet:

```
ohlc = {'open':'first','high':'max','low':'min','close':'last','volume':'su
bars = tick_df.resample('4H', label='right', closed='right').agg(ohlc).drop
```

### Execution modeling: spread, slippage, fills

Execution realism is essential. Backtest must simulate the same order lifecycle and cost model as live.

Core components: - Spread model - Preferred: historical per-bar median spread if available. - Fallback: fixed spread per pair (majors: ~0.5–1.5 pips typical; adapt per broker). - Apply half-spread to buy (ask) and sell (bid) sides. - Slippage model - Deterministic: slippage = k * ATR (default k=0.2). - Stochastic: sample from Normal(μ=0, σ=σ_k * ATR) then clip. - For limit orders, tie fill probability to aggressiveness (proximity to next bar high/low). - Fill rules for limit orders (deterministic algorithm) - If limit price lies within next bar's [low, high], consider filled at limit adjusted for spread and slippage. - For aggressive limits (>= next bar open for buys), high fill probability and lower slippage. - If not filled within expiry_bars, cancel and optionally place market at open (apply spread + slippage). - Partial fills - Optionally implement partial fills if using large size vs typical liquidity; otherwise assume full fills for LF. - Commission - If broker charges per-lot commissions, apply at entry/exit. - Swap/rollover model - Per-day funding based on position direction, pair, and notional. - For each calendar day a position is open and crosses broker rollover timestamp, apply swap = notional * swap_rate/365. - Use historical overnight rates if available; otherwise use static estimated rates per pair.

Codable fill example (pseudocode):

```
def simulate_entry(limit_price, next_bar, side, spread, atr):
    ask = next_bar.open + spread/2
    bid = next_bar.open - spread/2
    if side == 'buy':
        if limit_price <= next_bar.high and limit_price >= next_bar.low:
            price = limit_price + slippage(atr)
            executed_price = max(price, ask)  # must be >= ask
            return executed_price
    return None  # not filled this bar
```

### Backtest engine design: event-driven vs vectorized

- Vectorized backtester
  - Fast; uses arrays and pandas operations.
  - Good for strategy discovery and parameter sweeps.
  - Harder to model complex order lifecycles, partial fills, and state-dependent behaviors.
- Event-driven backtester
  - Simulates events (bar close, order sent, fill) and runs the same code as live.
  - Easier to ensure parity with live execution; handles complex order flows.
  - Slower but preferable for final validation and production parity.

Recommendation:

- Use vectorized for initial research and speed; use event-driven engine (shared with live execution module) for final testing and deployment validation.

Engine architecture (components):

- Data feed (sends closed bar event)
- Strategy (generates intents)
- Risk manager (sizes and validates)
- Broker simulator (executes orders with cost models)
- Portfolio (tracks positions, PnL, equity)
- Recorder (logs trades and snapshots)

## Order and trade simulation details

Key rules to code exactly as in live:

- Decision time: at closed bar t, use only data up to t.
- Limit order placement: price and expiry (1 bar default).
- Market fallback: cancel limit after expiry and place market at open of next bar.
- Stop loss placement: model as stop orders that trigger based on intra-bar high/low; when stop is within bar range, execute at stop price ± slippage and spread.
- OCO pairs: stop and target must cancel each other atomically in simulation.
- Time priority: when multiple price triggers occur in single bar (e.g., both stop and profit), define deterministic execution order (e.g., worst first for realism or price priority); document choice.

Intra-bar trigger handling (common ambiguity):

- If high > target and low < stop in same bar, order of events matters. Conservative approach: assume the worse outcome for the trader (stop hit) unless tick data is available. Alternatively simulate both scenarios probabilistically and penalize optimism.

Implementation choices (codable defaults):

- Conservative rule: if both stop and target are within same bar, assume stop occurred first (more realistic for retail).
- If tick data available, replay tick sequence to resolve exact order.

## Performance metrics and reporting

Essential metrics:

- Net profit / loss (currency)
- Return series and equity curve (time series)
- Annualized return and volatility

- Sharpe ratio (use appropriate risk-free rate for FX)
- Max drawdown and drawdown duration
- Trade statistics: total trades, wins, losses, win rate, avg win/loss, expectancy, avg holding period
- Execution metrics: avg spread cost/trade, avg slippage/trade, total swap cost
- Risk metrics: peak margin usage, % days with margin breach risk

Reporting:

- Produce per-period (monthly/quarterly/yearly) breakdowns.
- Include trade list CSV with full fields (entry_time, entry_price, exit_time, exit_price, qty, pnl, costs, swap).
- Plot equity curve and drawdowns.

## 9. Validation: walk-forward, out-of-sample, and robustness tests

- Walk-forward validation (rolling)
  - Split dataset into rolling training and testing windows (e.g., 2 years train, 1 year test rolling forward by 6 months).
  - Optimize parameters on training, apply to next test window, repeat and aggregate results.
- Forward-testing and paper trading
  - After backtest + walk-forward, run paper trading for 30–90 days with live market data to validate execution assumptions.
- Bootstrapping and Monte Carlo
  - Resample trades or blocks of returns to estimate distribution of outcomes; useful when trade counts are low.
- Sensitivity analysis
  - Vary key parameters (ATR mult, risk_pct, spread, slippage) to quantify sensitivity.
- Stress testing
  - Simulate extreme market moves, spread widening, and periods of low liquidity.
- Survivorship and sample-selection bias
  - Ensure data includes delisted pairs or symbol changes; for FX this is less prevalent but still ensure historical quoting changes are handled.

Practical walk-forward checklist:

- Choose realistic window sizes given expected trade frequency (e.g., if strategy averages 1 trade per 10 days, ensure each test window contains sufficient trades—at least 50 recommended).
- Avoid over-tuning to a single period; prefer parameter stability across windows.

## Common backtesting pitfalls and how to avoid them

- Lookahead bias
  - Cause: using future data (e.g., next bar close) for signal on bar t.
  - Fix: enforce strict use of closed bar data with timestamps and deterministic processing.
- Survivorship bias
  - Cause: removing instruments that ceased to exist after the sample period.
  - Fix: include historical quoting and consider continuity rules; for FX, ensure symbol convention changes are handled.
- Data snooping / overfitting
  - Cause: excessive parameter tuning on historical data.
  - Fix: limit free parameters, use walk-forward, cross-validation, and out-of-sample testing.
- Incorrect spread/slippage modeling
  - Cause: assuming zero spread or unrealistic slippage.
  - Fix: use historical spread series or conservative defaults; model spread widening during news.
- Ignoring rollover/swaps
  - Cause: assuming zero overnight funding.
  - Fix: include swap rates per pair and apply to multi-day holds.
- Improper intra-bar handling
  - Cause: naive use of OHLC without deterministic intra-bar rules.
  - Fix: document and implement deterministic intra-bar execution policies; use tick data for final validation.
- Not modeling partial fills or liquidity
  - Cause: assuming unlimited liquidity for large sizes.
  - Fix: set reasonable size limits or model partial fills when needed.

## Reproducibility and experiment management

- Version all data and code; store checksums for datasets.
- Use configuration files (JSON/YAML) for experiment parameters; log them with results.
- Tag experiments with git commit hash and environment (Python version, package hashes).
- Seed stochastic components used in testing; never rely on nondeterministic behavior for live decisions.
- Store results (metrics, trade CSVs, equity curves) in experiment DB or structured storage.

Minimum experiment record:

- experiment_id, git_commit, data_checksum, config_json, start_time, end_time, metrics_json, trade_list_path

### Example step-by-step backtest workflow (practical)

1. Ingest raw broker/data vendor OHLCV into raw store; record checksum.
2. Clean and normalize data; save versioned parquet.
3. Resample to 4H and daily UTC bars.
4. Compute indicators (ATR(14), EMAs, RSI) and cache features.
5. Implement event-driven simulator mirroring live execution policy (limit expiry, spread, slippage, swap).
6. Run backtest over full sample; produce metrics and trade list.
7. Run walk-forward tests (rolling windows) and sensitivity analysis (spread/slippage extremes).
8. Perform bootstrapping to evaluate metric stability.
9. Run paper trading for 30–90 days using identical execution code.
10. If acceptable, promote to live with small capital and strict kill switches.

### Minimal example: event-driven backtest pseudocode

```
for bar in bars:  # bars are closed bars in chronological order
    state.load_current_positions_and_orders()
    signals = strategy.evaluate(bar, indicators, state)
    intents = risk.size_and_validate(signals, state.account)
    for intent in intents:
        order = broker_sim.place_limit(intent)
        state.persist(order)
    # simulate fills inside next bar using next_bar_data
    next_bar = get_next_bar(bar)
    state = broker_sim.process_fills(next_bar, state)
    state.apply_rollover_if_crossing_roll_time(next_bar, rollover_rates)
    recorder.snapshot_equity(bar.timestamp, state.account)
```

Notes:

- Evaluation uses only historical data up to current bar.
- Fills are processed using next_bar (or intra-bar logic if tick data available).

### Example Python snippets

ATR (repeatable):

```
def atr(df, period=14):
    high_low = df['high'] - df['low']
    high_close = (df['high'] - df['close'].shift()).abs()
    low_close = (df['low'] - df['close'].shift()).abs()
    tr = pd.concat([high_low, high_close, low_close], axis=1).max(axis=1)
    return tr.rolling(period, min_periods=period).mean()
```

Limit fill check (simple):

```
def is_limit_filled(limit_price, next_bar):
    return (limit_price >= next_bar['low']) and (limit_price <= next_bar['h
```

Compute spread cost:

```
def apply_spread(price, side, spread_pips):
    half_spread = spread_pips / 2 * pip_to_price
    return price + half_spread if side=='buy' else price - half_spread
```

Apply rollover daily:

```
def apply_rollover(equity, position, swap_rate_daily):
    # swap_rate_daily is per notional unit
    swap_cost = position.notional * swap_rate_daily * (1 if position.side==
    equity += swap_cost
    return equity, swap_cost
```

### Backtest configuration example (JSON-style)

```
{
  "timeframe": "4H",
  "data_path": "data/eurusd_4h.parquet",
  "start_date": "2018-01-01",
  "end_date": "2025-11-08",
  "atr_period": 14,
  "risk_pct": 0.005,
  "stop_atr_mult": 2.0,
  "limit_expiry_bars": 1,
  "spread_model": {"type":"historical_median","path":"data/spread.csv"},
  "slippage": {"type":"deterministic","k":0.2},
  "swap_table": "data/swap_rates.csv"
}
```

### Backtest output artifacts (save these)

- trades.csv (one row per executed trade with full metadata)
- equity_curve.csv (timestamp, equity, drawdown)
- metrics.json (key performance indicators)
- params.json (all parameters used)
- data_version.txt (data checksum and provenance)
- logs/ (detailed run logs)

### Practical validation examples and sanity checks

- Zero-cost sanity: run backtest with zero spread/slippage and run again with conservative spread; compare effect to ensure spread impact is visible.

- Single-trade inspection: for each trade, inspect bars around entry/exit to verify fills match logic.
- Intra-bar ambiguity: pick random bars where stop/target overlap and verify chosen conservative rule yields plausible result.
- Equity conservation: check that sum(trade PnL + swaps + commissions) equals final equity change.

## Performance and scaling tips

- Use vectorized computations for indicators and initial parameter sweeps.
- Use event-driven simulation for final validation; parallelize walk-forward windows across cores/machines.
- Store intermediate computed features (e.g., ATR) to avoid recomputation across experiments.
- Use parquet format for efficient IO and partition by year for quick subsetting.

## End-of-chapter checklist for implementation

- Source raw OHLCV and spread/swap data; store checksums.
- Clean and normalize data; use UTC time alignment; resample to 4H/1D.
- Compute ATR(14) and other features; cache them.
- Implement event-driven backtester mirroring live order logic (limit expiry, market fallback).
- Implement spread, slippage (deterministic/stochastic), commission, and swap models.
- Validate intra-bar handling and adopt conservative resolution for ambiguous bars.
- Perform walk-forward, bootstrapping, and sensitivity analyses.
- Run 30–90 day paper trading using identical execution code.
- Archive experiment artifacts and maintain reproducible records.

# Implementable Strategies Using OHLCV

Chapter summary: provides codable, backtest-ready strategy blueprints for 4H and 1D Forex trading (short-swing, ~10-day horizon). Each strategy includes: data & timeframe, entry rule, stop rule, target/exit rule, sizing, execution modelling notes (spread/slippage/rollover), backtest parameters, common pitfalls, and minimal pseudocode/Python snippets. Defaults: UTC candles, ATR(14) for volatility, risk per trade = 0.5% equity, limit entries with 1-bar expiry and market fallback.

Contents:



A. Trend-following — EMA + ATR trailing (4H / 1D):

- Data: 4H or daily OHLCV, EMA_short = 20 (4H→20×4H ≈ ~40 days? use timeframe-appropriate), EMA_long = 50 (configurable), ATR(14)
- Idea: enter on momentum confirmation; use ATR trailing stop to capture swings.
- Entry:
  - Long: close > EMA_short and EMA_short > EMA_long and close crosses above EMA_short on bar close; ATR(14) > ATR_min
  - Short: symmetric
  - Place limit entry at close price.
- Stop:
  - Initial stop = entry_price − 2.0 * ATR(14) (long); symmetric for short.
- Exit / target:
  - Primary: trailing stop set to entry_price − (1.5 * ATR) and trailed by updating after each bar to max(trail, price − 1.5*ATR).
  - Secondary: exit when price closes below EMA_short (for long).
- Sizing:
  - dollar_risk = equity * 0.005
  - stop_distance = 2 * ATR
  - units = floor(dollar_risk / (stop_distance * pip_value))
- Execution model notes:
  - Apply half-spread at entry/exit; slippage = 0.2*ATR on fills.
  - Limit expiry = 1 bar then market fallback.
  - Include swap if held over rollover days.
- Backtest params:
  - Lookback 5+ years, walk-forward rolling windows, record avg holding period, win rate.

- Pitfalls:
    - Whipsaw in choppy regimes; add ATR_min filter and minimum trend slope (EMA_short − EMA_long > k*ATR).
- Pseudocode:

```
if close > EMA20 and EMA20 > EMA50 and close_prev <= EMA20_prev:
    entry = close
    stop = entry - 2*ATR
    place_limit_buy(entry)
# on each bar update trailing_stop = max(trailing_stop, close - 1.5*ATR)
# exit if close < EMA20 or price <= trailing_stop
```

B. Mean-reversion — RSI Extreme + ATR stop (1D):

- Data: Daily OHLCV, RSI(14), 20-day SMA, ATR(14)
- Idea: fade short-term extremes that statistically revert to mean within ~10 days.
- Entry:
    - Long: close < SMA20 − 2.5*ATR and RSI(14) < 30.
    - Short: close > SMA20 + 2.5*ATR and RSI(14) > 70.
    - Limit at close.
- Stop:
    - Initial stop = entry − 1.5*ATR (long).
- Target:
    - Primary target at SMA20 (mean) or partial scale-out at 0.8*SMA20 + 0.2*entry.
    - If not hit within 10 trading days, exit at market (time stop).
- Sizing:
    - Risk 0.5% equity; stop_distance = 1.5*ATR.
- Execution:
    - Spreads matter for one-day moves; apply median spread + slippage 0.2*ATR.
- Backtest:
    - Ensure sufficient samples; bootstrap trade outcomes due to low freq.
- Pitfalls:
    - Trend regimes where mean shifts; add trend filter: only take reversion trades if SMA slope |ΔSMA20| < small threshold.
- Pseudocode:

```
if close < SMA20 - 2.5*ATR and RSI < 30:
    entry = close
    stop = entry - 1.5*ATR
    target = SMA20
    expiry = current_bar + 10 days
```

C. Breakout with confirmation — Volatility Breakout (4H):

- Data: 4H OHLCV, ATR(14), 20-period Range (high−low)

- Idea: capture directional moves when volatility expands; require confirmation on close to reduce false breaks.
- Entry:
  - Long: close > recent_high(20) and close > open_of_bar (momentum check); optionally require volume spike or ATR increase (ATR > ATR_rolling_mean*1.2).
  - Place entry as aggressive limit at close.
- Stop:
  - Stop = entry − 1.5*ATR or below breakout support (recent_high − 0.5*ATR).
- Target:
  - 2:1 reward:risk fixed, or trail by ATR(14)*1.5.
- Execution:
  - Pre-check spreads widen during breakouts; model spread upscaling (multiply median spread by factor during ATR spikes).
- Backtest:
  - Measure false breakout rate; require minimum n trades for statistical confidence.
- Pitfalls:
  - Breakouts often fail in low-liquidity hours; add session filter (avoid low-volume 00:00–04:00 UTC).
- Pseudocode:

```
if close > rolling_high(20) and ATR > rolling_ATR_mean*1.2:
    entry = close
    stop = entry - 1.5*ATR
    place_limit_buy(entry)
```

D. Channel / Range Trading — Support/Resistance Flip (4H / 1D):

- Data: 4H or daily OHLCV; swing pivot detection (n=5)
- Idea: buy at support bounce; sell at resistance rejection.
- Entry:
  - Identify swing low pivot L where price low at t is < lows of n bars before & after.
  - Long: price closes above pivot low + buffer (e.g., 0.2*ATR) after bounce; confirm by close > previous close.
- Stop:
  - Stop below pivot low − 0.6*ATR (allow tail).
- Target:
  - Target at next resistance pivot or fixed R:R 1.5–2.0.
- Sizing:
  - Risk 0.5% equity; stop distance based on ATR.
- Execution:
  - Use limit entries slightly inside support to improve fill odds; model fill probability.

- Pitfalls:
    - False bounces when channel breaks; include breakout guard (cancel long if close < pivot low within 2 bars).
- Pseudocode:

```
if is_swing_low(t) and close > low_pivot + 0.2*ATR:
    entry = close
    stop = low_pivot - 0.6*ATR
```

E. Event-aware swing — Macro release safe handling (1D / 4H):

- Data: OHLCV, economic calendar (GMT/UTC), volatility measures
- Idea: avoid or hedge positions across major macro events (FOMC, NFP). If taking trades, widen stops or reduce size.
- Rules:
    - Pre-event blackout: do not open new positions during blackout window (e.g., 2 hours before to 1 hour after major release).
    - If holding position through event: reduce size or set wider stop = 3*ATR and lower risk per trade (0.25%).
    - Post-event filter: ignore signals for first N bars (e.g., 1–2 bars) to allow volatility settle.
- Execution:
    - Model spread widening and slippage as event multiplier (spread_factor 2–5x) in backtest for bars overlapping events.
- Pseudocode:

```
if upcoming_event_within(hours=2):
    skip_new_entries()
if holding and event_crosses_hold_period:
    set_stop(entry - 3*ATR)
    reduce_size(risk_pct=0.0025)
```

Strategy portfolio considerations:

- Correlation limits: limit total exposure to correlated pairs (e.g., EURUSD & GBPUSD) via net notional caps.
- Max concurrent trades: default 3–5 depending on capital and margin.
- Position scaling: prefer fixed sizing per strategy rather than dynamic scaling across many strategies unless proven by backtest.
- Portfolio margin: simulate aggregated margin to prevent cross-pair margin strain.

Common modeling & execution notes (applies to all):

- Decision timing: all signals evaluated on closed bar t (UTC). Place limit at close price; expire after 1 bar, fallback to market at next open.
- Spread: apply per-pair historical median; for event bars increase spread by factor (configurable).

- Slippage: deterministic default = 0.2*ATR; test sensitivity to 0.4–1.0*ATR for stressed environments.
- Swap: calculate and apply for positions held across rollover (broker specific); include in PnL.
- Partial fills: assume full fills for retail LF strategies unless notional exceeds reasonable liquidity; implement partial fills if sizes large.
- Risk defaults: ATR(14), stop_mult default 2.0 (trend) or 1.5 (reversion), risk_pct default 0.5%.

Backtest parameters recommended:

- Sample length: ≥ 3 years; prefer 5–10 years for stable regimes.
- Walk-forward: rolling windows with at least 1–2 years training and 6–12 months test; ensure sufficient trades per window (≥50 recommended).
- Monte Carlo: resample trade outcomes or block bootstrap to estimate distribution of equity curves.
- Sensitivity: vary spread and slippage ±50–200% to estimate breakpoints.

Example combined strategy pseudocode (generic):

```
for bar in bars:
    compute indicators (EMA, SMA, ATR, RSI, pivots)
    if not in_blackout(bar.timestamp):
        # trend
        if trend_long_condition(bar):
            intent = create_long_intent(entry=bar.close, stop=bar.close-2*A
        # reversion
        elif reversion_long_condition(bar):
            intent = create_long_intent(entry=bar.close, stop=bar.close-1.5
    send_intents_through_risk_and_execution(intent)
    update_trailing_stops_and_check_exits(bar)
    apply_rollover_if_crossing(bar)
```

Trade record fields to log for each trade (ensure reproducibility):

- strategy_id, symbol, side, entry_time, entry_price, entry_type(limit/market), stop_price, target_price, exit_time, exit_price, qty, pnl_gross, spread_cost, slippage_cost, swap_cost, commission, net_pnl, holding_period_bars, notes (e.g., event overlap)

Practical testing checklist per strategy:

- Run vectorized parameter sweep for discovery, then event-driven backtest for final validation.
- Validate single trade behavior visually (plot entry/exit on price chart) for random sample of trades.

