// MODULE · BACKTESTING

Every backtest lies. Honest ones admit how.

Seven biases, five engines, walk-forward, costs, and the eight metrics.

07	05	06	08
BIASES	ENGINES	WALK-FWD WINDOWS	METRICS

NEXURAL · SAGE IDEAS LLC
PUBLISHED 2026 · PERPLEXITY COMPUTER
nexural.com/learn/automation/backtesting

// table of contents

What you will learn.

Each section is self-contained. If you only have time for one, read section 03 — walk-forward is the single change with the biggest impact on whether your live performance matches your backtest.

// section 01

Seven biases.

Every backtest contains at least one bias. Honest research finds all of them. This section catalogues the seven biases that, in two decades of post-mortems, account for almost every retail blowup. For each: how it manifests, how to detect it, how to fix it. Severity follows Nexural convention — Sev1 (coral) is capital-loss risk, Sev2 (lime) is data integrity, Sev3 (magenta) is developer ergonomics.

Read in order, or jump to whichever applies to your current research.

// bias 1 of 7

Look-ahead bias

severity — SEV1

Strategy uses information unknowable at decision time.

Example

An indicator computed on the close of bar T is used to enter on the open of bar T. In live trading you cannot place an order before the close that produced the signal — yet many beginner Python notebooks do exactly that because the close of bar T is in the same row of the dataframe as the open of bar T.

How to detect

Shift every input by one bar and re-run. If returns collapse, look-ahead was present. vectorbt has .shift(1) built in; Backtrader's event loop naturally prevents it.

How to fix

Compute the signal on bar T, execute on bar T+1 open. Never let the same bar produce the signal AND the fill price. In dataframe code, always shift signals by one bar before multiplying by returns.

// bias 1 of 7 · code companion

Look-ahead detection in code

The code below implements the detect-or-fix recipe for this bias. Drop it into your research notebook — every honest backtest pipeline runs all seven of these checks before reporting any Sharpe number.

```
# Shift signal by 1 bar BEFORE multiplying by next-bar returns.
# Result: zero look-ahead exposure.
import pandas as pd

def safe_signal_returns(price: pd.Series, signal: pd.Series) -> pd.Series:
    # signal known at close of bar T -> apply to return on bar T+1.
    return signal.shift(1) * price.pct_change()

# Sanity check: compare 'fast' and 'slow' versions side-by-side.
fast = signal * price.pct_change()        # look-ahead version
slow = safe_signal_returns(price, signal) # honest version
print('look-ahead Sharpe:', fast.mean() / fast.std() * (252 ** 0.5))
print('honest Sharpe:    ', slow.mean() / slow.std() * (252 ** 0.5))
```

// bias 2 of 7

Survivorship bias

severity — SEV1

Backtest universe only contains names that survived to today.

Example

A 'buy bottom decile of S&P; 500 monthly' backtest over 20 years on Yahoo Finance shows 18% CAGR. The dataset never contained Enron, Lehman, Bear Stearns, Wirecard, or any of the 200+ companies that left the index after collapse. Live, the actual return is far lower because the dead companies are real losses you would have taken.

How to detect

Compare your universe at each rebalance against historical index membership tables (CRSP, Norgate). If your dataset starts every name on day one and never delists, it is survivorship-biased.

How to fix

Use a point-in-time universe: at each rebalance only consider companies that were in the index AT THAT DATE. Norgate, CRSP, EquityRT, and FactSet all expose this; free Yahoo/Stooq data does not.

// bias 2 of 7 · code companion

Point-in-time universe (pseudocode)

The code below implements the detect-or-fix recipe for this bias. Drop it into your research notebook — every honest backtest pipeline runs all seven of these checks before reporting any Sharpe number.

```
# Real-world: query a point-in-time membership table.
# Many free datasets do not expose this — you have to pay or build it.

def pit_universe(date) -> list[str]:
    # SELECT ticker FROM index_membership
    #   WHERE index = 'SP500' AND date <= :d
    #   AND (delisted IS NULL OR delisted > :d)
    return query(date)

# In a backtest loop:
for rebal_date in monthly_rebalance_dates:
    tickers = pit_universe(rebal_date)
    # rank, buy, repeat — using ONLY companies alive that day.
```

// bias 3 of 7

Overfitting (curve-fitting)

severity — SEV1

Parameters tuned to noise rather than signal.

Example

A grid search over 1,000 MA lengths returns Sharpe 2.8 at MA(47). MA(46) and MA(48) show Sharpe 0.4. The peak is a lucky number for that exact window. Live, the strategy uses MA(47) and never reproduces the result.

How to detect

Plot Sharpe across the parameter grid. A robust strategy shows a wide plateau of similar Sharpe values around the peak; an overfit one shows a single spike.

How to fix

Walk-forward analysis: pick parameters on 24 months, trade on the next 6, slide forward, repeat. Live Sharpe approximates the stitched out-of-sample Sharpe, never the in-sample one.

// bias 3 of 7 · code companion

Sharpe-plateau diagnostic

The code below implements the detect-or-fix recipe for this bias. Drop it into your research notebook — every honest backtest pipeline runs all seven of these checks before reporting any Sharpe number.

```
# A robust strategy shows a WIDE plateau around the optimum.
# An overfit one shows a single spike.
import vectorbt as vbt, numpy as np

windows = range(5, 200, 5)
sharpes = []
for w in windows:
    pf = vbt.Portfolio.from_signals(
        price,
        entries=price > price.rolling(w).mean(),
        exits=  price < price.rolling(w).mean(),
    )
    sharpes.append(pf.sharpe_ratio())

# Width of values within 90% of peak is your plateau.
peak = max(sharpes)
plateau = [w for w, s in zip(windows, sharpes) if s > 0.9 * peak]
print('plateau width:', len(plateau), 'parameters near peak')
# If plateau < 5 parameters, your peak is noise.
```

// bias 4 of 7

Zero slippage / fees

severity — SEV2	Fills assumed at mid-price with no transaction cost.

Example

A high-turnover mean-reversion strategy backtests at 4.2 Sharpe with 0 bps cost. With realistic 8 bps round-trip slippage plus commission, Sharpe drops to 0.4 and the strategy is unfundable. Costs scale with turnover — a 250-trades-per-year strategy at 10 bps round-trip loses 2.5% annual drag.

How to detect

Add 0 / 5 / 10 / 20 bps slippage tiers and plot Sharpe degradation. If 10 bps kills the strategy, it never had edge — costs just exposed it.

How to fix

Model slippage as a function of order size relative to ADV (% of average daily volume), and use historical bid-ask spreads. Tradier, IBKR, Polygon all publish realistic spread data.

// bias 4 of 7 · code companion

Slippage stress test

The code below implements the detect-or-fix recipe for this bias. Drop it into your research notebook — every honest backtest pipeline runs all seven of these checks before reporting any Sharpe number.

```
# Re-run the backtest at 5 cost levels. The strategy is investable
# only if Sharpe > 0.5 at the highest realistic level.
import vectorbt as vbt

for bps in [0, 5, 10, 20, 50]:
    pf = vbt.Portfolio.from_signals(
        price, entries=entries, exits=exits,
        fees=bps / 2 / 10000,       # half each side, in fraction
        slippage=bps / 2 / 10000,
    )
    print(f'{bps:3d} bps round-trip -> Sharpe {pf.sharpe_ratio():.2f}')
```

// bias 5 of 7

Data-snooping bias

severity — SEV1

You tested 50 strategies on the same data and reported the best one.

Example

You ran 50 backtests with different rules. The winner has Sharpe 2.1. Pure chance — at 95% confidence you would expect 2.5 false positives in 50 random tests on random data. The winner is not a strategy; it is a result of the multiple-comparisons problem.

How to detect

Track every variation you tried (yes, every one). Apply the Bonferroni correction: required p-value = alpha / N where N is the number of strategies tested.

How to fix

Reserve a hold-out set (the most recent 20% of data) that you NEVER touch during research. Validate the winner against the hold-out exactly once before going live. If it fails the hold-out, you data-snooped — start over.