- Test sensitivity to:
    - spread: median vs worst-case
    - slippage: 0.2*ATR vs 0.5–1.0*ATR
    - stop multiplier: 1.5–3.0
    - limit expiry: 1 vs 2 bars
- Paper trade each strategy for 30–90 days under live spreads and execution to measure realized slippage and swap impacts.

Scaling and money management recommendations:

- Start live at small fraction (e.g., 1–10% of intended capital) to validate execution assumptions.
- Use fixed risk per trade per strategy; cap portfolio risk (e.g., total risk across open trades ≤ 3% equity).
- Rebalance exposure if a strategy shows drift from expected slippage or swap costs.

Quick summary table (concise):

- Trend (EMA+ATR): medium frequency, holds days–weeks, stop 2*ATR, trail 1.5*ATR, risk 0.5%
- Reversion (RSI+ATR): lower frequency, ~10-day horizon, stop 1.5*ATR, time stop 10 days, risk 0.5%
- Breakout (volatility): opportunistic, requires ATR spike, stop 1.5*ATR, trail or 2:1 target
- Channel: frequent small trades, stops tight around pivot with ATR cushion, risk management critical
- Event-aware: safety wrappers—avoid/newly adjust positions around events; model spread widening

End-of-chapter checklist:

- Implement shared execution model (limit + 1-bar expiry + market fallback).
- Use ATR(14) for stops and sizing; default risk 0.5% equity.
- Model spread, slippage, and rollover in backtests identical to live code.
- Run walk-forward validation and bootstrap analyses; require sufficient trade counts per window.
- Paper test strategies 30–90 days before scaling; start live small.
- Enforce portfolio caps and correlation limits.

# Strategy Evaluation Workflow

## Purpose

Provide a compact, actionable framework for evaluating automated Forex strategies (4H / 1D, ~10-day horizon). Focus on metrics that matter for deployment, model selection (walk-forward), and realistic performance assessment with execution effects.

## Evaluation workflow (prescriptive)

1. Prepare data and simulate realistic execution (see Chapter 18/Backtesting):
    - Use UTC candles, true bid/ask mid handling, apply spreads, per-trade slippage model, rollover/swap costs, and commission schedule.
    - Enforce deterministic entry/exit at candle close rules where applicable; model limit fills probabilistically with conservative fill rates.
2. Split data:
    - Walk-forward blocks (recommended): rolling train/test windows (e.g., 24 months train → 3 months test) repeated through dataset.
    - Single holdout only if dataset is small; prefer chronological splits that preserve market regime order.
3. Run backtests with Monte Carlo resampling of execution parameters:
    - Vary spread ±20–50% and slippage distribution; jitter entry times within candle if modelling intrabar uncertainty.
    - Record metric distributions, not only point estimates.
4. Reject strategies with fragile performance across perturbations; prefer robust median metrics and low tail risk.

## Core metrics (always compute)

- Net profit (absolute and percentage).
- CAGR / annualized return (use geometric compounding).
- Annualized volatility of returns.
- Sharpe ratio (use risk-free ~ short-term cross-currency or 0 for comparability; report period).
- Sortino ratio (downside risk focus).
- Maximum drawdown (peak-to-trough) and drawdown duration.
- Calmar ratio (annualized return / max drawdown) for drawdown sensitivity.
- Win rate (percentage of profitable trades).
- Profit factor (gross profit / gross loss).
- Average win / average loss and pay-off ratio.
- Expectancy per trade = (win_rate * avg_win) - (loss_rate * avg_loss).
- Trades per year and average holding time.
- Exposure and utilization (average % of capital deployed).
- Turnover and total transaction costs (commissions + spread + slippage + swap).

- Tail risk metrics: 95th/99th percentile loss per trade, conditional drawdown at risk (CDaR).
- Kelly fraction (for sizing sanity check) and recommended scaling (use fractional Kelly with risk cap).
- Statistical significance: t-test on returns vs zero (beware non-normality); bootstrap confidence intervals for mean return.

### Practical evaluation table (report each backtest)

- Period (start/end)
- Net return (%)
- Annualized return (%)
- Annualized volatility (%)
- Sharpe
- Max drawdown (%)
- Trades
- Avg hold (days)
- Win rate (%)
- Profit factor
- Total costs (%)
- Remarks (e.g., stressed fills, low sample)

### Risk-adjusted and deployment-focused checks

- Return per unit of cost: Net return normalized by total transaction cost percentage.
- Capacity estimate: slippage sensitivity versus notional — estimate maximum deployable AUM before returns degrade materially (simulate increasing lot size).
- Liquidity filter: require average daily volume or currency-pair tick activity threshold for realistic scaling.
- Margin/leveraged stress: simulate margin calls and forced liquidation scenarios (use worst-case overnight moves).
- Overnight/rollover exposure check for pairs with large swaps.

### Robustness testing (must include)

- Parameter sensitivity: one-at-a-time sweeps and random hyperparameter search; present heatmaps of key metrics over parameter grid.
- Monte Carlo trade sequence resampling: bootstrap trade returns to estimate distribution of equity curves.
- Regime subsamples: evaluate performance across volatility regimes (low/med/high ATR), trending vs ranging periods, and interest-rate regime shifts.
- Slippage and spread stress tests: worst-case and percentile scenarios.
- Walk-forward analysis: track metric stability across out-of-sample periods; prefer strategies with stable out-of-sample Sharpe and bounded drawdowns.
- Survivorship and selection bias checks: ensure dataset includes delisted brokers/pairs where applicable; use whole-history tick/candle sources.

### Overfitting detection

- Too many free parameters relative to trade count: use rule-of-thumb min 30–50 trades per free parameter for credibility.
- Compare in-sample vs out-of-sample performance: large IS/OS divergence indicates overfitting.
- Use combinatorial purged cross-validation or OOS walk-forward to mitigate lookahead.
- Report p-values from bootstrap of strategy against simple null strategies (e.g., buy-and-hold on major pair, or random entry times with same hold distribution).

### Trade-level logging diagnostics (required)

- Per-trade record: entry_time, entry_price (bid/ask), entry_size, exit_time, exit_price, realized_pnl, fees, swap, max adverse excursion (MAE), max favorable excursion (MFE), slippage, reason_code (entry/exit rule name), fill_quality flag.
- Aggregate MAE/MFE histograms to inspect stop placement and whether stops are hit vs panic exits.
- Latency-impact table: simulate order placement latency (ms) vs fill rate and record sensitivity.

### Decision thresholds for deployment

- Minimum historical live-equivalent performance (example defaults; configurable):
    - Out-of-sample Sharpe >= 0.8 (annualized)
    - Max drawdown ≤ 15% (relative to strategy risk tolerance)
    - Profit factor ≥ 1.5
    - Minimum trades per year ≥ 50 (for statistical confidence on 4H/1D scale)
    - Robustness: > 70% of Monte Carlo scenarios positive median return
- If thresholds unmet, either simplify strategy, increase data, or reduce leverage.

### Reporting & visualization (must include)

- Equity curve with drawdown bands.
- Rolling Sharpe and rolling returns (90/180-day windows).
- Trade scatter: entry vs exit P/L, color-coded by rule.
- Parameter heatmaps, MAE/MFE distributions, and slippage sensitivity plots.
- Waterfall of costs (commissions, spread, slippage, swap) contributing to net return.

### Example pseudocode: evaluation pipeline (concise)

```
1. load_ohlcv()
2. apply_transaction_costs(spread_model, slippage_model, swap_schedule)
3. generate_signals(params)
4. simulate_orders(limit_policy, fill_prob)
5. record_trades_with_MAE_MFE()
```

6. `compute_metrics()`
7. `run_sensitivity_analyses(param_grid, monte_carlo_runs)`
8. `run_walkforward(train_window, test_window, step)`
9. `summarize_results_and_plots()`

### Final checklist before live (paper -> live)

- Paper trading P&L roughly matches backtest within cost/stress bands.
- Monitoring hooks: per-trade logs, alerts for fill anomalies, daily reconciliation, kill switch that can force flat within one candle.
- Capacity/stress plan documented (how to scale down/up and emergency unwind).
- Pre-registered performance acceptance criteria and rollback plan.

# Strategy Evaluation Metrics

Chapter summary: defines a concise, reproducible framework to evaluate pattern-based and rule-based FX strategies for 4H/1D automated systems. Includes required metrics, risk-adjusted measures, trade-level diagnostics, statistical tests, overfitting controls, and recommended reporting tables/plots for backtest and walk-forward analyses.

## Assumptions and defaults

- Timeframes: 4H and 1D (UTC candles).
- Decisioning: deterministic at candle close.
- Risk per trade default: 0.5% equity.
- ATR(14) used for stops, targets, and volatility filters.
- Execution realism modeled: spread, slippage, commission, rollover (see Chapter 20).
- Walk-forward: rolling walk-forward with non-overlapping OOS segments recommended.

## Required core metrics

Report for every backtest and OOS segment.

- Net Profit (absolute currency)
- Return on Equity (ROE) or Percent Return
- Annualized Return (CAGR)
- Annualized Volatility (std of daily returns scaled)
- Max Drawdown (peak-to-trough) and Drawdown Duration
- Sharpe Ratio (use risk-free ~0 for FX quoting; state annualization method)
- Sortino Ratio (downside risk)
- Calmar Ratio (CAGR / Max Drawdown)
- Win rate (percent profitable trades)
- Profit factor = gross_profit / gross_loss
- Expectancy per trade = average_profit_per_trade
- Average win / Average loss and Win/Loss size ratio
- Average holding time (bars and days)
- Trades per year and per instrument
- Exposure metrics: percent time in market, average position size
- Tail-risk measures: 95% and 99% VaR (historical) and Conditional VaR (CVaR)
- Trade sequence autocorrelation (to detect serial dependency)

## Trade-level diagnostics (essential for debugging)

- Entry price vs expected entry (slippage/spread differences)
- Fill latency and TTL expirations for limit entries
- Time-in-market per trade and intrabar fill patterns
- Rollover cost contribution to PnL per trade

- Reason for exit (TP, SL, time stop, cancellation, manual)
- Bookkeeping: record order ids, timestamps, account balance snapshots at entry/exit
- Per-trade PnL in pips and currency; normalized by risk (e.g., PnL / risk_amount)

### Performance decomposition

Recommended tables:

- By pattern type (H&S, triangle, flag, etc.)
- By timeframe (4H vs 1D)
- By instrument/pair
- By market regime (low/high ATR quantiles, trending vs range via ADX)
- By trade age (holding time buckets)
- By entry method (limit fill vs market/on-open)
- Include counts, gross profit, gross loss, expectancy, avg hold time per bucket.

### Statistical tests and significance

- Null hypothesis: returns are noise with mean zero (or equal to benchmark).
- t-test on trade returns: use caution—trade returns are not IID; prefer block bootstrap of trade sequences or time-series bootstrap (moving block bootstrap) to respect dependency.
- Bootstrap procedure:
    1. Aggregate trade returns series.
    2. Perform N=10,000 block bootstrap resamples (block length ~ 5–20 trades or days, tune for autocorrelation).
    3. Compute distribution of metric (mean return, Sharpe).
    4. p-value = fraction of bootstrap samples <= 0 (or <= benchmark).
- Monte Carlo trade-sequence shuffling: simulate many random permutations preserving holding times to estimate probability of observed max drawdown and streaks.
- Kelly fraction estimate: compute but present conservative fraction (e.g., Kelly/4) for position sizing exploration—not a default sizing method for leveraged FX.

### Robustness checks

- Parameter sensitivity grid:
    - Vary key parameters by coarse grid (±25% or sensible steps) and report stability of CAGR, Sharpe, drawdown.
    - Use heatmaps to show parameter regions with similar performance.
- Walk-forward validation:
    - Use rolling or anchored walk-forward. Example: 3-year in-sample, 1-year OOS for daily; for 4H use 18-month IS, 6-month OOS or sliding 24-week/8-week windows. Report aggregated OOS metrics.

- Purged K-Fold cross-validation:
    - When optimizing on time-series, apply purging and embargo around test folds to remove leakage.
- Out-of-sample replay:
    - Run the final (selected) rules on a truly held-out period (not used in any tuning) and report metrics.
- Survivorship and data integrity checks:
    - Verify no lookahead via indicator computation; confirm time alignment; check for missing bars; check consistent contract terms.

## Risk-adjusted and trade-quality metrics

- Expectancy (E) = (WinRate)*(AvgWin) - (LoseRate)*(AvgLoss)
- R-multiple distribution: compute R = trade PnL / initial risk. Analyze distribution: mean R, median R, percent R>1.
- Sharpe ratio (annualized): mean(daily_returns)/std(daily_returns) * sqrt(252). For 4H, convert returns to daily equivalents before annualizing.
- Sortino: use downside deviation (returns < 0) in denominator.
- Profit factor: >1 is profitable; >1.5 is decent; report significance with confidence intervals via bootstrap.
- Ulcer index: measure drawdown pain over time.
- Omega ratio: tail-weighted performance.

## Drawdown analysis and recovery

- Report full drawdown profile table: start_date, valley_date, recovery_date, drawdown_pct, peak_equity_at_start, valley_equity, days_to_recover.
- Stress-test worst-case scenarios (e.g., multiple consecutive losses exceeding expected sequence) via Monte Carlo of observed trade R distribution.
- Margin call / leverage stress: simulate increased volatility ×2 and resulting margin usage; compute probability of margin call given portfolio sizing rules.

## Transaction-cost sensitivity

- Perform sensitivity runs over a matrix of spread, slippage, commission, and rollover values (e.g., ±25%, ±50%, +100%). Present results in a table or heatmap to show fragility.
- For FX, swap rates can materially affect multi-day trades—include best/worst swap scenarios (long/short).

## Overfitting detection and prevention

- Rule-of-thumb: number of free parameters should be far less than sqrt(number_of_trades). Prefer simple rules.
- Use nested cross-validation and hold-out OOS.
- Use multiple instruments and timeframes to test generalization.

- Penalize strategies that require frequent parameter retuning or unstable parameter regions.
- Track “p-hacking” risk: correct for multiple comparisons (Bonferroni/Holm) when evaluating many hypotheses.
- If performing ML, enforce strict feature derivation rules (no future leakage), and use temporal CV with purging.

## Reporting and visualization

- Equity curve (net and gross) with drawdown shading.
- Rolling Sharpe/rolling CAGR and rolling win-rate (e.g., 6-month windows).
- Trade scatter: entry price vs exit price colored by R; R-multiple histogram.
- Return attribution by pattern/instrument/holding time (stacked bar).
- Heatmap of parameter sensitivity (rows/cols as parameters).
- Duration vs R scatter plot.
- Trade table: top 20 wins, top 20 losses with annotated reasons.
- Confusion matrix for predicted direction vs realized (for breakout bias systems).
- Trade sequence shaded by reason for exit and entry type.

## Benchmarking

- Compare to:
  - Cash benchmark (USD deposit) or cash+carry adjusted return if FX strategy is directional across currencies.
  - Passive hold (where applicable) — for FX pairs this is often meaningless; better: benchmark against mean-reversion simple strategy or baseline MA crossover.
  - Risk-parity or equal-volatility portfolio combinations if trading multiple pairs.
- Use risk-adjusted comparisons (Sharpe, Sortino, Calmar, information ratio vs benchmark).

## Practical evaluation checklist

Run for every backtest:

- ☐ Verify data integrity: no missing bars, correct timezone, correct spreads.
- ☐ Confirm deterministic decisioning: no future data used at entry calculation.
- ☐ Confirm execution realism: spread, slippage, commission, rollover modeled.
- ☐ Run baseline backtest and record core metrics.
- ☐ Run walk-forward and OOS; compare IS vs OOS performance degradation.
- ☐ Perform parameter sensitivity and report stable regions.
- ☐

Bootstrap p-values for core metrics and report confidence intervals.

- ☐
  Stress-test transaction costs and volatility multipliers.
- ☐
  Produce required visualizations and trade-level diagnostics.
- ☐
  If acceptable, run final strategy on held-out period and/or paper trading before live deployment.

## Minimal reproducible output format

For reports / appendices:

- CSV exports:
  - trade_log.csv with columns: trade_id, instrument, pattern_type, entry_time, exit_time, entry_price, exit_price, side, size, initial_risk, pnl_usd, pnl_pips, reason_exit, fees, swap, fill_type, slippage.
  - daily_equity.csv: date, equity, cash, unrealized_pnl, closed_pnl, drawdown.
  - parameter_grid_results.csv: param1, param2, trades, CAGR, Sharpe, MaxDD, ProfitFactor.
- JSON summary with consolidated metrics and bootstrap intervals.

## Quick formulas and conversions

- Annualize return (if returns are expressed per period):
  - CAGR = (EndingEquity / StartingEquity)^(periods_per_year / N_periods) - 1
- Annualize standard deviation:
  - sigma_annual = sigma_period * sqrt(periods_per_year)
  - periods_per_year: 252 for daily, ~42.5 for 4H (252*6 = 1512 4H bars per year; if using 24/5 with 6 bars/day then 252*6)
- Expectancy per trade:
  - E = (WinRate)*(AvgWin) - (1 - WinRate)*(AvgLoss)
- Position sizing (notional from risk):
  - notional = risk_amount / (stop_distance_in_price * pip_value_per_unit)
  - for FX: lots = notional / contract_size (e.g., 100,000 units per standard lot)

## Common pitfalls and how to check them quickly

- Inflated returns due to unrealistic fills: compare limit-fill-only vs market-fill assumed runs.
- Hidden reuse of OOS data in parameter tuning: verify tuning logs and avoid manual cherry-picking.
- Ignoring rollover: toggle swap off/on and note effect on multi-day trades.
- Ignoring margin effects: simulate leveraged positions with live margin rules to detect unrealistic position sizing.

### Example minimal evaluation summary (one-line per strategy)

- StrategyName, Timeframe, YearsTested, Trades, CAGR, Sharpe, MaxDD, WinRate, AvgHoldDays, ProfitFactor, Bootstrap_pvalue (Sharpe)
- e.g., H_S_4H,4H,5.2,312,12.4%,1.25,18.2%,48.1%,6.2,1.45,0.03

### Decision criteria for deployment (example rules)

- OOS Sharpe >= 0.7 and OOS CAGR >= 8% AND MaxDD <= 20% AND profit_factor >= 1.3
- Parameter stability: performance drops by <30% across ±25% parameter shifts
- Transaction-cost sensitivity: strategy remains profitable if spreads/slippage increased by 50%
- Minimum trade count: >= 100 trades in combined OOS windows (or adjust threshold with caution)

### Appendix

Simple Python snippet to compute basic metrics:

```
import numpy as np

returns = np.array(daily_pnl) / starting_equity
cagr = (1 + returns.sum()) ** (252 / len(returns)) - 1
ann_vol = returns.std() * np.sqrt(252)
sharpe = np.mean(returns) / returns.std() * np.sqrt(252)
max_dd = compute_max_drawdown(equity_series)
win_rate = sum(trade_pnl > 0) / len(trade_pnl)
profit_factor = trade_pnl[trade_pnl>0].sum() / -trade_pnl[trade_pnl<0].sum(
```

### Closing checklist

Include with every evaluation:

- Data integrity verified (timezone, missing bars)
- Execution model configured (spread/slippage/commission/swap)
- Walk-forward executed and aggregated OOS reported
- Bootstrap/Monte Carlo significance tests run
- Parameter sensitivity heatmaps produced
- Transaction-cost stress test performed
- Trade log & daily equity exported for audits
- Paper trading or simulation on live data before live deployment

# Implementation Notes

Chapter summary: provides concrete, prescriptive, codable guidance and concise Python-ready snippets for implementing 4H/1D automated FX trading systems: data handling, indicator computation, pattern detection hooks, backtest engine primitives, execution modeling (spread/slippage/rollover), position sizing, logging, and recommended project structure. All examples assume UTC candles, ATR(14) defaults, risk_per_trade = 0.5%, deterministic decisioning at candle close, and limit-entry-with-ttl behavior.