// bias 5 of 7 · code companion

Bonferroni correction for multiple testing

The code below implements the detect-or-fix recipe for this bias. Drop it into your research notebook — every honest backtest pipeline runs all seven of these checks before reporting any Sharpe number.

```
# If you tested N strategies, your p-value threshold tightens.
import numpy as np
from scipy import stats

def deflated_sharpe(sr, n_strategies, n_obs, skew=0, kurt=3):
    # Bailey & Lopez de Prado (2014), simplified.
    var = (1 - skew * sr + (kurt - 1) / 4 * sr ** 2) / (n_obs - 1)
    z = sr / np.sqrt(var)
    p_one = 1 - stats.norm.cdf(z)
    return p_one * n_strategies  # Bonferroni

# A 2.1 Sharpe on 500 obs after testing 50 variants:
# corrected p ≈ 0.04 — borderline, not significant.
```

// bias 6 of 7

Regime narrowness

severity — SEV2

Backtest only covers one market regime.

Example

A momentum strategy backtested 2009-2021 (one of the longest bull runs in history) shows 1.8 Sharpe. The same logic over 1999-2002 or March 2020 produces -40% drawdowns the backtest never saw. The strategy was momentum riding a regime, not edge.

How to detect

Segment results by regime: bull/bear, high-vol/low-vol, rates-rising/falling. If any regime has negative expectancy, you have regime risk.

How to fix

Backtest across at least one full market cycle (10+ years) including 2008, 2020, and a rising-rate period. If your data does not include these, your strategy has not been tested.

// bias 6 of 7 · code companion

Regime segmentation

The code below implements the detect-or-fix recipe for this bias. Drop it into your research notebook — every honest backtest pipeline runs all seven of these checks before reporting any Sharpe number.

```
# Classify each bar into a regime, then report Sharpe per regime.
# If ANY regime is negative, you have regime concentration.
import pandas as pd

vol_20 = price.pct_change().rolling(20).std() * (252 ** 0.5)
regime = pd.cut(vol_20,
                bins=[0, 0.10, 0.20, 1.0],
                labels=['low', 'mid', 'high'])

strat_returns = signal.shift(1) * price.pct_change()
print(strat_returns.groupby(regime).apply(
    lambda r: r.mean() / r.std() * (252 ** 0.5)
))
# Example output:
#   low  -> Sharpe  1.8
#   mid  -> Sharpe  0.4
#   high -> Sharpe -1.1   <- regime risk
```

// bias 7 of 7

Benchmark selection bias

severity — SEV3

Strategy compared against a cherry-picked benchmark.

Example

You report alpha vs. the S&P; 500, but your strategy trades small-caps. The honest benchmark is the Russell 2000 — and against it, your alpha is zero. You did not beat the market; you beat the wrong market.

How to detect

Match the benchmark to your investable universe. Small-cap → Russell 2000. International → MSCI EAFE. Crypto momentum → DeFi Pulse Index, not Bitcoin.

How to fix

State the benchmark up front, and pick it BEFORE the backtest. If you change benchmarks because the comparison looks better, you are data-snooping benchmarks.

// bias 7 of 7 · code companion

Match benchmark to universe

The code below implements the detect-or-fix recipe for this bias. Drop it into your research notebook — every honest backtest pipeline runs all seven of these checks before reporting any Sharpe number.

```
# Pseudocode: pick the benchmark BEFORE you backtest.

UNIVERSE_TO_BENCHMARK = {
    'US_LARGE':   'SPY',
    'US_SMALL':   'IWM',     # Russell 2000, not SPY
    'INTL_DEV':   'EFA',     # MSCI EAFE
    'EM':         'EEM',
    'CRYPTO_MOM': 'DPI',     # DeFi Pulse Index, not BTC
}

def alpha(strat_returns, benchmark_returns):
    excess = strat_returns - benchmark_returns
    return excess.mean() * 252  # annualized

# If you change the benchmark because alpha 'looks better',
# you are data-snooping benchmarks. Pick once, report honestly.
```

// section 02

Five engines.

Pick the engine that matches your research style, not the one with the most stars on GitHub. Honest comparison across language, paradigm, speed, accuracy, live-trading bridge, and cost. No affiliate links.