## Suggested project layout

- README.md
- requirements.txt
- data/
    - raw/
    - cleaned/
    - resampled/
- src/
    - data_io.py
    - indicators.py
    - patterns.py
    - execution_model.py
    - backtester.py
    - portfolio.py
    - sizing.py
    - reporting.py
    - broker_adapters/
        - oanda_adapter.py (example)
- tests/
    - test_indicators.py
    - test_patterns.py
    - test_backtester.py
- notebooks/
    - exploratory.ipynb
- logs/
- results/

## Key implementation principles

- Deterministic: decisions computed using only data up to current candle close.
- Idempotent order lifecycle: design orders/trades as state machines that can be safely replayed.
- Tight logging and reconciliation: log both strategy decisions and broker fill events with unique ids and timestamps.

- Modularity: separate data, signal generation, execution modeling, and portfolio sizing.
- Simulate execution realistically: model spread, slippage, partial fills, and rollovers.
- Unit tests: for indicators, pivot/pattern detectors, order/state machine behaviors.
- Configurable defaults: expose ATR_len, risk_per_trade, breakout_margin, max_wait_candles via config.

### Data ingestion and cleaning

- Use standardized CSV/Parquet input with columns: timestamp(UTC), open, high, low, close, volume, tick_count(optional).
- Verify monotonic timestamps, no duplicate indices, and consistent timezone (UTC).
- Fill missing bars explicitly (for FX 24/5, missing weekend bars are expected; ensure consistent handling).
- Resampling: if you ingest higher-frequency ticks or minutes, aggregate using:
    - open = first
    - high = max
    - low = min
    - close = last
    - volume = sum
- Save cleaned/resampled files to avoid repeated preprocessing.

### Indicators

- Use pandas and numba where heavy; prefer vectorized rolling functions.
- ATR(14) implementation (pandas):

```
def atr(df, length=14):
    high_low = df['high'] - df['low']
    high_close = (df['high'] - df['close'].shift()).abs()
    low_close = (df['low'] - df['close'].shift()).abs()
    tr = pd.concat([high_low, high_close, low_close], axis=1).max(axis=1)
    atr = tr.rolling(length, min_periods=1).mean()  # or Wilder EMA
    return atr
```

- Pivot detection (5-bar pivot example):

```
def is_pivot_high(series, i, k=2):
    return series[i] == series[i-k:i+k+1].max() and (series[i] > series[i-1
```

- Compute EMA/MA slopes using linear regression over last N bars for robust slope.

### Pattern detection

- Expose pattern detectors that return:
    - pattern_found (bool)

    - pattern_metadata: dict with key points (pivot indices/levels), neckline/lines, height, symmetry, suggested entry/stop/tps
- Keep detectors pure functions: input = dataframe slice + indicator values; output = metadata only.

Example pattern function signature:

```
def detect_head_shoulders(df_slice, atr_series, params):
    # returns (found:bool, meta:dict)
```

### Order and execution model

- Represent orders and positions as records with unique ids, status, timestamps, side, size, price, stop, tp, ttl, filled_qty, avg_fill_price.
- Order lifecycle states: NEW -> WORKING -> PARTIALLY_FILLED -> FILLED -> CANCELED -> REJECTED.
- For backtests, run an engine that iterates bars and:
    1. On bar close, run strategy to emit new orders.
    2. Feed orders into execution_model that simulates fills given bar OHLC and configured spread/slippage/liquidity rules.
    3. Update positions and ledger, log trades.
- Deterministic limit-order policy: if limit price is within bar high/low adjusted for spread, simulate fill at limit price (or better) with slippage applied if model requires.

### Execution realism primitives

- Spread model:
    - Support fixed per-instrument spread (pips) or time-varying spread series.
    - For each trade, compute effective buy_price = mid + spread/2, sell_price = mid - spread/2 for market fills. For limit fills, adjust threshold accordingly.
- Slippage model:
    - Two options: deterministic adverse slippage = k*ATR; or stochastic: normal(mean=mu, sd=sigma) bounded to realistic limits.
    - Apply slippage on market fills and on limit fills when price gaps beyond limit within the bar.
- Partial fills & liquidity:
    - Model max_fill_pct per bar; if order size > liquidity * max_fill_pct then fill partially and carry remainder.
- Commission & fees:
    - Apply per-side commission or per-notional fee.
- Rollover/swap:
    - On daily bar close for positions held overnight, subtract swap = notional * swap_rate / 365. For FX weekends, apply 3-day swap on Wednesday close (broker convention).
- Slippage example (conservative):
    - slippage = max(0.05*ATR, spread*0.5)

### Backtester core loop (simplified)

```
for i in range(start_idx, end_idx):
    bar = df.iloc[i]
    # 1. strategy decisions at close
    new_orders = strategy.on_bar_close(i, history)
    # 2. submit orders to execution model
    for order in new_orders:
        engine.submit_order(order)
    # 3. execution model processes fills using bar's OHLC
    fills = engine.match_orders(bar)
    # 4. update portfolio PnL, margins, apply swaps
    portfolio.apply_fills(fills, bar.timestamp)
    portfolio.apply_daily_rollover(bar.timestamp)
    # 5. logging
    logger.record_state(i, portfolio, fills)
```

### Position sizing

- For FX, compute lot size from risk_amount and stop_distance_in_price:

```
def compute_lots(equity, risk_pct, stop_pips, pip_value_per_lot=10):
    risk_amount = equity * risk_pct
    lot_size = risk_amount / (stop_pips * pip_value_per_lot)
    return max(min_lot, round_to_tick(lot_size))
```

- For cross pairs where pip value varies, compute pip_value dynamically given quote pair and account currency.
- Always enforce margin checks: required_margin = notional / leverage; reject size if available_margin < required_margin * margin_buffer (e.g., 1.2).

### Logging and reconciliation

- Log raw signals and final executed trades separately.
- Trade log fields: trade_id, order_id, pattern_type, entry_time, entry_price, entry_fill_type, size, stop_price, tp_prices, exit_time, exit_price, pnl, fees, swap, reason_exit, exchange_fill_details.
- Snapshot account states periodically (balance, equity, margin_used, free_margin).
- Maintain audit log for every order event; use JSON or append-only CSV.

### Testing strategy components

- Unit tests:
  - Indicators: known inputs -> known outputs.
  - Pattern detection: use synthetic price series embedding canonical patterns and assert detection.

    - Execution model: known bar + order -> expected fill price and qty.
- Integration tests:
    - Run deterministic backtest on a short synthetic dataset and compare to hand-calculated expected PnL.
- Regression tests:
    - Store baseline equity curve and assert new changes produce within-tolerance differences.

### Example: Oanda-like REST adapter (conceptual)

- Keep adapter thin and idempotent; map strategy order model to broker API order types (limit, market, stop, OCO).
- Always reconcile: after submitting order, poll for status and record fills; do not assume immediate fill.
- Example pseudocode for submit:

```
def submit_order_oanda(oanda_client, order):
    resp = oanda_client.create_order(instrument=order.instrument, units=ord
    return resp['orderFillTransaction'] or resp['orderCreateTransaction']
```

- For live testing, use broker-provided practice environment.

### Data-driven pattern testing tips

- Use synthetic augmentation to ensure pattern detectors are robust: inject Gaussian noise, varying volatility, small gaps.
- Create canonical templates for patterns and run detection across scaled/shifted versions to validate thresholds.
- Maintain a corpus of annotated real patterns (with timestamps and type) for manual QA and supervised ML tasks.

### Performance and scaling

- For multi-instrument, multi-timeframe backtests:
    - Precompute indicators and patterns per instrument to avoid recomputation.
    - Use vectorized numpy/pandas operations; move heavy inner loops to numba or Cython if needed.
    - Persist intermediate results (indicators, detected pattern metadata) to disk for reuse.
- Memory: stream backtests over large datasets where possible; avoid loading many years of tick data into memory unless necessary.

### Pseudocode: limit-entry-with-ttl order handling

```
def attempt_limit_entry(order, bar, ttl):
    # order: {side, price, size, create_idx}
    if order.status == 'NEW':
```

```
        # check if limit price within intrabar range (accounting spread)
        if side == 'buy' and bar.low <= order.price <= bar.high:
            fill_price = order.price + slippage_adjustment()
            fill_qty = compute_possible_fill(order.size, bar_liquidity)
            order.fill(fill_qty, fill_price)
        elif side == 'sell' and bar.low <= order.price <= bar.high:
            fill_price = order.price - slippage_adjustment()
            ...
        else:
            # not filled; check TTL
            if current_idx - order.create_idx >= ttl:
                if strategy.ttl_policy == 'market_on_expire':
                    market_price = bar.open  # or mid
                    order.fill(order.size, market_price + slippage)
                else:
                    order.cancel()
```

### Monitoring, metrics and alerts (production)

- Real-time dashboards: equity, PnL/day, current positions, realized/unrealized PnL, margin usage, open orders.
- Alerts (email/Slack/pagerduty) for:
  - Unhandled exceptions in strategy loop
  - Leverage or margin threshold breach
  - Kill-switch trigger conditions (e.g., drawdown > X%)
  - Order rejection rates > threshold
- Health checks: ensure data feed latency < threshold, broker connectivity alive, heartbeat logs.

### Safety features and kill-switches (implementation notes)

- Global circuit breakers:
  - Equity drawdown threshold (e.g., stop trading if equity drops > 10% in X days).
  - Max consecutive losses threshold.
  - Max daily loss threshold.
- Per-strategy:
  - Pause trading if OOS metrics fall below acceptable bounds or if parameter drift observed.
- Implement emergency STOP_MODE where system only cancels orders and flattens positions.

### Example small code snippets

1. Compute ATR and EMA slope

```
import pandas as pd
def atr_wilder(df, length=14):
    high = df['high']; low = df['low']; close = df['close']
    tr = pd.concat([high - low, (high - close.shift()).abs(), (low - close.
    atr = tr.ewm(alpha=1/length, adjust=False).mean()
    return atr

def ema_slope(df, length=21):
    ema = df['close'].ewm(span=length, adjust=False).mean()
    slope = (ema - ema.shift(length)).fillna(0) / length
    return slope
```

1. Position sizing (FX lot calc, simplified)

```
def compute_lots_fx(equity, risk_pct, stop_pips, pip_value_per_lot=10):
    risk_amount = equity * risk_pct
    lots = risk_amount / (stop_pips * pip_value_per_lot)
    # round to broker min increment, enforce min/max
    return max(round(lots, 2), 0.01)
```

1. Simple backtest engine skeleton

```
class Backtester:
    def __init__(self, df, strategy, execution_model, portfolio):
        self.df = df
        self.strategy = strategy
        self.exec = execution_model
        self.portfolio = portfolio

    def run(self):
        for i in range(self.start, self.end):
            bar = self.df.iloc[i]
            orders = self.strategy.on_bar_close(i, self.df.iloc[:i+1])
            for o in orders: self.exec.submit(o)
            fills = self.exec.match(bar)
            self.portfolio.apply(fills, bar['timestamp'])
            self.portfolio.apply_rollover(bar['timestamp'])
            self.log_state(i)
```

## Reproducibility and configuration

- Use explicit config files (YAML/JSON) for parameters, not hard-coded constants.
- Version data and code; log git commit id in run metadata.
- Save random seeds and deterministic RNG if stochastic elements used.

### Example run metadata to record

- run_id, git_commit, start_date, end_date, timeframe, instruments, parameter_values, initial_equity, fees_model, spread_model, slippage_model, walk_forward_windows, seed.

### Best practices for debugging

- Start with synthetic, simplified datasets to validate logic (single known trade sequence).
- Plot trades overlaying chart to visually confirm entry/exit behavior.
- Compare expected vs actual fill prices in logs.
- Re-run a single-bar replay to step through strategy logic.

### Unit test ideas

- Confirm ATR and EMA outputs vs known library (TA-Lib) results.
- Feed canonical H&S pattern and assert detector returns expected indices.
- Simulate single limit order and verify TTL and market_on_expire behavior.
- Validate lot sizing across different account currencies and cross-pair pip values.

### Lightweight production checklist (deploy-ready)

- ☐ Code in git, with CI running unit tests
- ☐ Config files for live/paper/backtest separated
- ☐ Logging and monitoring wired (alerts, dashboards)
- ☐ Backtest with realistic execution model and walk-forward OOS validated
- ☐ Paper trading run for minimum N days/trades (configurable)
- ☐ Kill-switch and emergency stop implemented and tested
- ☐ Broker adapter tested in sandbox and reconciled with logs
- ☐ Rollout plan and rollback plan documented

### Appendix: Helpful libraries and tools

- pandas, numpy — data handling
- numba — speed-ups for heavy loops
- empyrical/pyfolio — performance metrics & tear sheets
- backtrader/zipline/fastquant — backtest frameworks (note: adapt execution models)
- ccxt — broker/exchange adapters (mostly crypto)
- oandapyV20, ib_insync — broker APIs

- matplotlib/plotly — charting
- pytest — unit testing

## End-of-chapter quick checklist

- Data cleaned and timezone-UTC enforced
- ATR(14) and other indicators validated
- Pattern detectors unit-tested on synthetic examples
- Execution model configured (spread, slippage, commission, rollover)
- Backtester deterministic and idempotent
- Logging and reconciliation implemented
- Paper trading performed before live

# Trading Rulesets

This chapter provides the practical artifacts to implement, test, and deploy the strategies and systems detailed earlier. Use the pseudocode, schemas, and checklists as templates; adapt parameter defaults and transaction-cost assumptions to your chosen broker and instrument liquidity.

### Reproducible backtest setup

1. Environment
    - Python 3.9+ with pinned libraries: pandas, numpy, scipy, statsmodels, ta (optional), lightgbm/xgboost (optional), backtesting library or custom engine, matplotlib/plotly for charts.
    - Data/version control: DVC or hashed filenames; store raw tick/1m then derive 4H/1D via resampling.
    - Secrets: API keys in vault (do not hardcode).
2. Data ingestion & cleaning (pseudocode)

```
def load_tick_data(path):
    df = read_csv(path, parse_dates=['timestamp'])
    df['timestamp'] = df['timestamp'].dt.tz_convert('UTC')
    df = df.sort_values('timestamp').drop_duplicates(['timestamp', 'bid', '
    return df

def tick_to_ohlcv(tick_df, timeframe='4H'):
    tick_df['mid'] = (tick_df['bid'] + tick_df['ask']) / 2
    ohlc = tick_df.set_index('timestamp').resample(timeframe).agg({
      'mid': ['first','max','min','last'],
      'bid': ['first','max','min','last'],
      'ask': ['first','max','min','last'],
      'volume': 'sum'  # if available or tick count
    })
    ohlc.columns = ['open','high','low','close','bid_o','bid_h','bid_l','bi
    ohlc = ohlc.dropna()
    return ohlc
```

1. Time alignment & session rules

- Use UTC; align 4H windows to 00:00 UTC boundaries. Ensure daylight savings handled at timezone conversion.

1. Transaction cost model (TCM)

- Components: spread, commission, slippage, swap.
- Example functions:

```
def apply_spread(entry_mid, side, spread_pips):
    if side == 'LONG':
        return entry_mid + spread_pips/2
    else:
        return entry_mid - spread_pips/2

def sample_slippage(atr, mu=0, sigma_factor=0.1):
    return np.random.normal(loc=mu, scale=sigma_factor*atr)

def apply_commission(notional, commission_per_lot):
    return commission_per_lot * (notional / STANDARD_LOT_NOTIONAL)
```

1. Fill model for limit orders

- Fill probability function depends on limit distance, volatility, time-in-force:

```
def limit_fill_prob(distance, atr, wait_candles):
    # distance in price units from mid at order creation
    z = distance / atr
    base_prob = np.exp(-max(0,z*2))  # heuristic
    time_factor = min(1.0, 0.2 + 0.4*wait_candles)
    return base_prob * time_factor
```

1. Backtest engine essentials

- Deterministic candle-close decision points; event-driven order matching using next-candle open or modeled fills; trade-level logging with MAE/MFE; reusable simulation seeds.

## Example rulesets (concise, codable)

1. Trend Breakout (4H breakout with 1D trend filter) — baseline

- Entry: 4H close > 4H rolling max(20) and 1D close > EMA20_1D.
- Entry order: limit at close − 0.25*ATR.
- SL: entry − 2*ATR; TP: 3*ATR (50% at TP).
- Sizing: risk 0.5% equity per trade.
- Time exit: 10 days.

1. Mean-Reversion (pairs or single-pair z-score)

- For pair (A,B):
  - Compute spread = log(A) − β*log(B) with β from OLS over rolling 180-day window.
  - z = (spread − rolling_mean(30)) / rolling_std(30).
  - Entry: if z > +2 → SHORT spread (short A, long B); if z < −2 → LONG spread.
  - SL: stop if z reverts beyond 0 or adverse move of half-life*2.

- Size: hedge ratio from β; risk per leg total 0.5% equity.
- Exit: z crosses 0 or max 10 days.

1. Volatility-Adaptive Momentum

- Entry: rank recent 4H returns (n=5) vs ATR; enter top decile signals only.
- Position sizing scaled by target_volatility / realized_vol (ATR proxy), capped at 2x.
- Use wider SL (3*ATR) due to larger size.

1. News-Aware Event Strategy

- Pre-scan economic calendar for high-impact events (±24h). Block new entries within 12h before event and 24h after.
- If holding during event, reduce position by 50% or set wider SLs.

### Detailed pseudocode: unified trade engine (event-driven)

```
class StrategyEngine:
    def __init__(self, config, data):
        self.data = data  # dict of pair -> DataFrame (4H)
        self.config = config
        self.equity = config.starting_equity
        self.positions = {}  # pair -> Position object
        self.trades = []  # list of trade dicts
        self.order_id_seq = 0

    def run_backtest(self):
        for t in index_of_all_4H_candles():
            self.on_new_candle(t)
            self.match_orders(t)
            self.apply_daily_swap_if_midnight(t)
            self.reconcile(t)

    def on_new_candle(self, t):
        for pair, df in self.data.items():
            candle = df.loc[t]
            self.update_indicators(pair, candle)
            self.evaluate_entries(pair, candle)
            self.evaluate_exits(pair, candle)

    def evaluate_entries(self, pair, candle):
        if not self.positions.get(pair) and self.passes_filters(pair):
            if self.entry_signal(pair):
                entry_price = candle.close
                limit_price = entry_price - 0.25*self.atr(pair)
                order = self.create_limit_order(pair, size=self.calc_size(p
```

```
            self.send_order(order)

    def match_orders(self, t):
        for order in outstanding_orders():
            fill_prob = limit_fill_prob(abs(order.price - market_mid(t)), a
            if random() < fill_prob:
                fill_price = simulate_fill_price(order, t)
                self.execute_fill(order, fill_price, t)
            elif order.age > config.limit_window:
                # fallback to market
                market_price = get_open_price(order.pair, t+1)
                self.execute_fill(order, market_price, t+1)

    def execute_fill(self, order, price, t):
        # create Position or adjust existing
        pos = Position(pair=order.pair, entry_price=price, size=order.size,
        pos.stop = compute_SL(price, atr_at_time(order.pair, t))
        # log trade creation
        self.trades.append(trade_record(...))
        update_equity_on_fill(...)

    def evaluate_exits(self, pair, candle):
        pos = self.positions.get(pair)
        if pos:
            if candle.low <= pos.stop: self.close_position(pair, pos.stop,
            elif candle.high >= pos.take_profit: self.partial_close(...)
            elif pos.age >= config.max_holding_days: self.close_position(pa

    # additional methods: calc_size, apply_swap, reconcile, reporting...
```

### Trade & backtest logging schema

Recommended table columns:

- trade_id, strategy, pair, side, entry_time, entry_price, entry_mid, entry_spread, entry_size, stop_price, tp_price, exit_time, exit_price, exit_mid, exit_spread, exit_size, realized_pnl, commission, slippage, swap_total, MAE, MFE, max_drawdown_during_trade, holding_period_days, reason_exit, fill_quality_flag.

Persist logs as compressed Parquet and export daily CSV snapshot for reconciliation.

### Example parameter grids for sensitivity testing

- ATR multiplier for SL: [1.5, 2.0, 2.5, 3.0]
- Entry_buffer: [0.1*ATR, 0.25*ATR, 0.5*ATR]
- TP multiples: [2.0, 3.0, 4.0]

- Max holding days: [5, 10, 15]
- Risk per trade: [0.25%, 0.5%, 1.0%] Run combinatorial experiments with purged CV and summarize via heatmaps (Chapter 20).

## Quick reference code snippets

1. ATR(14) with pandas:

```
high, low, close = df['high'], df['low'], df['close']
tr1 = high - low
tr2 = (high - close.shift(1)).abs()
tr3 = (low - close.shift(1)).abs()
tr = pd.concat([tr1, tr2, tr3], axis=1).max(axis=1)
atr14 = tr.rolling(window=14, min_periods=14).mean()
```

1. EMA:

```
ema20 = close.ewm(span=20, adjust=False).mean()
```

1. Rolling max/min:

```
rolling_high20 = high.rolling(window=20, min_periods=20).max()
rolling_low20 = low.rolling(window=20, min_periods=20).min()
```

1. Half-life estimation (AR(1)):

```
delta_spread = spread.diff().dropna()
spread_lag = spread.shift(1).dropna()
phi = np.polyfit(spread_lag, delta_spread, 1)[0] + 1
half_life = -np.log(2) / np.log(phi) if phi>0 else np.inf
```

## Representative references

- P. Wilmott — “Paul Wilmott Introduces Quantitative Finance” (for risk/measure foundations).
- E. Chan — “Algorithmic Trading: Winning Strategies and Their Rationale” (market-neutral and pairs trading).
- T. Haug — “The Complete Guide to Option Pricing Formulas” (for risk modeling background — optional).
- Avellaneda & Lee — “Statistical arbitrage in the US equities market” (pairs/co-integration methods).
- Lopez de Prado — “Advances in Financial Machine Learning” (purged CV, labeling, ML pitfalls).
- Andersen et al. — “Volatility and time series models” (for volatility modeling).
- FX industry sources: broker documentation for spreads, swap schedules, and API docs (specific broker manual).
- Official regulators: CFTC, NFA, FCA, ESMA — for compliance checks.

## Appendix: Example small backtest command-line flow

- Steps:
    1. python fetch_data.py –pairs EURUSD GBPUSD –from 2016-01-01 –to 2025-11-01 –granularity 1m
    2. python resample.py –input tick_1m.parquet –to 4H,1D –tz UTC
    3. python backtest.py –strategy trend_breakout.json –tc_model tcm.yaml –out results/trend_4H
    4. python analyze.py –input results/trend_4H –plots equity,mae_mfe,heatmap
    5. python walkforward.py –config wf_config.yaml –runs 50

Include CI tests that run a mini backtest on a short sample to validate code paths.

## Final implementation checklist before live deployment

- Full unit test coverage for indicator functions, sizing, and order-state transitions.
- End-to-end integration test using demo broker (fills, cancels, reconnects).
- Paper trading for minimum 60 days with identical execution code.
- Documentation: runbook for incidents, daily reconciliation script, and team alerting channel.
- Legal & tax: KYC completed, tax advisor consulted, broker agreement reviewed.

# Implementation Notes

Chapter summary: provides concrete, prescriptive, codable guidance and concise Python-ready snippets for implementing 4H/1D automated FX trading systems: data handling, indicator computation, pattern detection hooks, backtest engine primitives, execution modeling (spread/slippage/rollover), position sizing, logging, and recommended project structure. All examples assume UTC candles, ATR(14) defaults, risk_per_trade = 0.5%, deterministic decisioning at candle close, and limit-entry-with-ttl behavior.

## Suggested project layout

- README.md
- requirements.txt
- data/
    - raw/
    - cleaned/
    - resampled/
- src/
    - data_io.py
    - indicators.py
    - patterns.py
    - execution_model.py
    - backtester.py
    - portfolio.py
    - sizing.py
    - reporting.py
    - broker_adapters/
        - oanda_adapter.py (example)
- tests/
    - test_indicators.py
    - test_patterns.py
    - test_backtester.py
- notebooks/
    - exploratory.ipynb
- logs/
- results/

## Key implementation principles

- Deterministic: decisions computed using only data up to current candle close.
- Idempotent order lifecycle: design orders/trades as state machines that can be safely replayed.
- Tight logging and reconciliation: log both strategy decisions and broker fill events with unique ids and timestamps.

- Modularity: separate data, signal generation, execution modeling, and portfolio sizing.
- Simulate execution realistically: model spread, slippage, partial fills, and rollovers.
- Unit tests: for indicators, pivot/pattern detectors, order/state machine behaviors.
- Configurable defaults: expose ATR_len, risk_per_trade, breakout_margin, max_wait_candles via config.

### Data ingestion and cleaning

- Use standardized CSV/Parquet input with columns: timestamp(UTC), open, high, low, close, volume, tick_count(optional).
- Verify monotonic timestamps, no duplicate indices, and consistent timezone (UTC).
- Fill missing bars explicitly (for FX 24/5, missing weekend bars are expected; ensure consistent handling).
- Resampling: if you ingest higher-frequency ticks or minutes, aggregate using:
    - open = first
    - high = max
    - low = min
    - close = last
    - volume = sum
- Save cleaned/resampled files to avoid repeated preprocessing.

### Indicators

- Use pandas and numba where heavy; prefer vectorized rolling functions.
- ATR(14) implementation (pandas):

```
def atr(df, length=14):
    high_low = df['high'] - df['low']
    high_close = (df['high'] - df['close'].shift()).abs()
    low_close = (df['low'] - df['close'].shift()).abs()
    tr = pd.concat([high_low, high_close, low_close], axis=1).max(axis=1)
    atr = tr.rolling(length, min_periods=1).mean()  # or Wilder EMA
    return atr
```

- Pivot detection (5-bar pivot example):

```
def is_pivot_high(series, i, k=2):
    return series[i] == series[i-k:i+k+1].max() and (series[i] > series[i-1
```

- Compute EMA/MA slopes using linear regression over last N bars for robust slope.

### Pattern detection

- Expose pattern detectors that return:
    - pattern_found (bool)

    - pattern_metadata: dict with key points (pivot indices/levels), neckline/lines, height, symmetry, suggested entry/stop/tps
- Keep detectors pure functions: input = dataframe slice + indicator values; output = metadata only.

Example pattern function signature:

```
def detect_head_shoulders(df_slice, atr_series, params):
    # returns (found:bool, meta:dict)
```

### Order and execution model

- Represent orders and positions as records with unique ids, status, timestamps, side, size, price, stop, tp, ttl, filled_qty, avg_fill_price.
- Order lifecycle states: NEW -> WORKING -> PARTIALLY_FILLED -> FILLED -> CANCELED -> REJECTED.
- For backtests, run an engine that iterates bars and:
    1. On bar close, run strategy to emit new orders.
    2. Feed orders into execution_model that simulates fills given bar OHLC and configured spread/slippage/liquidity rules.
    3. Update positions and ledger, log trades.
- Deterministic limit-order policy: if limit price is within bar high/low adjusted for spread, simulate fill at limit price (or better) with slippage applied if model requires.

### Execution realism primitives

- Spread model:
    - Support fixed per-instrument spread (pips) or time-varying spread series.
    - For each trade, compute effective buy_price = mid + spread/2, sell_price = mid - spread/2 for market fills. For limit fills, adjust threshold accordingly.
- Slippage model:
    - Two options: deterministic adverse slippage = k*ATR; or stochastic: normal(mean=mu, sd=sigma) bounded to realistic limits.
    - Apply slippage on market fills and on limit fills when price gaps beyond limit within the bar.
- Partial fills & liquidity:
    - Model max_fill_pct per bar; if order size > liquidity * max_fill_pct then fill partially and carry remainder.
- Commission & fees:
    - Apply per-side commission or per-notional fee.
- Rollover/swap:
    - On daily bar close for positions held overnight, subtract swap = notional * swap_rate / 365. For FX weekends, apply 3-day swap on Wednesday close (broker convention).
- Slippage example (conservative):
    - slippage = max(0.05*ATR, spread*0.5)

### Backtester core loop (simplified)

```
for i in range(start_idx, end_idx):
    bar = df.iloc[i]
    # 1. strategy decisions at close
    new_orders = strategy.on_bar_close(i, history)
    # 2. submit orders to execution model
    for order in new_orders:
        engine.submit_order(order)
    # 3. execution model processes fills using bar's OHLC
    fills = engine.match_orders(bar)
    # 4. update portfolio PnL, margins, apply swaps
    portfolio.apply_fills(fills, bar.timestamp)
    portfolio.apply_daily_rollover(bar.timestamp)
    # 5. logging
    logger.record_state(i, portfolio, fills)
```

### Position sizing

- For FX, compute lot size from risk_amount and stop_distance_in_price:

```
def compute_lots(equity, risk_pct, stop_pips, pip_value_per_lot=10):
    risk_amount = equity * risk_pct
    lot_size = risk_amount / (stop_pips * pip_value_per_lot)
    return max(min_lot, round_to_tick(lot_size))
```

- For cross pairs where pip value varies, compute pip_value dynamically given quote pair and account currency.
- Always enforce margin checks: required_margin = notional / leverage; reject size if available_margin < required_margin * margin_buffer (e.g., 1.2).

### Logging and reconciliation

- Log raw signals and final executed trades separately.
- Trade log fields: trade_id, order_id, pattern_type, entry_time, entry_price, entry_fill_type, size, stop_price, tp_prices, exit_time, exit_price, pnl, fees, swap, reason_exit, exchange_fill_details.
- Snapshot account states periodically (balance, equity, margin_used, free_margin).
- Maintain audit log for every order event; use JSON or append-only CSV.

### Testing strategy components

- Unit tests:
  - Indicators: known inputs -> known outputs.
  - Pattern detection: use synthetic price series embedding canonical patterns and assert detection.

    - Execution model: known bar + order -> expected fill price and qty.
- Integration tests:
    - Run deterministic backtest on a short synthetic dataset and compare to hand-calculated expected PnL.
- Regression tests:
    - Store baseline equity curve and assert new changes produce within-tolerance differences.

### Example: Oanda-like REST adapter (conceptual)

- Keep adapter thin and idempotent; map strategy order model to broker API order types (limit, market, stop, OCO).
- Always reconcile: after submitting order, poll for status and record fills; do not assume immediate fill.
- Example pseudocode for submit:

```
def submit_order_oanda(oanda_client, order):
    resp = oanda_client.create_order(instrument=order.instrument, units=ord
    return resp['orderFillTransaction'] or resp['orderCreateTransaction']
```

- For live testing, use broker-provided practice environment.

### Data-driven pattern testing tips

- Use synthetic augmentation to ensure pattern detectors are robust: inject Gaussian noise, varying volatility, small gaps.
- Create canonical templates for patterns and run detection across scaled/shifted versions to validate thresholds.
- Maintain a corpus of annotated real patterns (with timestamps and type) for manual QA and supervised ML tasks.

### Performance and scaling

- For multi-instrument, multi-timeframe backtests:
    - Precompute indicators and patterns per instrument to avoid recomputation.
    - Use vectorized numpy/pandas operations; move heavy inner loops to numba or Cython if needed.
    - Persist intermediate results (indicators, detected pattern metadata) to disk for reuse.
- Memory: stream backtests over large datasets where possible; avoid loading many years of tick data into memory unless necessary.

### Pseudocode: limit-entry-with-ttl order handling

```
def attempt_limit_entry(order, bar, ttl):
    # order: {side, price, size, create_idx}
    if order.status == 'NEW':
```

```
        # check if limit price within intrabar range (accounting spread)
        if side == 'buy' and bar.low <= order.price <= bar.high:
            fill_price = order.price + slippage_adjustment()
            fill_qty = compute_possible_fill(order.size, bar_liquidity)
            order.fill(fill_qty, fill_price)
        elif side == 'sell' and bar.low <= order.price <= bar.high:
            fill_price = order.price - slippage_adjustment()
            ...
        else:
            # not filled; check TTL
            if current_idx - order.create_idx >= ttl:
                if strategy.ttl_policy == 'market_on_expire':
                    market_price = bar.open  # or mid
                    order.fill(order.size, market_price + slippage)
                else:
                    order.cancel()
```

### Monitoring, metrics and alerts (production)

- Real-time dashboards: equity, PnL/day, current positions, realized/unrealized PnL, margin usage, open orders.
- Alerts (email/Slack/pagerduty) for:
  - Unhandled exceptions in strategy loop
  - Leverage or margin threshold breach
  - Kill-switch trigger conditions (e.g., drawdown > X%)
  - Order rejection rates > threshold
- Health checks: ensure data feed latency < threshold, broker connectivity alive, heartbeat logs.

### Safety features and kill-switches (implementation notes)

- Global circuit breakers:
  - Equity drawdown threshold (e.g., stop trading if equity drops > 10% in X days).
  - Max consecutive losses threshold.
  - Max daily loss threshold.
- Per-strategy:
  - Pause trading if OOS metrics fall below acceptable bounds or if parameter drift observed.
- Implement emergency STOP_MODE where system only cancels orders and flattens positions.

### Example small code snippets

1. Compute ATR and EMA slope

```
import pandas as pd
def atr_wilder(df, length=14):
    high = df['high']; low = df['low']; close = df['close']
    tr = pd.concat([high - low, (high - close.shift()).abs(), (low - close.
    atr = tr.ewm(alpha=1/length, adjust=False).mean()
    return atr

def ema_slope(df, length=21):
    ema = df['close'].ewm(span=length, adjust=False).mean()
    slope = (ema - ema.shift(length)).fillna(0) / length
    return slope
```

1. Position sizing (FX lot calc, simplified)

```
def compute_lots_fx(equity, risk_pct, stop_pips, pip_value_per_lot=10):
    risk_amount = equity * risk_pct
    lots = risk_amount / (stop_pips * pip_value_per_lot)
    # round to broker min increment, enforce min/max
    return max(round(lots, 2), 0.01)
```

1. Simple backtest engine skeleton

```
class Backtester:
    def __init__(self, df, strategy, execution_model, portfolio):
        self.df = df
        self.strategy = strategy
        self.exec = execution_model
        self.portfolio = portfolio

    def run(self):
        for i in range(self.start, self.end):
            bar = self.df.iloc[i]
            orders = self.strategy.on_bar_close(i, self.df.iloc[:i+1])
            for o in orders: self.exec.submit(o)
            fills = self.exec.match(bar)
            self.portfolio.apply(fills, bar['timestamp'])
            self.portfolio.apply_rollover(bar['timestamp'])
            self.log_state(i)
```

## Reproducibility and configuration

- Use explicit config files (YAML/JSON) for parameters, not hard-coded constants.
- Version data and code; log git commit id in run metadata.
- Save random seeds and deterministic RNG if stochastic elements used.

### Example run metadata to record

- run_id, git_commit, start_date, end_date, timeframe, instruments, parameter_values, initial_equity, fees_model, spread_model, slippage_model, walk_forward_windows, seed.

### Best practices for debugging

- Start with synthetic, simplified datasets to validate logic (single known trade sequence).
- Plot trades overlaying chart to visually confirm entry/exit behavior.
- Compare expected vs actual fill prices in logs.
- Re-run a single-bar replay to step through strategy logic.

### Unit test ideas

- Confirm ATR and EMA outputs vs known library (TA-Lib) results.
- Feed canonical H&S pattern and assert detector returns expected indices.
- Simulate single limit order and verify TTL and market_on_expire behavior.
- Validate lot sizing across different account currencies and cross-pair pip values.

### Lightweight production checklist (deploy-ready)

- ☐ Code in git, with CI running unit tests
- ☐ Config files for live/paper/backtest separated
- ☐ Logging and monitoring wired (alerts, dashboards)
- ☐ Backtest with realistic execution model and walk-forward OOS validated
- ☐ Paper trading run for minimum N days/trades (configurable)
- ☐ Kill-switch and emergency stop implemented and tested
- ☐ Broker adapter tested in sandbox and reconciled with logs
- ☐ Rollout plan and rollback plan documented

### Appendix: Helpful libraries and tools

- pandas, numpy — data handling
- numba — speed-ups for heavy loops
- empyrical/pyfolio — performance metrics & tear sheets
- backtrader/zipline/fastquant — backtest frameworks (note: adapt execution models)
- ccxt — broker/exchange adapters (mostly crypto)
- oandapyV20, ib_insync — broker APIs

- matplotlib/plotly — charting
- pytest — unit testing

## End-of-chapter quick checklist

- Data cleaned and timezone-UTC enforced
- ATR(14) and other indicators validated
- Pattern detectors unit-tested on synthetic examples
- Execution model configured (spread, slippage, commission, rollover)
- Backtester deterministic and idempotent
- Logging and reconciliation implemented
- Paper trading performed before live

# Common Failures

Chapter summary: overviews the most common operational, modeling, and behavioral failure modes for automated 4H/1D Forex strategies, explains why they occur, introduces detection signals, and provides concrete mitigations (codable where applicable). Each item includes quick checks and config actions to prevent or recover.

Structure - Model & research failures - Data failures - Execution & broker failures - Risk & sizing failures - Operational & deployment failures - Behavioral/organizational failures - Quick detection rules and automated mitigations - Recovery playbook snippets

## A. Model & research failures

1. Overfitting / Data snooping

- Why: excessive parameter tuning on historical data, selecting parameters that exploit noise.
- Detection signals:
  - Large performance gap between IS and OOS.
  - Extreme sensitivity to small parameter changes.
  - High number of parameters relative to trades.
- Mitigations:
  - Use walk-forward, purged k-fold CV, nested CV.
  - Limit free parameters; prefer coarser parameter grids.
  - Require minimum N trades in OOS (rule: N >= 100 or combine multiple instruments).
  - Apply bootstrap p-values and report confidence intervals.
- Codable checks:
  - compute_oos_degradation = (IS_sharpe - OOS_sharpe) / IS_sharpe; if > 0.4 flag for review.

1. Survivorship / Sample selection bias

- Why: selecting instruments or periods that remove poor performers or using truncated datasets.
- Detection:
  - Sudden missing instruments in live vs backtest.
- Mitigation:
  - Use full history, document instrument selection criteria, apply out-of-time holdouts.

1. Lookahead leakage

- Why: using future information by mistake (shift errors, improper rolling windows).

- Detection:
    - Strategy signals referencing indicator values that change after entry when recomputed.
- Mitigation:
    - Enforce indicator code tests: for each index i, ensure signal(i) depends only on df[:i+1].
    - Use unit tests that assert no future-slice is used.

1. Ignoring nonstationarity / regime shifts

- Why: market behavior changes, rendering past patterns less predictive.
- Detection:
    - Rolling performance degradation, changes in trade distribution or volatility.
- Mitigation:
    - Implement regime detection (ATR quantiles, ADX/trend vs range detectors).
    - Use adaptive parameters (re-tune on rolling windows) but validate via walk-forward.
- Codable example:
    - If ATR_now > ATR_median * 3 or < ATR_median * 0.5, pause trading or tighten thresholds.

## B. Data failures

1. Missing or duplicated bars

- Why: vendor glitches, timezone mismatches.
- Detection:
    - Non-monotonic timestamps, duplicated indices, gaps longer than expected.
- Mitigation:
    - Data ingestion validation: enforce monotonic timestamps, detect gaps and either fill or mark as invalid.
- Codable check:
    - if any(diff(timestamps) != expected_interval): alert + pause/reconcile.

1. Bad ticks / extreme spikes

- Why: feed errors, outliers.
- Detection:
    - Single-bar moves > X*ATR or > Y% change.
- Mitigation:
    - Outlier filtering: cap per-bar moves or discard bars if flagged; prefer using tick-level aggregation and volume checks.

- Codable rule:
    - if bar_range > 10*ATR: mark bar as suspect, exclude from indicator updates, log.

1. Timezone and DST errors

- Why: mixing local broker time and UTC.
- Detection:
    - Candle alignment mismatch vs broker statements.
- Mitigation:
    - Standardize on UTC; convert broker times on ingest; unit tests for timezone offsets.

## C. Execution & broker failures

1. Unrealistic fill assumptions

- Why: assuming midprice fills or ignoring spread and slippage.
- Detection:
    - Backtest fills consistently better than paper/live fills.
- Mitigation:
    - Model spread, slippage, partial fills; calibrate models using paper/canary fills.
- Codable monitor:
    - track_slippage = realized_fill_price - expected_entry_price; alert if mean_slippage > threshold.

1. Order rejections and stalled orders

- Why: broker limits, invalid params, connectivity.
- Detection:
    - Spike in rejection count, high TTL expirations.
- Mitigation:
    - Pre-check order size/margin, rate limits, and validate responses; implement retry/backoff and alerting.
- Codable rule:
    - if rejection_rate_last_N > 0.02: pause new orders and notify.

1. Partial fills and liquidity mismatch

- Why: large notional vs market depth, broker limitations.
- Detection:
    - Repeated partial fills, large unfilled quantities.
- Mitigation:
    - Limit max notional per order, implement slicing/vwap-like execution for large fills.

- Codable: compute_fill_ratio = filled_qty / requested_qty; if < min_fill_ratio, flag.

1. Broker-side rule differences (stop behavior, swap conventions)

- Why: stop loss types, slippage at broker, swap roll 3-day on Wed.
- Detection:
    - Discrepancies between expected and broker behavior in sandbox.
- Mitigation:
    - Read broker docs, test in sandbox, make adapter to map local orders to broker specifics.

## D. Risk & sizing failures

1. Margin exhaustion and unexpected leverage effects

- Why: ignoring margin implications across correlated positions.
- Detection:
    - Rapidly increasing margin_used, margin call warnings.
- Mitigation:
    - Compute portfolio margin usage before order acceptance; enforce margin buffers (e.g., available_margin >= required_margin * 1.2).
- Codable rule:
    - if projected_margin_post_trade > free_margin / margin_buffer: reject order.