Backtrader

LANGUAGE	Python	PARADIGM	Event-driven

Speed	~50k bars/sec single-thread. Slow for sweeps.
Accuracy	Bar-by-bar event loop. Honest about partial fills, commission, order routing.
Live bridge	Built-in: IBKR, Oanda, VC, ccxt. Maintenance slowed since 2022.
LOC (minimal)	~40 LOC for SMA crossover.
Why pick it	Strategy classes feel right; deep community indicators.
Cost	Free / MIT.

vectorbt

LANGUAGE	Python	PARADIGM	Vectorized

Speed	100×-1000× Backtrader for parameter sweeps; Numba JIT-compiled.
Accuracy	No event loop — slippage / partial fills must be modeled explicitly.
Live bridge	None native. Build your own broker layer.
LOC (minimal)	~12 LOC for SMA crossover.
Why pick it	Best-in-class parameter optimization and heatmaps.
Cost	Free open-source + paid vectorbtpro.

Zipline-reloaded

LANGUAGE	Python	PARADIGM	Event-driven

Speed	~30k bars/sec; daily-bar US equity research.
Accuracy	Originally Quantopian engine; minute-level is painful to set up.
Live bridge	Limited; community IB/Alpaca extensions exist but unmaintained.
LOC (minimal)	~50 LOC for SMA crossover.
Why pick it	Pipeline API for cross-sectional equity research is best-in-class.
Cost	Free / Apache-2.0.

LEAN (QuantConnect)

LANGUAGE	C# / Python	PARADIGM	Event-driven

Speed	~70k bars/sec local; cloud scales to thousands of nodes.
Accuracy	Production-grade order modeling per broker (slippage, partial fills, margin, fees).
Live bridge	First-class: IBKR, Tradier, Alpaca, Coinbase, Binance, Bybit, Kraken, OANDA, Zerodha.
LOC (minimal)	~45 LOC for SMA crossover.
Why pick it	One codebase from backtest → paper → live across many brokers.
Cost	Free locally; cloud $0-$80/mo research, more for live + data.

Custom event-driven (asyncio)

LANGUAGE	Python	PARADIGM	Event-driven

Speed	You control the budget. 10k-500k bars/sec depending on hot path.
Accuracy	Whatever you model. Most under-model partial fills, overestimate cash.
Live bridge	You build it. Same code runs in backtest and live.
LOC (minimal)	~200 LOC starter.
Why pick it	Total control of order semantics and live↔backtest code parity.
Cost	Free / engineering time.

// section 03

Walk-forward analysis.

Train on 24 months, lock parameters, trade on the next 6 months. Slide forward 6 months. Repeat. Six windows over five years gives six independent out-of-sample evaluations. The stitched out-of-sample equity curve is the honest result — never the in-sample number.

Why this matters

Live trading periodically re-fits a strategy: every month, every quarter, every year. Walk-forward mimics that cadence. If your strategy works only when fit on the whole 20 years at once, it does not work — it just happened to overfit a particular slice of history.

The six windows

Each row below shows one window. The TRAIN range is where parameters are picked; the TEST range is where those parameters are deployed unchanged. Reaching window 6 means six independent honest evaluations.

W	Train (months)	Test (months)	What you learn
1	1 - 24	25 - 30	First out-of-sample slice. If it diverges from in-sample, you overfit.
2	7 - 30	31 - 36	Slide forward 6 mo. Mirrors live re-fit cadence.
3	13 - 36	37 - 42	Three independent slices. Mean tells you what to expect live.
4	19 - 42	43 - 48	Variance across slices tells you how confident to be.
5	25 - 48	49 - 54	Stitch every out-of-sample slice into one equity curve.
6	31 - 54	55 - 60	If half are losing, the strategy has no edge — only lucky periods.

// section 03 (cont.)

Walk-forward in code.