1. Position concentration / correlation risk

- Why: multiple pairs highly correlated produce effective overexposure.
- Detection:
    - High realized correlation across open positions; aggregated directional exposure.
- Mitigation:
    - Limit net directional exposure per currency; compute currency-level exposure and cap it.
- Codable example:
    - compute_currency_exposure(); if any currency_abs_exposure > exposure_limit -> reject new trade.

1. Size creep / drift in lot sizing

- Why: compounding position sizing increases volatility beyond expectation.
- Detection:
    - Increasing average position sizes vs plan.
- Mitigation:
    - Enforce fixed max position size and max growth per period; cap lot size to percent of equity.

## E. Operational & deployment failures

1. Missing monitoring, late alerts

- Why: alerts misconfigured or suppressed.
- Detection:
  - Incidents discovered late by humans or external parties.
- Mitigation:
  - Test alerting channels, use synthetic checks, require alert ack workflow.
- Codable test:
  - schedule_synthetic_alert_test() weekly; require ack.

1. Inadequate testing before deploy

- Why: skipping paper testing or insufficient test coverage.
- Detection:
  - Unexpected behavior on live canary, failing reconciliation.
- Mitigation:
  - Mandatory paper-trading phase with documented acceptance criteria and automated CI gating.

1. Insufficient change control

- Why: ad-hoc parameter changes without review.
- Detection:
  - Sudden performance changes correlated to config change times.
- Mitigation:
  - Require PR review, record parameter change logs, require rollback plan.

## F. Behavioral/organizational failures

1. Manual overrides and cookbook bias

- Why: human overrides that invalidate audit trail.
- Detection:
  - Discrepancies between strategy intent and executed trades with manual notes.
- Mitigation:
  - Record manual actions; require two-person approval for non-trivial overrides; maintain immutable audit log.

1. Over-trading after small wins / revenge trading

- Why: human behavioral response to short-term outcomes.
- Detection:
  - Increased trade frequency after drawdowns or large wins.

- Mitigation:
    - Enforce trading limits and automated cooldown timers after X consecutive trades or losses.

## G. Quick detection rules (to implement as automated health checks)

- Data freshness: if now - last_bar_time > allowed_lag -> raise CRITICAL.
- High rejection rate: if rejections_last_100 / 100 > 0.02 -> WARNING.
- Slippage spike: if mean_slippage_24h > baseline_slippage * 2 -> WARNING.
- Equity drawdown: if drawdown_from_peak >= auto_suspend_threshold -> CRITICAL -> auto-suspend.
- Consecutive losses: if consecutive_losses >= N_stop -> SUSPEND.
- Margin buffer: if free_margin / required_margin < margin_buffer_limit -> ALERT.

## H. Automated mitigations (codable actions)

- Auto-suspend trading: set system state to SUSPENDED, cancel working orders, notify ops.
- Throttle sizing: reduce max allowed lot by factor f (e.g., 0.5) when slippage spikes.
- Increase pattern thresholds: raise min_height_multiplier from 1.2->1.6 when false breakout rate high.
- Switch to conservative execution: convert limit entries to passive-only or increase ttl.
- Temporarily disable strategies per instrument when instrument-specific anomalies detected.

## I. Recovery playbook snippets (short)

1. Data outage recovery

- Action: pause new order submission; replay last known good data; reconcile with broker; resume once data stable.

1. Execution degradation (slippage/rejections)

- Action: throttle sizes, pause auto-increases, run diagnostics, escalate to ops.

1. Unexpected drawdown

- Action: enter REVIEW_MODE (pause new trades), run quick performance compare (last N trades vs expected), if drawdown triggers breach then SUSPEND live trading and alert.

1. Margin call near

- Action: initiate emergency flatten per runbook if margin_imminent flag true; else reduce leverage and new orders.

### J. Root-cause investigation checklist

- Collect: full logs, order/trade ledger, market data for window, system metrics (latency, CPU), recent config changes, git commit id.
- Reproduce: run backtest on exact timeframe with same execution parameters; attempt to reproduce fills.
- Patch: implement fix in staging, run regression tests, then deploy with canary.

### K. Example codable health-check rules (Python-like)

```
def health_checks(state):
    if now - state.last_bar_ts > timedelta(minutes=10):
        alert('DATA_STALE')
    if state.rejection_rate_100 > 0.02:
        alert('HIGH_REJECTION')
    if state.mean_slippage_24h > 2 * state.expected_slippage:
        throttle_sizes(0.5); alert('SLIPPAGE_SPIKE')
    if state.drawdown_pct >= auto_suspend_threshold:
        suspend_trading('DRAWDOWN_LIMIT')
```

### L. Post-incident improvements (must track)

- Update tests to cover the root cause.
- Add new monitoring metrics if missing.
- Update runbooks and assign owners.
- Schedule retrospective with stakeholders.

### End-of-chapter checklist

Use before resuming live after an incident: - Incident fully documented with timeline and root cause - Tests updated and CI passing - Monitoring/alerts added or tuned - Runbook updated; owners assigned - Canary/replay run completed and validated

### Deployment, Monitoring, and Safety

Chapter summary: provides a prescriptive, operational blueprint to move from backtest to paper and live trading, plus monitoring, alerting, and safety controls. Covers deployment stages, infrastructure, observability, incident response, and operational checklists tuned for 4H/1D Forex bots (10-day swing horizon).

### Deployment stages

1. Local dev & unit tests
2. Backtest + walk-forward + stress tests
3. Paper trading (sandbox/practice broker)
4. Limited live roll-out (canary)
5. Full production

## Infrastructure and architecture (recommended)

- Separation of concerns:
  - Data ingestion layer (market data, instrument metadata, swap rates)
  - Strategy engine (signal generation; stateless functions if possible)
  - Execution layer (order management, fills, broker adapter)
  - Risk & position manager (sizing, margin checks, leverage)
  - Ledger & persistence (trade log, account snapshots)
  - Monitoring & alerting (metrics, dashboards, notifications)
  - Orchestration (scheduler, runbooks)
- Deployment models:
  - Cloud VPS (AWS, GCP, Azure) or on-prem VM — ensure low-latency to broker where important; for 4H/1D latency is less critical than reliability.
  - Containerize services (Docker) for reproducibility; use Kubernetes for multi-instance scalability and rolling upgrades.
- Database and storage:
  - Time-series DB or parquet/CSV for historical candles; transactional DB (Postgres) for orders/trades; S3 for backups.
- Secrets management:
  - Store API keys in secret manager (Vault, AWS Secrets Manager). Do not commit keys to git.

## Run modes and environments

- dev: unit tests, synthetic data, debug logging
- staging/paper: real market data, sandbox broker account, full execution model
- prod/live: real money, limited access, stricter monitoring
- Ensure config-driven environments; no hard-coded environment checks in logic.

## Deployment checklist (pre-live)

- ☐ All unit/integration tests pass
- ☐ Backtest & walk-forward documented and reproducible
- ☐ Execution model validated against broker sandbox
- ☐ Paper trading run for min_duration and/or min_trades:
  - Suggestions: 4–8 weeks or 50–200 trades, whichever longer, for 4H/1D strategies
- ☐ Monitoring & alerts configured
-

- ☐ Disaster recovery & rollback plan documented
- ☐ Legal and brokerage requirements satisfied (KYC, account types)
- ☐ Team access control and runbook availability

## Order management and reconciliation

- Idempotent order submission: attach client_order_id and store local order state; on reconnection reconcile with broker state by querying open orders and fills.
- Reconciliation frequency: at every strategy loop iteration and periodic full-day reconciliation.
- Reconciliation fields: broker_order_id, client_order_id, submitted_price, filled_qty, avg_fill_price, fees, swaps, timestamps.
- Keep immutable append-only trade log for audit; include git commit id and config snapshot for each run.

## Execution policies for live trading

- Limit-first preference: submit limit orders with ttl and fall back to market only under strict rules.
- Max slippage / fill tolerance: reject market entry if estimated slippage > threshold or liquidity insufficient.
- Max notional per order: set conservative caps per instrument; split large orders into chunks if needed.
- Pre-checks before placing order:
    - Sufficient free margin
    - No conflicting open orders (e.g., duplicate pattern signals)
    - Not in “suspended” mode due to high-system risk
- Post-fill actions:
    - Immediately set hard stop orders at broker (if supported) and OCO TP orders; log both local and broker ids.
    - Do not rely solely on local process to enforce stops—use broker-side stops where possible.

## Monitoring and observability

- Metrics to collect (real-time time-series):
    - System health: process heartbeat, message queue length, data feed lag, API latency
    - Strategy health: open positions, realized/unrealized PnL, exposure, margin_used, percent_time_in_market
    - Execution metrics: order submission rate, rejection rate, avg fill latency, average slippage, TTL expirations

    - Risk metrics: current drawdown, day_loss, consecutive_losses, largest_open_position
- Logs:
    - Structured logs (JSON) with context: run_id, timestamp, level, module, message, correlation ids.
    - Retain logs for at least 90 days for audits (longer if required by compliance).
- Dashboards:
    - Equity curve + drawdown
    - Live positions and PnL by instrument
    - Recent trades list with reason for exit
    - Alerts stream
- Heartbeat & synthetic checks:
    - Regularly scheduled synthetic trades or mock order submissions to validate path to broker (in sandbox).
    - Data freshness check: alert if last-candle timestamp older than threshold.

## Alerting and notification design

- Alert severity levels:
    - INFO: noncritical events (paper->live transitions)
    - WARNING: recoverable issues (minor API latency spikes, minor rejections)
    - CRITICAL: immediate operator action required (broker disconnect, margin breach, kill-switch triggered)
- Delivery channels:
    - Slack/email for INFO/WARN
    - PagerDuty/SMS for CRITICAL
- Example alert rules:
    - CRITICAL: free_margin < margin_threshold (e.g., 1.1x maintenance margin)
    - CRITICAL: equity drawdown > stop_trading_drawdown (configurable)
    - CRITICAL: > X consecutive order rejections in last Y minutes
    - WARNING: data feed lag > 2x expected bar interval
    - WARNING: fill_slippage_mean > baseline_slippage * 2 for last N fills
- Alert payload: include run_id, timestamp, current equity, recent trades, last 5 log lines, suggested runbook action.

## Safety controls and kill-switches

- Global kill-switches (automated and manual):
    - Equity drawdown threshold: auto suspend trading if drawdown_from_peak >= D1 (e.g., 10%).
    - Daily loss limit: if realized loss_today >= D2 (e.g., 2% equity), stop trading for rest of day.
    - Consecutive loss count: if N consecutive losing trades >= N_max (e.g., 6), suspend strategy.
    - Max open positions: global upper bound to avoid overexposure.

    - Max leverage per instrument or aggregate portfolio.
- Automated response on kill:
    - Cancel all pending orders.
    - Option A (default): Do NOT auto-liquidate; alert operators and await manual confirmation to flatten (preference for human-in-loop).
    - Option B (configurable emergency): Aggressive flatten by market orders if imminent margin call.
- Manual kill:
    - Provide secure UI/CLI to set system to STOP_MODE; require two-person confirmation for live disabling in production (for risk control).

## Operational runbooks

- Data feed outage:
    1. If feed gap > threshold, pause new orders.
    2. If feed remains down > longer_threshold, cancel open limit orders and alert ops.
- Broker disconnect:
    1. Pause strategy execution.
    2. Attempt reconnection; if unsuccessful within timeout, alert on-call and keep positions with stop orders in broker.
- Unexpected large slippage event:
    1. Pause new trades, evaluate recent fills, alert ops.
    2. Consider scaling down size or halting for X hours.
- Margin call imminent:
    1. Preemptively reduce position sizes or flatten non-critical positions per policy.
    2. Alert finance/ops and supervisors.
- False positive pattern flood:
    1. Temporarily raise pattern amplitude thresholds (e.g., min_height = 1.5*ATR) via config flag.
    2. Monitor trade rate; if flood subsides, revert gradually.

## Observability examples (metrics with frequencies)

- Heartbeat: every 30s
- Data freshness: every 1min
- Order health: every bar iteration
- Equity & exposure snapshot: every bar close and every hour
- Daily summary email/report: at UTC 00:30

## Paper trading and staged rollout

- Paper trading:
    - Use broker sandbox with identical execution semantics if available; otherwise, simulate broker using realistic fills and delays.
    - Run for pre-defined minimum: e.g., 8 weeks or 100+ trades for 4H/1D systems.

    - Evaluate OOS metrics against backtest expectations; log divergences.
- Canary live rollout:
    - Start with low capital (e.g., 1–5% of intended allocation) or reduced size (e.g., 10% lots).
    - Observe fills, slippage, and broker quirks for 1–8 weeks.
    - Gradually increase allocation if metrics remain within acceptable bounds.
- Multi-region concerns:
    - For geographically distributed brokers, consider separate canaries per broker.

### Operational metrics for acceptance during canary:

- Slippage within ±X% of backtest assumption (default X=50% tolerance)
- Fill rates for limit orders >= expected_fill_rate (e.g., 60%)
- Rejection rate <= 2%
- No critical alerts triggered

### Security & access controls

- Least-privilege access to trading keys; separate paper and live keys.
- Multi-factor authentication for sensitive actions (deploy, change risk params, kill-switch).
- Audit trails: who changed what (config, code deploy) and when.
- Regular secrets rotation schedule.

### Compliance, recordkeeping, and audits

- Maintain complete trade audit trail: orders, fills, account snapshots, run_id, git commit, config snapshot.
- Retain records per regulatory requirement (often multi-year); compress/store in immutable backups.
- Periodic review of P&L attribution and reconciliation with broker statements.
- If trading on behalf of others, ensure appropriate licensing and disclosures.

### Incident post-mortem process

For any CRITICAL incident:

1. Triage and immediate mitigation (stop trading, flatten if needed).
2. Capture full logs and trade state.
3. Assemble incident team and produce timeline within 24–48 hours.
4. Root cause analysis and remediation plan with owners and deadlines.
5. Post-mortem report publicly stored and referenced for future releases.

### Maintenance and change control

- Use CI/CD pipelines for code deployments with automated tests.
- Require peer review for changes to strategy logic or risk parameters.

- Maintain change log of parameter updates with rationale.
- Schedule regular revalidation (quarterly) of live strategies: re-run backtest on last N months, review performance drift.

### Capacity planning and scaling

- Estimate expected order throughput and storage needs (trade logs, tick/ohlc storage).
- Monitor CPU/memory for indicator computation and backtester replay jobs.
- Plan for horizontal scaling: separate workers for strategy/exec/recon to limit blast radius.

### Human-in-the-loop decisions

- Critical risk changes should escalate to human operator:
    - Large parameter adjustments
    - Suspension of strategies beyond automated thresholds
    - Broker transfers or manual liquidation decisions
- Provide UI for quick manual overrides with secure authentication.

### Post-deployment monitoring KPIs (track daily/weekly)

- Realized daily P&L vs expected P&L
- Slippage and spread realized mean/stdev
- Order rejection & cancellation rates
- TT (time to detect and resolve incidents)
- Percentage of planned vs actual trades executed

### Runbook excerpt — emergency flatten (if configured)

1. Trigger: margin_imminent OR kill_switch_manual_confirmed
2. Action:
    - Cancel all working orders
    - For each open position:
        - Submit market order to flatten (size equal to current position)
        - Log broker response and fill details
3. Post-action:
    - Verify flat state via broker positions endpoint
    - Notify ops and stakeholders

### Final deployment checklist

Use before flipping to live:

- Backtest & walk-forward validated and documented
- Paper trading completed: min_duration / min_trades met
- Canary plan defined: start_size, monitoring KPIs, increase plan

- Monitoring dashboards & alerts live and tested
- Kill-switches implemented and tested (manual and automated)
- Secret management & access controls in place
- Reconciliation & audit logging verified
- Incident runbooks accessible and owners assigned

## End-of-chapter quick checklist

- CI tests passing and deployable artifact produced
- Sandbox broker adapter tested and reconciled
- Paper trading completed for required duration
- Monitoring & alerts configured and exercised
- Emergency stop and flatten procedures tested
- Canary deployment executed with acceptance criteria

# Common Failures

Chapter summary: overviews the most common operational, modeling, and behavioral failure modes for automated 4H/1D Forex strategies, explains why they occur, introduces detection signals, and provides concrete mitigations (codable where applicable). Each item includes quick checks and config actions to prevent or recover.

Structure - Model & research failures - Data failures - Execution & broker failures - Risk & sizing failures - Operational & deployment failures - Behavioral/organizational failures - Quick detection rules and automated mitigations - Recovery playbook snippets

## A. Model & research failures

1. Overfitting / Data snooping

- Why: excessive parameter tuning on historical data, selecting parameters that exploit noise.
- Detection signals:
  - Large performance gap between IS and OOS.
  - Extreme sensitivity to small parameter changes.
  - High number of parameters relative to trades.
- Mitigations:
  - Use walk-forward, purged k-fold CV, nested CV.
  - Limit free parameters; prefer coarser parameter grids.
  - Require minimum N trades in OOS (rule: N >= 100 or combine multiple instruments).
  - Apply bootstrap p-values and report confidence intervals.
- Codable checks:
  - compute_oos_degradation = (IS_sharpe - OOS_sharpe) / IS_sharpe; if > 0.4 flag for review.

1. Survivorship / Sample selection bias

- Why: selecting instruments or periods that remove poor performers or using truncated datasets.
- Detection:
  - Sudden missing instruments in live vs backtest.
- Mitigation:
  - Use full history, document instrument selection criteria, apply out-of-time holdouts.

1. Lookahead leakage

- Why: using future information by mistake (shift errors, improper rolling windows).

- Detection:
    - Strategy signals referencing indicator values that change after entry when recomputed.
- Mitigation:
    - Enforce indicator code tests: for each index i, ensure signal(i) depends only on df[:i+1].
    - Use unit tests that assert no future-slice is used.

1. Ignoring nonstationarity / regime shifts

- Why: market behavior changes, rendering past patterns less predictive.
- Detection:
    - Rolling performance degradation, changes in trade distribution or volatility.
- Mitigation:
    - Implement regime detection (ATR quantiles, ADX/trend vs range detectors).
    - Use adaptive parameters (re-tune on rolling windows) but validate via walk-forward.
- Codable example:
    - If ATR_now > ATR_median * 3 or < ATR_median * 0.5, pause trading or tighten thresholds.

## B. Data failures

1. Missing or duplicated bars

- Why: vendor glitches, timezone mismatches.
- Detection:
    - Non-monotonic timestamps, duplicated indices, gaps longer than expected.
- Mitigation:
    - Data ingestion validation: enforce monotonic timestamps, detect gaps and either fill or mark as invalid.
- Codable check:
    - if any(diff(timestamps) != expected_interval): alert + pause/reconcile.

1. Bad ticks / extreme spikes

- Why: feed errors, outliers.
- Detection:
    - Single-bar moves > X*ATR or > Y% change.
- Mitigation:
    - Outlier filtering: cap per-bar moves or discard bars if flagged; prefer using tick-level aggregation and volume checks.

- Codable rule:
    - if bar_range > 10*ATR: mark bar as suspect, exclude from indicator updates, log.

1. Timezone and DST errors

- Why: mixing local broker time and UTC.
- Detection:
    - Candle alignment mismatch vs broker statements.
- Mitigation:
    - Standardize on UTC; convert broker times on ingest; unit tests for timezone offsets.

## C. Execution & broker failures

1. Unrealistic fill assumptions

- Why: assuming midprice fills or ignoring spread and slippage.
- Detection:
    - Backtest fills consistently better than paper/live fills.
- Mitigation:
    - Model spread, slippage, partial fills; calibrate models using paper/canary fills.
- Codable monitor:
    - track_slippage = realized_fill_price - expected_entry_price; alert if mean_slippage > threshold.

1. Order rejections and stalled orders

- Why: broker limits, invalid params, connectivity.
- Detection:
    - Spike in rejection count, high TTL expirations.
- Mitigation:
    - Pre-check order size/margin, rate limits, and validate responses; implement retry/backoff and alerting.
- Codable rule:
    - if rejection_rate_last_N > 0.02: pause new orders and notify.

1. Partial fills and liquidity mismatch

- Why: large notional vs market depth, broker limitations.
- Detection:
    - Repeated partial fills, large unfilled quantities.
- Mitigation:
    - Limit max notional per order, implement slicing/vwap-like execution for large fills.

- Codable: compute_fill_ratio = filled_qty / requested_qty; if < min_fill_ratio, flag.

1. Broker-side rule differences (stop behavior, swap conventions)

- Why: stop loss types, slippage at broker, swap roll 3-day on Wed.
- Detection:
  - Discrepancies between expected and broker behavior in sandbox.
- Mitigation:
  - Read broker docs, test in sandbox, make adapter to map local orders to broker specifics.