A reference implementation using vectorbt. Each iteration fits parameters on the train slice, then deploys them on the test slice without further tuning.

```
import vectorbt as vbt
import pandas as pd

price = vbt.YFData.download("SPY", start="2020-01-01").get("Close")
train_months = 24
test_months  = 6
step_months  = 6

results = []
for start in pd.date_range(price.index[0], price.index[-1] - pd.DateOffset(months=train_months +
test_months), freq=f"{step_months}MS"):
    train_end = start + pd.DateOffset(months=train_months)
    test_end  = train_end + pd.DateOffset(months=test_months)

    # 1. Find best params on train
    train = price.loc[start:train_end]
    pf_grid = vbt.Portfolio.from_signals(
        train,
        entries=train > train.rolling(WIDTHS := range(10, 120, 5)).mean(),
        exits  =train < train.rolling(WIDTHS).mean(),
        freq="1D",
    )
    best_w = pf_grid.sharpe_ratio().idxmax()

    # 2. Lock params, trade on test
    test = price.loc[train_end:test_end]
    pf_test = vbt.Portfolio.from_signals(
        test,
        entries=test > test.rolling(best_w).mean(),
        exits  =test < test.rolling(best_w).mean(),
        freq="1D",
    )
    results.append({"window_start": start, "best_w": best_w, "test_sharpe": pf_test.sharpe_ratio()})

print(pd.DataFrame(results))
```

Reading the result

Look at the column test_sharpe across all six windows. If the mean is > 1.0 and the variance across windows is small (most windows positive), the strategy is robust. If half the windows are negative, the strategy has no edge.

// section 03 (cont.)

Anti-patterns.

Three common ways walk-forward analysis gets compromised. If you do any of these, the out-of-sample Sharpe is not honest.

Peeking at the test set

Looking at out-of-sample results, going back to tweak the in-sample logic, then re-running. Every iteration silently turns the test set into training data.

Too-narrow parameter grid

If your grid contains only the parameters that worked in past research, walk-forward will always find them. Use a wide grid that includes 'bad' values.

Train window too long

A 10-year train, 1-month test means each test is dominated by ancient parameters. Match train length to your real re-fit cadence.

// section 04

Realistic cost modeling.

Backtests with zero commission and zero slippage are not backtests. They are charts of what a price series did, with hopeful arithmetic on top. The single largest delta between backtest and live for most retail strategies is the cost model.

The four cost components

Component	What it is	Typical retail (US stocks)
Commission	Broker fee per share or per trade.	$0 retail (PFOF-funded) or 0.1-1.0¢/share.
Spread / slippage	Difference between displayed price and your fill.	1-10 bps large-cap, 10-50 bps small-cap.
Market impact	Your order moves the price against you.	Negligible at retail size; significant above 1% ADV.
Borrow cost (shorts)	Stock loan fee for shorting.	0-5% annual for liquid names; 20%+ for hard-to-borrow.

// section 04 (cont.)

The cost survival table.

How much edge survives at different cost levels. Numbers are illustrative — they apply a simple linear drag of turnover × cost to a gross Sharpe ratio. Real degradation is non-linear (worse at the tails), so treat these as a floor.

Strategy	Turnover (×/yr)	Gross Sharpe	Net @ 5 bps	Net @ 10 bps	Net @ 20 bps
Low-freq (monthly)	2	1.2	1.16	1.12	1.04
Mid-freq (weekly)	6	1.8	1.68	1.56	1.32
High-freq (daily)	30	2.5	1.90	1.30	0.10
Intraday	250	3.5	0.50	-2.50	-8.50

Reading the table: a high-frequency strategy with 30× annual turnover and a 2.5 gross Sharpe collapses to 0.10 Sharpe at 20 bps round-trip costs. If you cannot trade at < 10 bps (retail with PFOF + small size in liquid names), the strategy is not investable.

// section 04 (cont.)

Modeling slippage in code.

vectorbt and Backtrader both accept a slippage parameter. Set it from real historical bid-ask spreads, not from a guess. Polygon, IEX Cloud, and IBKR all publish quote data; for crypto use the venue's order book history.

```
# vectorbt — turnover-aware slippage model
import vectorbt as vbt

# 8 bps round-trip = 4 bps each side (entry + exit)
pf = vbt.Portfolio.from_signals(
    price, entries=entries, exits=exits,
    fees=0.0002,        # 2 bps each side commission
    slippage=0.0004,    # 4 bps each side slippage
    freq="1D",
)
print("Net Sharpe:", pf.sharpe_ratio())
print("Net annual return:", pf.annual_return())
```

Sanity check

After modeling realistic costs, plot the equity curve. If it lost money in 2018 (low-vol), 2020 (high-vol), or 2022 (rising rates), and you still want to deploy it, you have decided the strategy works only in specific regimes — fine, but be honest about that.