## D. Risk & sizing failures

1. Margin exhaustion and unexpected leverage effects

- Why: ignoring margin implications across correlated positions.
- Detection:
  - Rapidly increasing margin_used, margin call warnings.
- Mitigation:
  - Compute portfolio margin usage before order acceptance; enforce margin buffers (e.g., available_margin >= required_margin * 1.2).
- Codable rule:
  - if projected_margin_post_trade > free_margin / margin_buffer: reject order.

1. Position concentration / correlation risk

- Why: multiple pairs highly correlated produce effective overexposure.
- Detection:
  - High realized correlation across open positions; aggregated directional exposure.
- Mitigation:
  - Limit net directional exposure per currency; compute currency-level exposure and cap it.
- Codable example:
  - compute_currency_exposure(); if any currency_abs_exposure > exposure_limit -> reject new trade.

1. Size creep / drift in lot sizing

- Why: compounding position sizing increases volatility beyond expectation.
- Detection:
  - Increasing average position sizes vs plan.
- Mitigation:
  - Enforce fixed max position size and max growth per period; cap lot size to percent of equity.

## E. Operational & deployment failures

1. Missing monitoring, late alerts

- Why: alerts misconfigured or suppressed.
- Detection:
    - Incidents discovered late by humans or external parties.
- Mitigation:
    - Test alerting channels, use synthetic checks, require alert ack workflow.
- Codable test:
    - schedule_synthetic_alert_test() weekly; require ack.

1. Inadequate testing before deploy

- Why: skipping paper testing or insufficient test coverage.
- Detection:
    - Unexpected behavior on live canary, failing reconciliation.
- Mitigation:
    - Mandatory paper-trading phase with documented acceptance criteria and automated CI gating.

1. Insufficient change control

- Why: ad-hoc parameter changes without review.
- Detection:
    - Sudden performance changes correlated to config change times.
- Mitigation:
    - Require PR review, record parameter change logs, require rollback plan.

## F. Behavioral/organizational failures

1. Manual overrides and cookbook bias

- Why: human overrides that invalidate audit trail.
- Detection:
    - Discrepancies between strategy intent and executed trades with manual notes.
- Mitigation:
    - Record manual actions; require two-person approval for non-trivial overrides; maintain immutable audit log.

1. Over-trading after small wins / revenge trading

- Why: human behavioral response to short-term outcomes.
- Detection:
    - Increased trade frequency after drawdowns or large wins.

- Mitigation:
    - Enforce trading limits and automated cooldown timers after X consecutive trades or losses.

## G. Quick detection rules (to implement as automated health checks)

- Data freshness: if now - last_bar_time > allowed_lag -> raise CRITICAL.
- High rejection rate: if rejections_last_100 / 100 > 0.02 -> WARNING.
- Slippage spike: if mean_slippage_24h > baseline_slippage * 2 -> WARNING.
- Equity drawdown: if drawdown_from_peak >= auto_suspend_threshold -> CRITICAL -> auto-suspend.
- Consecutive losses: if consecutive_losses >= N_stop -> SUSPEND.
- Margin buffer: if free_margin / required_margin < margin_buffer_limit -> ALERT.

## H. Automated mitigations (codable actions)

- Auto-suspend trading: set system state to SUSPENDED, cancel working orders, notify ops.
- Throttle sizing: reduce max allowed lot by factor f (e.g., 0.5) when slippage spikes.
- Increase pattern thresholds: raise min_height_multiplier from 1.2->1.6 when false breakout rate high.
- Switch to conservative execution: convert limit entries to passive-only or increase ttl.
- Temporarily disable strategies per instrument when instrument-specific anomalies detected.

## I. Recovery playbook snippets (short)

1. Data outage recovery

- Action: pause new order submission; replay last known good data; reconcile with broker; resume once data stable.

1. Execution degradation (slippage/rejections)

- Action: throttle sizes, pause auto-increases, run diagnostics, escalate to ops.

1. Unexpected drawdown

- Action: enter REVIEW_MODE (pause new trades), run quick performance compare (last N trades vs expected), if drawdown triggers breach then SUSPEND live trading and alert.

1. Margin call near

- Action: initiate emergency flatten per runbook if margin_imminent flag true; else reduce leverage and new orders.

### J. Root-cause investigation checklist

- Collect: full logs, order/trade ledger, market data for window, system metrics (latency, CPU), recent config changes, git commit id.
- Reproduce: run backtest on exact timeframe with same execution parameters; attempt to reproduce fills.
- Patch: implement fix in staging, run regression tests, then deploy with canary.

### K. Example codable health-check rules (Python-like)

```
def health_checks(state):
    if now - state.last_bar_ts > timedelta(minutes=10):
        alert('DATA_STALE')
    if state.rejection_rate_100 > 0.02:
        alert('HIGH_REJECTION')
    if state.mean_slippage_24h > 2 * state.expected_slippage:
        throttle_sizes(0.5); alert('SLIPPAGE_SPIKE')
    if state.drawdown_pct >= auto_suspend_threshold:
        suspend_trading('DRAWDOWN_LIMIT')
```

### L. Post-incident improvements (must track)

- Update tests to cover the root cause.
- Add new monitoring metrics if missing.
- Update runbooks and assign owners.
- Schedule retrospective with stakeholders.

### End-of-chapter checklist

Use before resuming live after an incident: - Incident fully documented with timeline and root cause - Tests updated and CI passing - Monitoring/alerts added or tuned - Runbook updated; owners assigned - Canary/replay run completed and validated

# Regulatory and Brokerage Considerations

## Purpose

Provides actionable guidance for selecting brokers, understanding regulatory regimes, structuring accounts, and building trading systems that meet legal, operational, and compliance constraints for automated Forex trading (4H / 1D, ~10-day horizon).

## Jurisdiction and regulatory regimes (high-level checklist)

- **Identify your legal domicile and counterparty jurisdiction.** Broker conduct and account protections depend on both.
- **Major regulatory bodies to know:**
  - USA: CFTC (retail forex rules), NFA (member brokers), SEC (rarely FX).
  - UK: FCA.
  - EU: ESMA (and national regulators like BaFin, AMF); MiFID II rules apply.
  - Australia: ASIC.
  - Japan: FSA.
  - Offshore: Belize, Seychelles — lighter supervision; higher counterparty risk.
- **Key regulatory features to verify:**
  - Broker licensing and regulator contact.
  - Segregation of client funds from broker operating funds.
  - Default resolution and investor compensation schemes (if any).
  - Leverage limits, margin close-out rules, and negative balance protection.
  - Reporting and recordkeeping obligations.

## Broker selection criteria (operational checklist)

- **Execution quality**
  - Average spread and spread variability on target pairs (4H/1D need tight stable spreads).
  - Re-quotes, slippage statistics, and fill rates (request historical fill data or demo testing).
  - ECN/STP vs market-maker: ECN often better for transparency; market-makers may internalize orders.
- **Pricing model & connectivity**
  - Streaming raw market data (tick-level) vs delayed aggregated candles.
  - Access methods: REST, FIX, WebSocket, MT5/MT4 bridge, proprietary API.
  - Order types supported (IOC, FOK, limit, market, stop, OCO).
- **Costs**
  - Commissions per lot, mark-up on spread, swap/rollover schedule, inactivity fees, withdrawal fees.
  - Minimum ticket size, lot step, and margin requirements per pair.

- **Account types**
  - Segregated vs pooled liquidity, institutional vs retail, demo vs live, subaccounts.
  - Multi-currency base accounts and conversion costs.
- **Operational robustness**
  - API rate limits, session timeouts, connection failover, guaranteed SL/TP support.
  - Historical data access for backtesting (tick or 1m and OHLCV).
- **Reputation & audits**
  - Third-party audits, community feedback, regulatory disciplinary history.
- **Collateral and custody**
  - How client funds are held, interest on balances, and rehypothecation terms.

## Onboarding, KYC/AML, and tax considerations

- **KYC/AML**
  - Expect identity verification, proof of residence, and source-of-funds documentation for live accounts.
  - Corporate accounts require formation documents, beneficial owner disclosures, and possibly FATCA/CRS forms.
- **Tax reporting**
  - Understand local tax treatment of Forex gains (capital gains vs ordinary income; mark-to-market in some jurisdictions).
  - Brokers may provide 1099-B (US) or year-end statements; ensure recordkeeping for trade-level P&L.
- **Corporate structuring**
  - Consider trading via LLC or corporate entity for liability, but validate tax and regulatory implications with an accountant.

## Account types and margin/leverage specifics

- **Leverage limits**
  - Retail limits vary (e.g., ESMA caps, US regulatory caps). Use conservative leverage in deployment.
- **Margin calculation models**
  - Cross-margin vs isolated margin; understand how positions across pairs affect margin.
- **Margin calls and liquidation**
  - Broker margin call thresholds, speed of liquidation, and priority of positions during enforced closeouts.
- **Hedging**
  - Some jurisdictions/brokers restrict simultaneous opposite positions per symbol (hedging). Confirm behavior for multi-leg strategies.

## Transaction cost modeling for broker selection

- **Spread components**
  - Benchmark mid-price vs broker bid/ask. Measure realized spread cost = executed price − mid-price at fill time.
- **Commissions & minimums**
  - Include per-lot commission and any per-ticket minimums; convert to % of notional for uniform comparison.
- **Swap/rollover**
  - Overnight financing costs: examine long/short asymmetry and how 10-day horizons aggregate swap impact.
- **Slippage**
  - Measure empirical slippage on live or demo fills; factor worst-case in backtests.
- **Liquidity impact**
  - For larger account sizes, estimate slippage as function of order size and market depth; request depth-of-book or run execution cost tests.

## API, data, and operational requirements for automated systems

- **API reliability**
  - Uptime SLA, retry semantics, idempotency support for order calls, and order state webhooks.
- **Data fidelity**
  - Prefer broker tick feeds or DMA tick dumps for realistic intrabar simulation; verify timezone (UTC recommended).
- **Order semantics**
  - Confirm how broker treats order timestamps, partial fills, and reported fill price (average vs last).
- **Session management**
  - Reconnection strategies, socket heartbeats, and session tokens refresh.
- **Throttling and limits**
  - Respect rate limits; implement local queuing, backoff, and batching to avoid rejections.
- **Failover & redundancy**
  - Plan for multi-broker setup or broker+backup read-only data feed to avoid single point of failure.

## Legal agreements and API terms

- **Master agreements**
  - Read terms: order routing, algorithmic trading policies, market data licensing, and liability clauses.
- **API usage**
  - Some brokers prohibit certain automated behaviors (e.g., latency arbitrage) — request written confirmation for your strategy class.

- **Indemnities & limits**
  - Look for clauses on system outages, erroneous fills, and broker claims; negotiate SLAs if institutional.

## Risk, compliance, and operational controls to implement

- **Pre-trade controls**
  - Maximum order size, per-instrument exposure limits, position limits, and daily loss caps enforced client-side and on gateway.
- **Post-trade reconciliation**
  - Daily reconciliation of executed trades vs internal logs; P&L and position audits.
- **Monitoring & alerts**
  - Fill anomaly alerts (fill quality, partial fills), latency spikes, rejected orders, and margin threshold breaches.
- **Kill switches**
  - Manual and automated kill switches: time-based (market close), drawdown-triggered, and connectivity-triggered.
- **Permissions & access**
  - Use least-privilege API keys (read vs trade), rotate credentials, and log all operator actions.
- **Back-office**
  - Keep immutable trade logs, ledger of fees, and audit-ready records for regulator queries.

## Best practices for testing and transition to live

- **Demo/Simulation**
  - Start with demo accounts from chosen broker; run parallel paper trading to match live connectivity and market microstructure.
- **Staged rollout**
  - Small live capital, monitored for performance and execution differences; ramp up with predefined criteria.
- **Live equivalence checks**
  - Compare slippage, fills, and realized P&L vs backtest and demo; adjust cost models and position sizing.
- **Record consent**
  - Maintain documented checklist of acceptance criteria for moving from paper to live (P&L alignment, error rates, API stability).

## Specific issues for 4H/1D, ~10-day horizon strategies

- **Rollover exposure**
  - Multi-day holds accumulate swaps; quantify per-pair and include in cost model.
- **Spread widenings**
  - Overnight and news-event spread spikes affect 1D strategies; ensure spread stress tests.

- **Liquidity**
  - Major pairs are very liquid; exotic pairs can have thin weekend gaps and larger spreads—avoid or size down.
- **Margin across correlated positions**
  - Positions in correlated majors can increase margin use; use portfolio margin logic or isolate risk.

## Institutional considerations (if scaling)

- **Prime brokerage / liquidity providers**
  - For larger AUM, consider introducing prime brokers, direct liquidity pools, or aggregated ECN access.
- **Custody and settlement**
  - Institutional clients may require segregated custodians and audited reporting.
- **Regulatory reporting**
  - Large trading operations may face transaction reporting, best execution obligations, and periodic audits.

## Red flags when choosing a broker

- Lack of clear regulatory registration.
- Refusal to provide historical fill and spread data.
- Hidden fees, opaque commission structures, or inconsistent pricing.
- Frequent outages during high-volatility periods.
- No segregation of client funds or negative reviews about withdrawals.

## Minimal compliance & operational checklist (one-page)

- Verify regulator license & client fund segregation.
- Confirm API order semantics, rate limits, and SLAs.
- Collect swap and commission schedules; compute per-trade cost.
- Run 30-day demo with identical execution code; log fills and compute slippage distribution.
- Implement pre-trade hard limits and kill switches.
- Maintain KYC/tax docs; consult tax advisor for reporting.
- Establish daily reconciliation and weekly review of execution quality.

## Example broker comparison table (attributes to populate)

| Attribute | Broker A | Broker B | Broker C |
|---|---|---|---|
| Regulator | | | |
| Typical spread EUR/USD (pips) | | | |
| Commission per lot | | | |
| API type | | | |
| Tick data availability | | | |

| Attribute | Broker A | Broker B | Broker C |
|---|---|---|---|
| Swap long/short | | | |
| Margin closeout thresh. | | | |
| Max leverage (retail) | | | |
| Demo support | | | |
| Withdrawal time | | | |

Fill this with measured values during broker evaluation.

## Practical templates and code snippets (pseudocode)

- API key handling: store secrets in encrypted vault; limit key scopes.
- Reconnect loop outline:

```
while True:
  try:
    connect_stream()
    subscribe_ticks()
    heartbeat_monitor()
  except TransientError:
    backoff_and_retry()
  except CriticalError:
    trigger_kill_switch()
    alert_ops()
    break
```

- Pre-trade limit check:

```
if order_notional > max_notional_per_pair:
  reject_order()
if projected_margin_after_entry > margin_cap:
  reject_order()
```

## 16. Final recommendations

- Prefer regulated, transparent brokers with tick-level data and programmatic access.
- Build conservative transaction-cost and margin assumptions into backtests.
- Start small, require live equivalence before scaling, and maintain rigorous operational controls and logs.
- Consult local counsel/accountant for tax and corporate structuring decisions.

# Swing Trading Plan

This chapter provides a complete, codable 10-day (≈10 calendar days; typical holding 1–10 days) swing-trading plan for Forex on 4H/1D timeframes. Includes entry/exit rules, sizing, execution realism, backtest settings, monitoring, and checklist for live rollout. Assumes Python implementation with ATR(14) defaults, UTC candles, 0.5% default risk per trade.

## Overview

- Strategy class: Trend-following with mean-reversion control and volatility-adaptive stops.
- Instruments: Majors only (EUR/USD, GBP/USD, USD/JPY, USD/CHF, AUD/USD). Avoid exotics.
- Timeframes: Primary 4H signals, 1D confirmation for directional bias.
- Horizon: target 3–10 days per trade, average ~5 days.
- Expected trade frequency: ~5–50 trades/year per pair (depends on filters); minimum trades/year across portfolio ≈50 for statistical confidence.

## Ruleset

### 1) Data and preprocessing

- Use UTC OHLCV candles; resample tick/1m to 4H and 1D as:
  - 4H: aligned to 00:00 UTC (00:00, 04:00, …).
  - 1D: 00:00 UTC daily candle.
- Compute ATR14 on 4H series (convert to pips or price units).
- Maintain bid/ask mid as mid = (bid+ask)/2 for signal calc; model spreads on execution.

### 2) Signal generation (entry)

- Trend filter (1D):
  - Compute 20-period EMA on 1D close. Direction = LONG if close_1D > EMA20_1D, SHORT if close_1D < EMA20_1D.
- Entry trigger (4H):
  - Momentum breakout: On 4H close, if Direction == LONG and close_4H breaks above the 20-period high on 4H (i.e., close_4H > max(high[-20:-1])), signal LONG.
  - Conversely for SHORT: close_4H < min(low[-20:-1]).
- Mean-reversion safety filter:
  - Require 4H ATR14 < ATR_threshold (e.g., ATR14 < 1.5 * rolling median ATR(90)) to avoid extreme volatility.
- Trade frequency filter:
  - Max one new trade per pair per 72 hours.

    - Max simultaneous pairs exposure: 5.

**3) Entries (execution)**

- Order type: Limit entry placed at signal price (close_4H) minus (for LONG) entry_buffer = 0.25 * ATR14; for SHORT add 0.25 * ATR14.
- Fallback: If limit not filled within next 2 4H candles, issue market order at next candle open.
- Apply slippage model in backtest: fill_price = executed_price ± Normal(0, 0.1 * ATR) clipped to realistic bounds; add spread cost using broker spread schedule.

**4) Stops and targets**

- Stop loss:
    - ATR-based hard stop: SL = entry_price − 2.0 * ATR14 for LONG (reverse sign for SHORT).
    - Soft trailing stop: after price moves favorably by 1.5 * ATR, move stop to entry + 0.75 * ATR (lock partial profit).
- Profit target & scaling:
    - Primary target at 3.0 * ATR (take-profit level).
    - If price reaches 3.0*ATR, close 50% size and move remaining position stop to breakeven + 0.2*ATR; let remainder trail with ATR-based trailing (trail step = 0.5*ATR).
- Time-based exit:
    - If position age >= 10 calendar days, exit fully at next 4H close regardless.
- Adverse exit:
    - If daily swap accumulates above cost_cap (configurable, e.g., 0.1% of position notional), close or avoid new trade.

**5) Sizing & risk controls**

- Risk per trade: default 0.5% of account equity (configurable).
- Lot sizing:
    - Notional_size = (equity * risk_per_trade) / (SL_pips * pip_value)
    - Convert notional to lots respecting broker min lot and lot step.
- Max position size per pair: 3% of equity.
- Portfolio-level max exposure: 20% of equity across all FX notionals (notional defined in base currency).
- Kelly sanity: compute Kelly fraction from historical returns; cap applied_size = min(calculated_size, 0.5 * Kelly, max_position_size).

**6) Execution realism & transaction costs**

- Spread: use pair-specific median spread (measured from broker or use default EUR/USD = 0.6 pip) added to execution cost.
- Slippage: modeled as Normal(0, 0.1*ATR) per trade in backtests; stress test ±50%.

- Swap/Rollover: apply broker swap schedule daily; include in P&L and time-based exit checks.
- Commission: include per-lot commission where applicable.
- Partial fills: model small probability of partial fills for large orders; backtests should simulate fill delays and partial volume.

**7) Backtesting configuration (recommended)**

- Historical window: at least 5 years if available; include multiple regimes.
- Walk-forward: rolling 24-month train → 3-month test windows.
- Monte Carlo: 1,000 resamples varying spread ±30% and slippage ±30%.
- Transaction cost sensitivity: runs with baseline, stressed (+50% costs), and optimistic (-25%).
- Purging: apply purged-combinatorial cross-validation for parameter tuning.
- Metrics to record: per Chapter 20 core metrics plus MAE/MFE, trade-level logs.

**8) Monitoring and alerts (live)** - Required real-time checks: - Fill-quality alert: slippage > 2*expected or partial fill >50%. - Equity drawdown alert: strategy drawdown > 5% (auto-paper escalate) and >10% triggers kill switch. - Connectivity/heartbeat: if data feed/API reconnects >3 times within 30 minutes, pause new entries. - Swap accumulation alert for open trades > cost_cap. - Daily reconciliation: - End-of-day P&L vs broker statement; reconcile fills and position sizes. - Reporting: - Daily summary: open positions, equity, realized P&L, intraday alerts. - Weekly review: trade logs, updated parameter sensitivity heatmap.

**9) Live rollout plan (phased)**

- Stage 0: Unit tests for all signal, sizing, and risk modules.
- Stage 1: Backtest + walk-forward validation with cost-stressed scenarios.
- Stage 2: Paper trading for 60 calendar days with identical API and live spreads; require P&L within ±20% of backtest expectation after accounting for costs.
- Stage 3: Small live capital (e.g., 1–5% of target deployable AUM) with tight monitoring for 30 days. Only scale if fill quality and P&L match.
- Stage 4: Gradual scale-up with dynamic capacity checks and re-simulated slippage as AUM grows.