// section 05

Eight metrics. What each hides.

Sharpe ratio alone is not enough. Honest reporting includes at least these eight. Each metric reveals one dimension of performance; each one hides at least one important thing. The eighth metric is whatever the previous seven do not catch.

For each metric below we give: the formula, what a good value looks like, what the metric silently hides, what to always report alongside it, the deeper interpretation, and a real war story where the metric told the truth but the full picture told a different one. The goal is not to memorize formulas — it is to internalize the limits.

Read these in order on first pass, then keep this section as reference. Every backtest you publish should carry all eight of these numbers, with confidence intervals where sample size permits.

// metric 01 of 08

Sharpe ratio

Formula. `(mean(returns) - rf) / std(returns) × √(annualization)`

Good value. Above 1.0 net of costs; above 2.0 excellent; above 3.0 suspicious.

What it hides. Penalizes upside the same as downside; says nothing about left-tail risk; can be high while drawdown is unsurvivable.

Always pair with. Sortino (downside-only) and maximum drawdown.

Deeper read

Sharpe is symmetric: a 5% spike up costs the ratio the same as a 5% spike down. That is mathematically convenient and behaviorally wrong — investors care about losses more than equivalent gains. A Sharpe of 2.0 measured over 18 months can disappear over the next 6, especially if the strategy's edge depends on a single regime (volatility-low equities, USD weakness, carry, etc.). Always report Sharpe alongside its 95% confidence interval — for daily data over 3 years, the standard error is roughly Sharpe/√n_years, so a reported Sharpe of 1.5 over 3 years has a CI roughly [0.6, 2.4]. That uncertainty is the honest story.

War story

LTCM (1998) reported Sharpe ratios above 4.0 in 1996-97. Within 4 months in 1998 the fund lost $4.6B and was wound down. The Sharpe was correct given the data observed; the data observed did not include a Russian sovereign default. Sample size and regime coverage matter more than the headline number.

// metric 02 of 08

Sortino ratio

Formula. `(mean(returns) - rf) / std(downside returns)`

Good value. Typically 1.3-1.6× Sharpe for positive-skew strategies.

What it hides. Still summary statistics. A 30% drawdown that recovers fast can show a healthy Sortino.

Always pair with. Max drawdown and time-to-recover.

Deeper read

Sortino fixes Sharpe's symmetry problem by only penalizing returns below a target (usually zero or risk-free). For strategies with naturally positive skew — long volatility, trend-following, lottery-ticket options — Sortino tells a more accurate story than Sharpe. For naturally negative-skew strategies (carry, short volatility, mean reversion), Sortino looks too good because the disasters are rare; pair it with conditional value at risk (CVaR) at 95% or 99%. Rule of thumb: if Sortino/Sharpe > 2.0, you have heavy positive skew; if < 1.2, you have hidden left tail risk.

War story

Many quant equity 'low volatility' funds posted high Sortino numbers in 2017-2019 because the downside was muted. February 2018's 'Volmageddon' liquidated the inverse VIX ETN XIV in a single day after years of low-downside performance. The Sortino looked beautiful right up until it didn't exist.

// metric 03 of 08

Calmar ratio

Formula. `CAGR / |max drawdown|`

Good value. Above 1.0 means you make in a year what your worst drawdown took.

What it hides. Single-point estimate. A 3-year drawdown shows the same Calmar as a 3-month one.

Always pair with. MAR ratio (average drawdown) and underwater curve.

Deeper read

Calmar is intuitive — it converts pain into time. A Calmar of 1.0 means you earn back your worst loss in one year of normal performance. Below 0.5 means your edge does not adequately compensate for capital risk. Above 2.0 over 5+ years is extraordinary. The blind spot is duration: Calmar treats a 25% drawdown lasting 36 months identically to a 25% drawdown lasting 3 months, even though the first one would force most investors to redeem. Always plot the underwater curve (drawdown over time) alongside the Calmar number.

War story

AQR's risk-parity products had elegant Calmar ratios through 2014. The 2015-2016 commodity drawdown lasted 14 months — within historical norms by depth but exhausting by duration. Investors with monthly liquidity terms redeemed before recovery. Time underwater is its own form of risk.

// metric 04 of 08

Maximum drawdown

Formula. `max(peak - equity) / peak`

Good value. Personal risk tolerance. Retail abandons at 20%; institutions at 15%.

What it hides. Says nothing about the path. -20% over 8 months feels different from -20% in 3 days.

Always pair with. Time-to-recover and Ulcer Index.

Deeper read

Max drawdown is the single most useful retail-facing risk metric because it answers the only question that matters during a drawdown: 'how bad does this get before I capitulate?' But it is a single point estimate — by definition, it has no distribution. Always extend it: (1) drawdown duration in days, (2) recovery duration in days, (3) the 5 worst drawdowns ranked, not just the worst, and (4) what was happening in the market when the drawdown occurred. A 22% drawdown during the COVID crash is qualitatively different from a 22% drawdown during a quiet 2017.

War story

Long-Term Capital's max drawdown over its full operating life was 92%. The drawdown duration was 5 weeks. The recovery duration was 'never' — the fund was wound down. Reporting only 'max drawdown 92%' would understate the catastrophe; the duration and recovery profile complete the picture.

// metric 05 of 08

Ulcer index

Formula. `RMS of percent drawdown values across the equity curve`

Good value. Lower is better; under 5% is comfortable.

What it hides. Single number with no widespread reporting; counterparties may not recognize it.

Always pair with. Max drawdown (point estimate) and equity curve plot.

Deeper read

Ulcer integrates drawdown depth and duration into a single number — exactly what max drawdown misses. Strategies with the same max drawdown but spent more time underwater have higher Ulcer values, accurately reflecting the lived experience. The downside: nobody outside Martin Capital and a small subset of CTAs uses it as a reporting standard. Investors will ask 'what was your worst drawdown?' not 'what was your Ulcer index?' Compute it for yourself as an internal honesty check; report it alongside max drawdown, never instead of.

War story

Martin & McCann, who invented the metric, designed it explicitly to address how stocks with the same max drawdown can feel very different. Their original 1989 paper showed that a stock with a 30% drawdown that recovered quickly had an Ulcer roughly half that of a stock with the same 30% drawdown that lingered. Lived experience compressed into one number.

// metric 06 of 08

Win rate

Formula. `winning trades / total trades`

Good value. Strategy-dependent. Trend: 30-40%. Mean reversion: 60-75%.

What it hides. 85% win rate can still be net-losing if losses are larger than wins. Meaningless alone.

Always pair with. Profit factor and average win / average loss.

Deeper read

Win rate is the most weaponized metric in trading marketing because retail intuitively wants to be 'right' often. A 90% win rate sounds astonishing. A 90% win rate means 9 winners average $100 and 1 loser averages $1,100 — the strategy loses $200 over 10 trades. The win-rate-to-payoff relationship matters far more than either number alone. Honest reporting includes: win rate, average win, average loss, largest win, largest loss, and the win rate required to break even given the average payoff ratio. If the strategy needs >80% win rate to break even, it has almost no margin for regime change.

War story

Many short-volatility option-selling strategies advertised by retail educators tout 90%+ win rates. The math: collect $1 in premium 9 times, lose $20 once, net $-11. Real funds like LJM Preservation & Growth ran this exact profile and lost 80%+ in a single February 2018 day. The win rate was honest. The payoff ratio buried the truth.

// metric 07 of 08

Profit factor

Formula. `sum gross profits / sum gross losses`

Good value. Above 1.5 net of costs; above 2.0 excellent.

What it hides. Easy to inflate by removing or capping a few large losses. Run sensitivity on the largest 5 trades.

Always pair with. Sharpe and trade count.

Deeper read

Profit factor is the cleanest single-number answer to 'does the edge actually exist?' A profit factor below 1.0 means losses exceed gains — not investable. Between 1.0 and 1.3, the edge is real but fragile to costs and execution. Between 1.5 and 2.0, robust. Above 2.0 over hundreds of trades, exceptional. The fragility test: recompute profit factor with your single best trade removed, then with your top 5 best removed. If profit factor collapses below 1.2, your edge is concentrated in tail events that may not recur. This is the 'one trade away from a martingale' test.