**10) Failure modes & contingency actions**

- Case: Unexpected spread widening → Action: Pause new entries; widen SL/TP in config or switch to market-only exit.
- Case: API outage → Action: Market-close remaining positions at next available safe fill or trigger pre-set stop orders if supported.
- Case: Consecutive losing trades exceed N (e.g., 6) → Action: Auto pause strategy and require manual review.
- Case: Swap costs materially exceed forecast → Action: Halt trades in affected pairs and re-run backtest with updated swap.

**11) Example parameter set (defaults)**

- ATR period: 14 (4H)
- EMA20 period (1D): 20
- Breakout lookback: 20 (4H)
- Entry_buffer: 0.25 * ATR
- SL: 2.0 * ATR
- TP: 3.0 * ATR (partial take at TP)
- Risk per trade: 0.5% equity
- Max exposure per pair: 3% equity
- Max portfolio exposure: 20% equity
- Limit fill window: 2 * 4H candles
- Max trades per pair per 72 hours
- Time exit: 10 days

**12) Example pseudocode (concise)**

```
for each new 4H candle:
  update_indicators()
  if no_open_trade(pair) and trend1D_ok(pair):
    if breakout_signal(pair):
      price = close_4H
      place_limit(entry_price = price - 0.25*ATR)
      wait up to 2 candles
      if not filled: market_enter()
      compute_SL = entry - 2*ATR
      size = calc_size(equity, risk=0.5%, SL)
      place_stop(SL)
      place_TP(3*ATR)
on price_move:
  if price >= entry + 1.5*ATR and not_trailing:
    move_stop(entry + 0.75*ATR)
  if price >= entry + 3*ATR:
    close(50%)
    move_stop(breakeven + 0.2*ATR)
on each day:
  apply_swap_costs()
  if position_age >=10 days: exit_next_4H_close()
monitoring_loop()
```

**13) Backtest diagnostics to produce**

- Equity curve + drawdown bands.
- MAE/MFE per trade histogram.
- Slippage sensitivity chart.
- Trade frequency by pair and by calendar month (seasonality).

- Rolling Sharpe (90/180-day) and rolling average hold time.
- Execution cost waterfall (spread, commission, slippage, swap).

**14) Acceptance criteria before scaling capital**

- Paper/live equivalence: realized slippage and spread within ±25% of backtest assumptions.
- Out-of-sample Sharpe >= 0.8 and profit factor >=1.5 over 3-month live-test.
- Drawdown behavior consistent with backtest; tail losses within modeled percentiles.
- Operational maturity: automated kills, alerts, daily reconciliation in place.

**15) Quick checklist for traders (one page)**

- Confirm data alignment to UTC and broker tick parity.
- Populate pair-specific spread and swap schedule.
- Run walk-forward + Monte Carlo backtests.
- Implement pre-trade hard risk checks (size, margin).
- Paper trade 60 days; verify P&L and fill quality.
- Start live small; monitor closely; scale by pre-defined rules.

## Closing note

This plan is prescriptive and conservative: adapt parameters to your risk tolerance and instrument liquidity, but always re-run full backtests and live-paper validation after changes. Use Chapter 20 and Chapter 25 checklists during rollout for metrics and broker compliance.

# Advanced Topics

## Purpose

Provides practical, cautionary, and codable guidance on advanced techniques relevant to automated Forex trading (4H / 1D, ~10-day horizon): machine learning caveats, feature engineering, statistical methods (co-integration/pairs), portfolio optimization, and risk-aware model deployment. Emphasizes reproducibility, out-of-sample rigor, and execution realism.

## Machine learning — pragmatic stance and caveats

- Use ML only where it clearly adds value over rule-based logic (feature interactions, non-linear patterns). For 4H/1D FX, sparse signal frequency and regime shifts often limit ML gains.
- Primary ML pitfalls:
  - Data snooping / overfitting: high-dimensional feature spaces + low trade counts → spurious patterns.
  - Non-stationarity: market regime changes invalidate learned models quickly.
  - Labeling leakage: using future information (e.g., post-close realized return beyond allowed horizon) in training labels.
  - Execution mismatch: models trained on mid-prices but executed on bid/ask without modeling spreads/slippage.
- Recommendations:
  - Prefer simple models (regularized logistic regression, gradient-boosted trees with shallow depth) over deep nets for low-frequency signals.
  - Limit features to economically interpretable transforms and apply feature selection with purged time-series CV.
  - Always include transaction-cost-aware loss (penalize small predicted returns that would be eaten by costs).
  - Report model stability across time: retrain cadence, rolling performance, and feature importance drift.

## Supervised learning — labeling and loss design

- Label design (10-day swing target):
  - Define target return $R_T$ = (price at exit – entry) / entry over fixed holding window OR until TP/SL hit.
  - Use thresholded labels: +1 if $R_T > \theta_{up}$, −1 if $R_T < -\theta_{down}$, 0 otherwise — ensures focus on economically meaningful moves.
- Loss functions:
  - Use cost-sensitive losses that incorporate expected transaction cost per prediction: e.g., weighted cross-entropy where weight = expected net return after costs.

    - Consider ranking losses (pairwise ranking) when the goal is selecting top-k candidates.
- Avoid regression to predict raw returns unless large sample size exists; classification or ordinal ranking tends to be more robust.

## Feature engineering — best practices

- Temporal features:
    - Lagged returns (multi-scale: 4H, 1D, 5D), volatility (ATR14), realized vol (rolling std), momentum (EMA differences), and seasonality (hour/day-of-week).
- Price-structure features:
    - Recent highs/lows, breakout distances (price − rolling max/min), MAE/MFE estimates from recent trades.
- Volume/proxy liquidity:
    - When true volume unavailable, use tick-count, spread, or quote updates as liquidity proxies.
- Macro & events:
    - Interest rate differentials, scheduled economic calendar flags (major news within ±24h), and funding-roll dates.
- Normalization & stability:
    - Standardize features using rolling mean/std on training windows; avoid global normalization that leaks future stats.
    - Use winsorization to limit outliers; log transforms for skewed variables.
- Interaction features:
    - Cross-pair correlations, carry vs momentum interactions, distance-to-stop normalized by ATR.
- Dimensionality reduction:
    - Prefer targeted selection (domain knowledge) over blind PCA. If using PCA/autoencoders, validate that components survive across regimes.

## Validation methodologies for time-series ML

- Never use i.i.d. CV. Use time-series aware methods:
    - Purged K-fold CV and combinatorial purged CV for hyperparameter selection.
    - Rolling/expanding-window walk-forward (train on past window, test on immediate future).
    - Avoid overlap between train and val that would include label-horizon overlap (purge buffer >= max holding period).
- Performance evaluation:
    - Use economically adjusted metrics: profit factor after costs, return per trade, and drawdown behavior, not only AUC.
    - Bootstrap or Monte Carlo resampling of trade sequences to estimate uncertainty.

- Backtest with simulated execution (spread, slippage, partial fills). If model predicts probability p, translate to execution policy (top-k, threshold, or probabilistic execution) and simulate resulting trades.

## Model risk controls and deployment practices

- Pre-deployment:
  - Stability tests: retrain model on rolling windows and measure feature importance and performance variance.
  - Adversarial checks: stress test with increased spreads, latency, and sudden volatility spikes.
- Live controls:
  - Probability calibration monitoring: check predicted vs realized hit rates; trigger retrain or rollback on drift.
  - Shadow mode: run model predictions in parallel with paper account and compare trade decisions and P&L.
  - Kill criteria: consecutive losses, model confidence collapse, or estimation of transaction-cost exceedance.
- Governance:
  - Model registry with versioning, training data snapshot, hyperparameters, and performance metrics.
  - Retrain cadence documented and automated with validation gates before deployment.

## Statistical arbitrage and co-integration/pairs trading

- Applicability:
  - Pairs/co-integration methods can work for FX when pairs share common drivers (e.g., EUR/USD vs EUR/GBP) or synthetic constructions.
  - Use for mean-reversion trades with short-to-medium horizons; 10-day horizon can suit co-integration if mean-reversion speed is sufficient.
- Testing for co-integration:
  - Use Engle-Granger two-step or Johansen test on log-price spreads; prefer Johansen for multi-series.
  - Ensure stationarity of spread (ADF test) and estimate half-life of mean reversion: half-life = $-\ln(2)/\ln(\varphi)$ from AR(1) fit.
  - Require half-life << desired holding period (e.g., half-life ≤ 5 days for 10-day horizon) for mean reversion to materialize.
- Execution and risk:
  - Hedge ratio estimation: use OLS on historic window but de-mean and re-estimate frequently; consider total-least-squares if both series noisy.
  - Use dynamic hedge ratios (Kalman filter) to adapt to structural shifts.
  - Apply stop-loss and maximum holding times; include funding and spread costs for both legs.

- Portfolio-level considerations:
    - Diversify pairs to reduce idiosyncratic risk; apply correlation constraints to avoid concentration.
    - Monitor residuals for regime shifts; halt trading when spread stationarity fails.

## Portfolio construction and optimization

- Objectives:
    - Maximize risk-adjusted return subject to capacity, turnover, and transaction costs.
- Risk measures:
    - Use volatility targeting, CVaR constraints, and drawdown-based objectives (e.g., minimize conditional drawdown).
- Optimization frameworks:
    - Mean-variance with transaction-cost-aware returns (adjust expected returns by expected costs).
    - Risk-parity or volatility parity for simpler, robust allocations when expected returns uncertain.
    - Hierarchical Risk Parity (HRP) for correlated FX signals to avoid inversion issues in covariance matrices.
- Practical constraints:
    - Enforce limits: per-pair max notional, margin caps, and maximum portfolio leverage.
    - Include turnover penalty in objective to reduce whipsaw trading (L1 or L2 penalty on changes in weights).
- Estimation stability:
    - Use shrinkage estimators for covariance (Ledoit-Wolf) or factor models to stabilize inverse covariance calculations.
    - Rebalance cadence: align rebalancing frequency with strategy horizon (e.g., weekly or bi-weekly for 10-day horizon).
- Backtest with rebalancing costs:
    - Simulate execution impact of rebalancing trades (spread, slippage, margin changes).

## Advanced signal combination & ensemble methods

- Combine heterogeneous signals (momentum, carry, mean-reversion, ML probabilities) via:
    - Rule-based stacking: hierarchy of rules with vetoes (e.g., no new longs when carry negative).
    - Score normalization: convert raw signals to z-scores using rolling stats, then weighted sum.
    - Meta-model: train simple model on historical signal mixes to predict trade outcome; validate with purged CV.

- Weighting:
    - Use utility-based weights that penalize signals with high turnover or low capacity.
    - Regularly recalibrate weights using walk-forward returns and cost-adjusted performance.

### Transaction-cost-aware modeling

- Always simulate tick- or 1m-level fills where possible to capture intrabar slippage and spread evolution.
- Model price impact for scaling:
    - Simple linear model: slippage% = α + β*(order_size / avg_daily_volume).
    - Use empirically estimated coefficients per pair; for FX, use notional vs market depth measures.
- Optimize execution:
    - Use limit orders and TWAP/VWAP-like slicing for large orders; simulate fill probability and time-to-fill trade-offs.
    - Implement smart order routing if multiple brokers/liquidity providers available.

### Backtesting pitfalls unique to advanced methods

- Look-ahead via feature leakage: e.g., features using realized volatility computed over future window inadvertently.
- Survivorship bias in currency pair datasets (less common than equities but can occur with synthetic or broker-provided pairs).
- Mis-specified accounting of swaps, fees, and margin across legs.
- Over-optimization of hyperparameters without purged CV leads to unstable live performance.

### Explainability and interpretability

- For decision-making and regulatory/audit requests, produce:
    - Feature importance (SHAP or permutation) with time stability plots.
    - Trade-level explanations showing which features drove the decision and expected net edge after costs.
- Prefer simpler, interpretable models when possible. Keep documentation of feature construction and training data.

### Co-existence of ML and rule-based systems (hybrid approaches)

- Use ML as a signal generator, but apply deterministic rule-based filters for execution and risk (e.g., do not open trades during news).
- Ensemble safety: require consensus across multiple weak models or apply model-based confidence thresholds before execution.
- Fallback: if model confidence below threshold or data missing, revert to conservative rule-based strategy or pause entries.

### Stress testing and scenario analysis

- Conduct scenario sims including:
    - Spike in spreads (x5), severe slippage, and liquidity evaporation.
    - Rapid regime shift: retrain model on pre-shock data and test on post-shock behavior.
    - Correlation breakdown across pairs leading to concentrated loss.
- Run reverse stress tests to find inputs that produce unacceptable drawdowns and document mitigations.

### Tools, libraries, and compute considerations

- Use reproducible pipelines: data versioning (DVC), environment (conda/pipenv), and model registries.
- Example libraries:
    - pandas, numpy, scipy — data handling
    - scikit-learn, xgboost/lightgbm — ML models
    - statsmodels — time-series tests, cointegration
    - arch — volatility models
    - cvxpy — portfolio optimization
    - backtesting/backtrader/zipline/empyrical/pyfolio — backtest and analytics (ensure they support proper time-series CV)
- Compute:
    - Keep training pipelines lightweight; for frequent retrain use incremental models or scheduled retrain during low market hours.

### Documentation, reproducibility, and auditability

- For each model/strategy maintain:
    - Data provenance: sources, timestamps, resampling rules.
    - Training artifacts: dataset snapshots, code, random seeds, hyperparameters.
    - Backtest reports: metrics, walk-forward results, Monte Carlo sensitivity.
    - Deployment manifest: API keys used (vaulted), runtime environment, versioned binaries.
- Automated daily and weekly reports for monitoring distributional drift and performance.

### Research checklist before adopting an advanced technique

- Does it improve net P&L after realistic transaction costs and slippage? If not, discard.
- Is the improvement robust across regimes and significant on out-of-sample tests?
- Does the approach scale to required AUM without unacceptable price impact?
- Are model risks documented and mitigations (kill-switches, fallback strategies) implemented?

## Example advanced recipes (concise)

- Dynamic volatility-targeted momentum:
  - Scale position size by target_vol / realized_vol (ATR-based), cap scaling to ±2x, with minimum size floor. Backtest with slippage model and half-life checks.
- Kalman-filter hedge ratios for pairs:
  - Estimate dynamic hedge ratio online; trade when spread z-score exceeds threshold and mean-reversion half-life short enough.
- Cost-aware ranking ensemble:
  - Score candidate trades by expected net return (predicted return − expected cost) and select top-k that fit margin/exposure constraints.

## Final notes

- Advanced methods can yield edges but introduce model, governance, and operational complexity. Always quantify net benefit after costs and run extensive out-of-sample and stress tests before deploying live.
- Prioritize simple, robust approaches first; iterate with disciplined research and monitoring.

# More Frequent Timeframes

Purpose: A concise introduction to how strategy design, execution modeling, risk, and failure modes change when moving from 4H/1D to higher-frequency timeframes (e.g., 1H, 15m, 1m). Focus is theoretical and prescriptive—enough to guide adaptation without deep implementation detail.

### High-level differences (theory)

- Signal density and sample size
  - More candles → more signals and trades; enables faster learning but increases noise.
  - Shorter horizons tend to produce lower signal-to-noise ratio per trade; need larger sample sizes for statistical confidence.
- Market microstructure matters more
  - Bid/ask dynamics, order-book depth, and quote updates drive fills and slippage. Tick-level behavior can dominate P&L.
  - Latency, partial fills, and order routing become critical.
- Costs and turnover
  - Turnover rises substantially; transaction costs (spread, commissions, market impact) quickly erode gross edges.
  - Execution algorithms (slicing, smart limit placement) are required to control cost.
- Volatility and thresholds
  - Price moves measured in pips shrink relative to spread; ATR scaled to timeframe is smaller—so constant multipliers (e.g., SL = 2*ATR14) imply different absolute risk.
  - Volatility clustering appears at different scales; microstructure noise produces spurious breakouts.
- Signal persistence and regime lifetime
  - Regimes change faster; models must adapt more frequently or use very short retrain windows.
  - Mean-reversion half-lives at higher frequency may be too short for human intervention, requiring fully automated execution.
- Execution primitives and order types
  - Use of IOC/FOK, pegged orders, and advanced routing increases; exchange/broker features vary and matter more.
- Risk and correlation
  - Higher intraday correlations across pairs during news spikes; portfolio exposures can change quickly within day.
  - Overnight swap costs are less relevant for intraday, but intraday funding and financing of margin can still affect capital use.

## Practical implications (what to change)

- Data fidelity
  - Require tick or 1m data; strict handling of timestamp accuracy, message ordering, and missing ticks.
- Transaction-cost modeling
  - Move from static spread assumptions to dynamic, time-of-day dependent spread/slippage models and explicit impact functions by size.
- Order handling
  - Implement low-latency, idempotent APIs, fast heartbeats, and aggressive retry/backoff. Local pre-trade checks must be cheaper and faster.
- Sizing & risk
  - Reduce per-trade notional due to higher trade frequency; incorporate turnover constraints and daily loss limits more aggressively.
- Validation & CV
  - Use intraday purged CV and maintain out-of-sample testing pockets that reflect intraday regime shifts (e.g., market open volatility).
- Monitoring & automation
  - Real-time monitoring with millisecond-to-second granularity; automated circuit breakers and soft/hard kill switches critical.
- Backtesting realism
  - Backtests must simulate limit order book behavior, partial fills, queue position, and microstructure-induced biases (avoid using only candle-close fills).

## Common failure threads when switching to higher frequency

- Underestimating transaction costs
  - Edge disappears when realistic spreads, commissions, and impact are applied; many strategies that look profitable on 4H/1D become unprofitable intraday.
- Latency and execution mismatches
  - Strategy logic assumes instant fills at candle close; in intraday settings, delays produce significant slippage and signal obsolescence.
- Data and lookahead errors
  - Improper handling of asynchronous tick data, mismatched timestamps, or resampling artifacts create lookahead bias or inconsistent signals.
- Overfitting to microstructure noise
  - Large feature sets with high-frequency inputs produce spurious patterns; models memorize noise and fail live.
- Insufficient infrastructure
  - Using broker APIs or hardware inadequate for required speed leads to frequent rejections/outages during critical times.
- Volatility regime micro-spikes
  - Intraday news or liquidity withdrawals cause extreme slippage and forced losses; without fast safeguards, losses compound quickly.

- Reconciliation and bookkeeping strain
    - Huge volume of events increases reconciliation complexity; missed or mislogged partial fills lead to mismatched positions and risk miscalculation.
- Overtrading and behavioral drift
    - More signals tempt parameter tinkering and over-optimization; lack of discipline increases drawdown risk.

## Quick mitigation checklist (if moving down in timeframe)

- Recompute transaction-costs empirically on target timeframe; require realistic worst-case scenarios.
- Lower risk per trade and enforce strict daily loss caps and max trades per day.
- Upgrade data to tick/1m with verified timestamps; implement message de-dup and ordering logic.
- Implement and test limit-order fill models with queue-position heuristics; simulate partial fills.
- Increase monitoring cadence and automated safety controls (latency alerts, fill-quality thresholds, immediate kill-switch).
- Restrict model complexity; require minimum trades-per-parameter rule to avoid overfitting.
- Run short-duration live pilot with micro-capital to observe fill behavior before scaling.
- Document and automate reconciliation at higher frequency (per-minute snapshots) to catch divergence early.

## Closing remark

Moving to more frequent timeframes can increase opportunities but requires disproportionate investment in execution, data quality, cost modeling, and operational controls. Treat the shift as a new product: revalidate every assumption (costs, latency, capacity, model stability) rather than transplanting 4H/1D rules unchanged.

# Costs of Trading

This chapter provides practical templates and codable methods to estimate real-world costs, breakeven AUM, capacity, and business viability for automated Forex strategies (4H / 1D, ~10-day horizon).

## Key cost components (per strategy)

- Execution costs
  - Spread cost (realized): average executed spread in pips converted to account currency.
  - Commission: per lot or per notional.
  - Slippage: realized difference between intended fill price and executed price (modeled/stressed).
- Financing & holding costs
  - Swap/rollover: daily long/short financing per notional.
  - Overnight funding or financing fees for margin.
- Operational costs
  - Infrastructure: servers, cloud instances, redundancy, market data subscriptions.
  - Development & maintenance: dev time, QA, monitoring tooling.
  - Brokerage fees: withdrawal, wire, account fees.
- Compliance & overhead
  - Legal, audit, tax, and accounting fees.
- Capital costs
  - Cost of capital or opportunity cost (use discount rate for owner capital).

## Unit economics — per-trade and per-year

- Define per-trade cost (example functions):
  - cost_spread = (spread_pips / pip_value) * position_size
  - cost_commission = commission_per_lot * lots_traded
  - cost_slippage = expected_slippage * position_size
  - cost_swap_total = daily_swap * holding_days * position_size
  - total_per_trade_cost = sum above
- Annualize:
  - trades_per_year × average_per_trade_net_return (after costs) → gross/ net return.
  - turnover = sum(|notional_traded|) / AUM per year.