War story

In 2019, a widely-shared TradingView Pine script for 'momentum breakout' showed a profit factor of 3.4 over 6 years. Removing the single best trade (a March 2020 GameStop-like move on a different ticker) dropped profit factor to 1.6. Removing the top 5 dropped it to 0.9. The strategy had no statistical edge; it had survivorship into one regime.

// metric 08 of 08

Average exposure

Formula. `average percent of capital deployed across all bars`

Good value. Strategy-dependent. Trend systems: 60-90%. Statistical-arb: 10-30%.

What it hides. A 4 Sharpe strategy deployed 5% of the time is interesting but not investable as stand-alone.

Always pair with. Capital efficiency (return on used vs. allocated capital).

Deeper read

Exposure tells you whether the strategy is a standalone product or only useful in a portfolio. A Sharpe-4 strategy in cash 95% of the time generates almost no compounding — capital sits idle earning the risk-free rate. Such strategies are valuable as overlays or as components in a multi-strategy portfolio, but standalone they produce too little return-on-allocated-capital to justify the operational burden. Always report Sharpe on returns (when invested) and Sharpe on the full equity curve including cash periods.

War story

Renaissance Medallion has very high exposure and reported Sharpe well above 2.0 — that combination is what makes it Medallion. Most public quant strategies achieving Sharpe > 3 do so by being deployed under 20% of the time. The Sharpe is real; dollar-weighted returns are modest unless the strategy is part of a larger portfolio.

// section 06

Honest reporting checklist.

Before you call a backtest 'done', tick each item below. If any one fails, you do not yet have evidence the strategy works.

[ ] 01	Data is point-in-time (no survivorship bias).
[ ] 02	Signals are shifted by ≥ 1 bar before execution (no look-ahead).
[ ] 03	Walk-forward analysis ran ≥ 6 windows across ≥ 5 years.
[ ] 04	Realistic commission and slippage modeled (not zero).
[ ] 05	Strategy tested across at least one bear market (2008, 2020).
[ ] 06	Hold-out set was untouched until final validation.
[ ] 07	Benchmark matches the investable universe.
[ ] 08	Sharpe, Sortino, Calmar, max drawdown, Ulcer, win rate, profit factor, exposure all reported.
[ ] 09	Trade-count is statistically significant (≥ 100, ideally ≥ 300).
[ ] 10	All variations tried are documented (multiple-testing correction).

Ten boxes. If you cannot tick all ten, document which you cannot tick AND why — that is the honest report.

// further reading

Sources and further reading.

[1] Bailey & López de Prado, 'The Probability of Backtest Overfitting', 2014.
https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2326253

[2] Harvey & Liu, 'Backtesting', Journal of Portfolio Management, 2015.
https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2345489

[3] López de Prado, 'Advances in Financial Machine Learning', Wiley, 2018.
https://www.wiley.com/en-us/Advances+in+Financial+Machine+Learning-p-9781119482086

[4] vectorbt documentation — portfolio modeling and slippage.
https://vectorbt.dev/api/portfolio/

[5] Backtrader documentation — analyzers and commission info.
https://www.backtrader.com/docu/

[6] QuantConnect LEAN — order modeling and brokerage models.
https://www.quantconnect.com/docs/v2/writing-algorithms/reality-modeling/

[7] Norgate Data — point-in-time historical equity universes.
https://norgatedata.com/

[8] CRSP — historical US securities database with delisting events.
https://www.crsp.org/products/research-products/crsp-us-stock-databases

[9] Pardo, 'The Evaluation and Optimization of Trading Strategies', Wiley, 2008.
https://www.wiley.com/en-us/The+Evaluation+and+Optimization+of+Trading+Strategies%2C+2nd+Edition-p-9780470128015

[10] Aronson, 'Evidence-Based Technical Analysis', Wiley, 2006 — multiple-testing correction.
https://www.wiley.com/en-us/Evidence+Based+Technical+Analysis-p-9780470008744

Continue reading → Next module: Data Engineering