## Breakeven AUM and capacity modeling

- Breakeven AUM (simple):
  - Let E_net_return_per_notional = expected_net_return_rate (after per-notional costs).

    - Fixed annual operational cost F (USD).
    - Breakeven_AUM ≈ F / E_net_return_per_notional
- Capacity with market impact:
    - Impact model: slippage_pct = α + β * (order_notional / ADV)
    - Simulate increasing AUM: larger order sizes → higher β*(size/ADV) → per-trade returns decline.
    - Find AUM where net return after cost <= target hurdle (e.g., risk-free + required margin).
- Practical steps:
    - Estimate ADV (average daily volume) per pair or liquidity metric.
    - Model distribution of order sizes given target position sizing and scaling rules.
    - Run scenario sims (base, stressed liquidity, peak-volume) and plot net return vs AUM.

## Profitability waterfall (report)

Present a waterfall from gross P&L to net owner profit: - Gross P&L - − Spread cost - − Commission - − Slippage - − Swap - = Strategy Net P&L (trading) - − Infrastructure & ops - − Taxes - − Other fees - = Net owner profit

## Pricing products (if offering strategy as service)

- Fee models:
    - Performance fee (e.g., 20% of profits) with high-water mark.
    - Flat management fee (AUM%) + lower performance fee.
    - Subscription per seat or per-broker connector.
- Example: For retail signal provider, backtest breakeven AUM to cover support/dev and compute per-client pricing.

## Tax & legal considerations

- Tax treatment varies: capital gains vs ordinary income; consult local advisor.
- Withholding and VAT may apply depending on jurisdiction and business model.
- Maintain trade-level P&L and tax lot histories.

**Ensure you know your country’s legal regulations regarding to income from trading.**

## Codable checks & reporting

- Daily: realized P&L breakdown (by cost component), turnover, margin utilization.
- Weekly/monthly: expected vs realized costs, capacity headroom metric.
- Alerts: running ops costs exceed X% of gross trading profit → raise pricing or scale down.

### Example pseudo-workflow (annual cost simulation)

```
for AUM in grid:
  simulate_trading(AUM, impact_model)
  compute_annual_gross()
  subtract_variable_costs()
  subtract_fixed_ops()
  record_net_margin(AUM)
plot(net_margin vs AUM)
```

### Lessons Learned

### Purpose

Chapter provides structured templates and anonymized exemplar postmortems with actionable lessons to improve future strategy design, risk controls, and operational robustness.

### Postmortem structure (template)

- Summary: one-paragraph incident summary and impact on P&L/AUM.
- Timeline: precise timestamps (UTC) of key events.
- Root cause analysis: technical, market, or human factors.
- Contributing factors: list systemic weaknesses.
- Detection: how the issue was discovered.
- Mitigations & fixes: immediate and long-term remediation steps.
- Metrics & evidence: equity curve, trade logs, MAE/MFE, latency, spreads.
- Action items: prioritized, owners, and deadlines.
- Post-implementation review: scheduled follow-up to verify fixes.

### Representative postmortems (anonymized, concise)

1. Case: Overnight Gap Blowup (holding weekends)

- Summary: Strategy held several positions over weekend; large geopolitical gap caused outsized loss of 8% equity.
- Root cause: Time-based exit rules allowed weekend exposure; tail event not modeled. Swap/overnight risk underestimated.
- Lessons:
    - Do not hold large notional over known gap windows without explicit tail risk hedges.
    - Model weekend gap distributions and include gap-cost-stress scenarios.

    - Introduce weekend auto-reduction policy and larger SLs or remove weekend exposure for certain pairs.

1. Case: Broker Liquidity Withdrawal During News

- Summary: On a high-impact NFP release, broker widened spreads and rejected limit orders; several market orders filled at extreme slippage producing 3% drawdown.
- Root cause: Inadequate news gating and overreliance on limit fills without fallback controls. Broker behaviour under stress differed from demo.
- Lessons:
    - Block entries and optionally reduce size within 60 minutes pre/post high-impact events.
    - Implement fill-quality detection and immediate pause/new-entries suspension during spread > threshold.
    - Maintain secondary broker or liquidity route for critical exits.

1. Case: Model Retrain Overfit with Feature Leakage

- Summary: New ML model showed stellar backtest but collapsed in live after retrain—20% drawdown in 2 months.
- Root cause: Label leakage (future window overlap) during training and insufficient purged CV. Model learned regime-specific artefacts.
- Lessons:
    - Use purged CV and strict OOS protocols; log training data snapshots and seeds.
    - Limit model complexity; require minimum trades-per-parameter.
    - Implement shadow testing before live swap.

1. Case: API Credential Compromise (small unauthorized trades)

- Summary: Unexpected positions opened overnight due to leaked API key script; quick detection stopped further loss.
- Root cause: API key committed to repo during development; no rotation.
- Lessons:
    - Enforce vaulting of secrets, key rotation, least-privilege keys, and automated alerts for anomalous API activity.

1. Case: Scaling-induced Slippage (capacity mis-estimate)

- Summary: Scaling AUM from $2M → $10M produced performance collapse due to unmodeled market impact.
- Root cause: Capacity estimate based on per-trade notional without ADV-relative impact model.
- Lessons:
    - Recompute capacity with impact model; scale gradually with monitoring and adaptive sizing.

## Common cross-cutting lessons

- Always reconcile live fills against backtest assumptions and maintain per-trade logs.
- Build layered defenses: pre-trade checks, execution monitors, and post-trade reconciliation.
- Automate short-term mitigations (pause on anomalies) and schedule manual reviews for root causes.
- Keep incident runbooks and on-call rotations; simulate incident drills periodically.

## Practical postmortem checklist (codable)

- Pull immutable trade logs and compare to broker statements.
- Compute deviation metrics: realized spread vs modeled, avg slippage, fills rejected.
- Re-run backtest with observed spreads/slippage window.
- Create action list with severity tags and remediation deadlines.

# Trading Ethics

Purpose: the chapter provides concise guidance on ethical trading behavior, avoiding market abuse, recognizing red flags that may trigger regulatory scrutiny, and best practices to ensure compliant algorithmic trading.

## Ethical principles

- Do no harm: avoid strategies that exploit systemic vulnerabilities, create unwarranted volatility, or harm counterparties.
- Transparency: maintain auditable logs of decisions, data, and model changes.
- Fairness: do not use privileged access to disadvantage other market participants.
- Accountability: assign ownership for automated decisions and incidents.

## Market abuse categories to avoid

- Spoofing: placing orders to create false supply/demand signals with intent to cancel before fill.
- Layering: submitting multiple non-genuine orders across price levels to manipulate perceived depth.
- Quote stuffing: flooding the market with orders to slow competitors or exploit latency.
- Wash trading / self-trading: executing trades that result in no genuine change of beneficial ownership to create illusion of activity.
- Insider trading: trading on material, non-public information.

All above are illegal in many jurisdictions; do not design strategies that rely on these behaviors.

## Red flags for regulators and counterparties

- High order-to-execution ratio with large cancellations.
- Repeated large orders clustered just to move quote then cancel.
- Rapidly changing order patterns that coincide with information releases.
- Concentrated activity that repeatedly triggers forced liquidations in counterparties.
- Significant discrepancies between declared strategy behavior and observed trades.

## Practical safeguards to show compliance

- Order-behavior limits: cap cancel rates, require minimum time-in-book for orders unless explicitly for risk exit.
- Logging: preserve full order book events, order lifecycle, and operator actions for at least regulatory-prescribed retention period.
- Access controls: require multi-person approvals for strategies that place large or potentially market-moving orders.

- Testing policies: never test market-impacting behaviors on live venues; use sandbox or dedicated testing venues.
- Pre-deployment review: legal & compliance sign-off with documented rationale for automated behaviors.
- Incident reporting: clear internal escalation for suspicious activity and cooperation procedures with regulators.

### Responsible ML & algorithm design

- Interpretability: favor explainable decision rules for high-impact automated actions.
- Human oversight: implement human-in-the-loop thresholds for actions that could materially affect markets.
- Avoid adversarial tactics: do not create algorithms to game other participants' risk controls or data feeds.

### Industry best practices & code of conduct

- Follow exchange and broker rules on algorithmic trading.
- Benchmark behavior to industry standards (e.g., ISDA/AFME guidelines where applicable).
- Maintain transparent documentation of strategy intent and risk limits that can be shared with counterparties under NDA.

### Response to regulatory inquiry (practice)

- Keep an incident pack: logs, reconciliations, configuration snapshots, and operator notes.
- Reply promptly and transparently; cooperate with audits.
- Engage counsel for guidance if potential violations are suspected.

# Glossary

A concise, implementation-focused glossary of key Forex and trading terms used in this book. Definitions emphasize codability and practical use in automated systems.

- 4H / 1D
  - 4H: four-hour candle timeframe (UTC-aligned).
  - 1D (daily): 24-hour candle aligned to 00:00 UTC.
- ATR(n)
  - Average True Range over n periods; default ATR(14). Used for volatility scaling, stops, and sizing.
- Ask / Bid / Mid
  - Ask: the price to buy the base currency.
  - Bid: the price to sell the base currency.
  - Mid: (bid + ask) / 2; used for signal calculations when modeling execution costs separately.
- Bar / Candle
  - OHLCV record for a timeframe (open, high, low, close, volume/tick-count).
- Backtest
  - Historical simulation of strategy logic including entry/exit, transaction-cost model, and slippage assumptions.
- Bootstrap / Monte Carlo resampling
  - Resampling methods to estimate metric distributions by randomizing trades, costs, or returns.
- Breakout
  - Price movement beyond a recent high/low lookback window used as an entry signal.
- Buy/Sell (Long/Short)
  - Long: buy base currency expecting appreciation.
  - Short: sell base currency expecting depreciation.
- Carry / Swap / Rollover
  - Carry: interest differential between currencies in a pair.
  - Swap/Rollover: broker financing credit/debit for holding positions overnight.
- Commission
  - Explicit per-trade or per-lot fee charged by broker.
- Conditional orders (OCO, OTO)
  - OCO: One-Cancels-Other. OTO: One-Triggers-Other. Conditional logic to manage linked entries/exits.
- Correlation
  - Statistical relationship between returns of two instruments; used for diversification and portfolio constraints.

- Drawdown
    - Peak-to-trough decline in equity. Report both magnitude (%) and duration (time).
- ECN / STP / Market-maker
    - ECN: electronic communications network (transparent price aggregation).
    - STP: straight-through processing (passes orders to liquidity).
    - Market-maker: broker may internalize orders and set prices.
- Edge
    - Expected net return per trade after costs.
- Execution latency
    - Time between order decision and broker acknowledgement/fill; impacts slippage and fill probability.
- Fill / Partial fill / Fill probability
    - Fill: order executed. Partial fill: only portion executed. Fill probability: modelled likelihood of limit order execution.
- Forward testing / Paper trading
    - Live-simulated trading with real market data and execution without real capital; final step before live.
- Hedging
    - Holding offsetting positions to reduce net exposure.
- Half-life (mean reversion)
    - Time for a mean-reverting spread to decay by 50%; estimated from AR(1) coefficient.
- Idempotency
    - Order/system operations that can be retried without duplicating side effects; crucial for robust APIs.
- In-sample (IS) / Out-of-sample (OOS)
    - IS: data used for model/parameter fitting. OOS: holdout data for unbiased evaluation.
- Limit order / Market order / Stop order
    - Limit: execute at specified price or better.
    - Market: execute immediately at current price.
    - Stop: becomes market or limit when price trigger reached.
- Liquidity / Depth of book
    - Liquidity: ability to trade size with minimal price impact. Depth of book: available volume at price levels.
- Lookahead bias
    - Using future information inadvertently in backtests; critical error that produces over-optimistic results.
- Lot / Contract size / Notional
    - Lot: standard unit for trade size (e.g., 100,000 base units for standard FX lot).
    - Notional: the dollar (or account base) value of position.
- MAE / MFE
    - MAE (Max Adverse Excursion): worst price move against a trade while open.

    - MFE (Max Favorable Excursion): best price move in favor while trade open.
- Margin / Margin call / Margin closeout
    - Margin: collateral required to open/maintain leveraged position.
    - Margin call: requirement to add funds.
    - Margin closeout (liquidation): broker forced closing when margin falls below threshold.
- Market impact
    - Adverse price movement caused by executing the trade itself (function of order size vs liquidity).
- Mid-price slippage model
    - Slippage applied relative to mid-price, then spread added separately for execution realism.
- Monte Carlo trade-sequence resampling
    - Randomly reordering or sampling trades to estimate distribution of equity outcomes.
- Net Exposure / Gross Exposure
    - Net: algebraic sum of long/short exposures. Gross: sum of absolute exposures across positions.
- Nominal / Notional value
    - Face value of position used for margin and sizing calculations.
- Order State Machine
    - Deterministic lifecycle for orders (Created → Sent → PartiallyFilled → Filled → Cancelled → Rejected) with idempotent transitions.
- Overnight gap
    - Price jump between close of one trading day and open of the next; material for 1D strategies.
- Overfitting
    - Excessively tailoring model to historical data leading to poor OOS performance.
- pandas
    - Refers to the Python library “pandas” (written lowercase). Pandas is a data-analysis library widely used by traders and quants to load, clean, analyze, and visualize time series (like currency price data).
- Pair trading / Cointegration
    - Trading two correlated instruments by exploiting mean-reverting spread if cointegrated.
- Payoff ratio / Expectancy
    - Payoff ratio = avg_win / avg_loss. Expectancy = (win_rate * avg_win) − (loss_rate * avg_loss).
- Performance attribution
    - Breakdown of returns into components: gross P&L, commissions, spread, slippage, swap.
- Pip / Tick
    - Pip: smallest conventional price increment (e.g., 0.0001 for EUR/USD).
    - Tick: minimum price movement instrument-specific.

- Position sizing / Risk per trade
    - Rules converting risk budget (e.g., 0.5% equity) and stop distance into trade size.
- Price impact model (linear/nonlinear)
    - Mathematical relation estimating slippage as function of order size relative to liquidity.
- Purged CV / Combinatorial CV
    - Time-series cross-validation methods removing overlapping label periods to prevent leakage.
- Python
    - Modern and effective programming language. Popular for Artificial Intelligence application and for a lot of other areas.
- Quote stuffing / Latency arbitrage
    - Toxic market behaviors exploiting stale/slow pricing; avoid strategies that rely on such anomalies.
- Rollover window
    - Broker-defined time when swaps are applied (commonly 21:00–22:00 GMT as reference).
- Sharpe ratio / Sortino ratio / Calmar ratio
    - Sharpe: excess return / volatility.
    - Sortino: return / downside volatility.
    - Calmar: annualized return / max drawdown.
- Slippage
    - Difference between intended execution price and actual fill price.
- Spread
    - Ask − Bid at a given time; component of transaction cost.
- Stop loss (hard/soft/trailing)
    - Hard stop: fixed price where trade is closed.
    - Soft stop: discretionary or algorithmically trailed.
    - Trailing stop: stop that follows price with defined step/threshold.
- Strategy capacity
    - Maximum deployable capital before performance degrades materially due to market impact and liquidity limits.
- Synthetic pair
    - Constructed pair from two or more market instruments (e.g., EUR/JPY via EUR/USD × USD/JPY).
- Tick data / 1m data / Resampling
    - Tick: every market update. 1m: one-minute aggregated candles. Resampling: converting higher-frequency data to target timeframe.
- Timezone normalization
    - Aligning all data timestamps to UTC (recommended) for deterministic behavior.
- Transaction cost model (TCM)
    - Composite model including spread, commission, slippage, and swap applied during backtests.

- Trade lifecycle
    - Full record of trade events including order placement, fills, adjustments, and P&L; must be logged immutably.
- Trade frequency / Turnover
    - Trades per period and total portfolio turnover; impacts transaction costs and tax treatment.
- Value at Risk (VaR) / CVaR
    - VaR: loss percentile over horizon. CVaR (Conditional VaR): average loss beyond VaR threshold.
- Walk-forward analysis
    - Repeated rolling training/testing protocol to simulate ongoing model selection and stability.
- Whipsaw
    - Rapid alternating signals causing small losses; common in higher-volatility regimes.
- Windowing / Lookback
    - Number of past bars/periods used to compute indicators or features.
- Z-score (spread)
    - (spread − mean) / std; used to detect deviations for mean-reversion trades.

This glossary is intentionally concise and aligned with the book's codable defaults (UTC candles, ATR14, 0.5% risk). Use these definitions as the canonical references when implementing code, tests, backtests, and documentation.

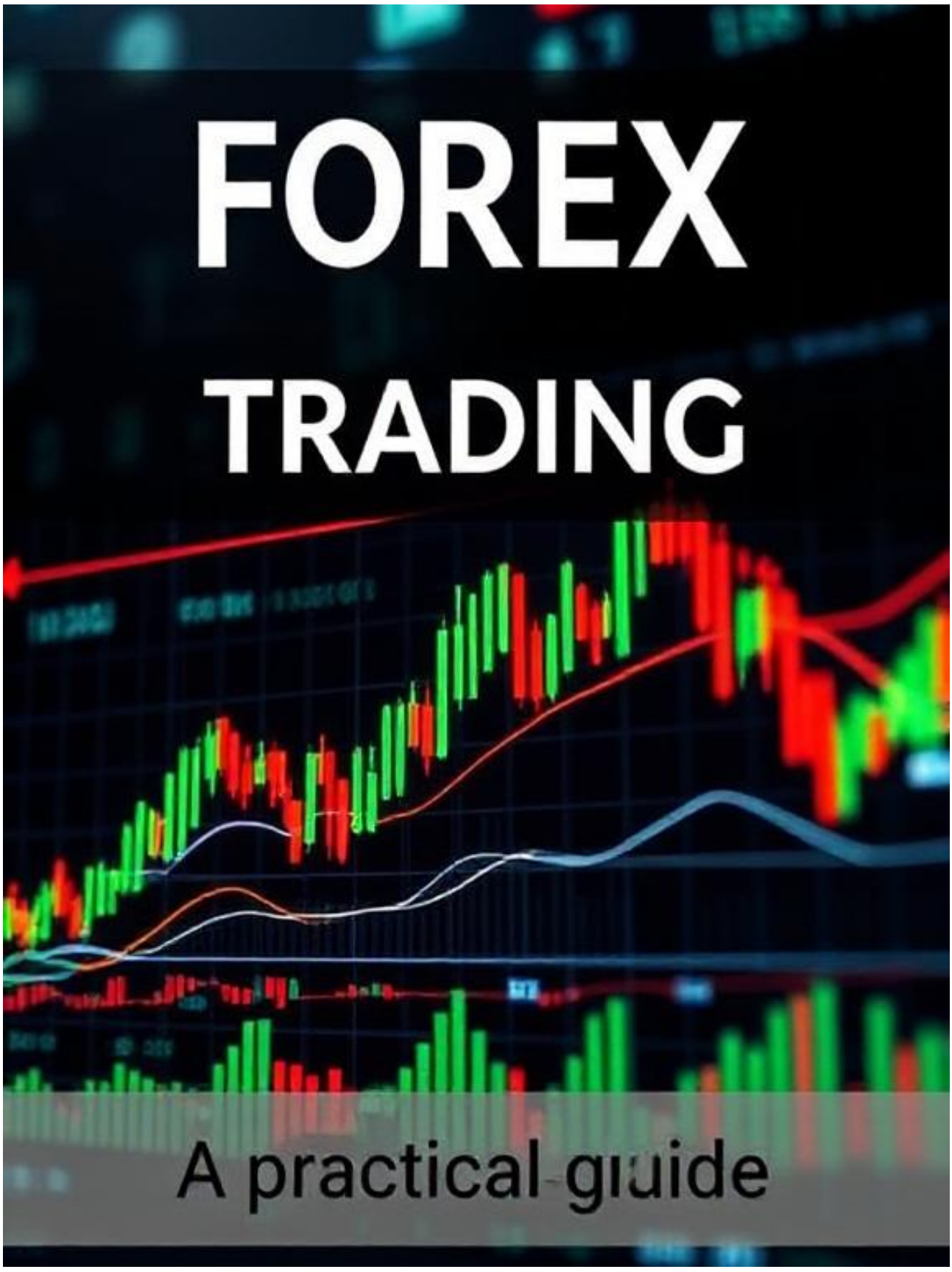
# About the author

Hill Smith is the pen name of a former city dweller who relocated to the countryside and found a new purpose in livestock mountain farming. He writes various stories, mostly revolving around farming, preparedness, and futures analysis, and some are science fiction. His writings are available as ebooks at this site:

*github.com/hillsmithbooks*

Comments can be sent to this address:

*hill.smith.books@gmail.com*


FOREX
TRADING
A practical guide