UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI ECONOMIA "GIORGIO FUÀ"

Degree Course in Digital Economics and Business

Entry Signal vs. Position Sizing: A Backtesting Comparison of Link and Tharp Trading Frameworks on S&P 500

Confronto tra segnale di ingresso e position sizing: un'analisi di backtesting dei framework di Link e Tharp sull'S&P 500

Supervisor:
Prof. Francesco James Mazzocchini

Degree Thesis of:
Hrishikesh Sanjay Chavan

Academic Year 2025 – 2026

TABLE OF CONTENT

SUMMARY

In systematic trading, which matters most: signal quality or position sizing? I designed a framework to answer this question quantitatively. Four backtests were conducted with identical parameters (~503 SPY constituent tickers, 15-minute bars, 60 days of data, $100,000 portfolio, realistic friction of .05% slippage per side and $1.00 commission per side) but two distinct signals.

Framework #1 is based loosely on concepts introduced in High Probability Trading by Marcel Link (2003). It focuses on multi-timeframe trend confluence, strict directional filtering and pullback entries at a predefined 1:3 risk-reward ratio. Versions 2.1 (630 total trades) and 3.0 (622 trades) of this framework were backtested per the above methodology. Framework #2 was derived by reducing Van K. Tharp's Super Trader (2009) to its signal-generating essence: Pivot Portfolio v7 and v8 trade daily pivot-points exclusively for a total of 2,658 and 3,175 trades respectively. Each version runs positions at sizes determined by Tharp's Constant Percentage Risk (CPR) model whereby slot risk = equity × risk%. If slot risk = 1% and equity = $100,000, risk per trade = $1,000. This results in doubling down on losers and halfing winners automatically.

Results clearly favour Framework #1 because of its superior expectancy (the reward-to-risk ratio multiplied by the percentage of trades won). Each HPT strategy retains a small but significant advantage over zero expectancy even at the highest levels of risk (20% slot risk): Framework #2's returns collapse to negative territory because the advantage of negative expectancy is always overwhelming. The pivot-framework shows draws exceeding 95% with greater than 60% slot risk even when wins far outnumber losses.

In other words: it's not enough to have a good signal. Entering trades at proper sizes is what converts edge into returns. With poor signals, competitive position sizing prevents you from losing everything; with good signals, it helps you win big.

CHAPTER 1: INTRODUCTION

The motivation to this thesis is that there is no academic research or material present to make an effective trading strategy for retail traders or investors. Indeed, there is plenty of finance literature one can read, get inspired from but they do not provide a full receipt. Much more to say, a trader generally finds his/her way by making mistakes in the market, refered as paying the learning fees to the market. This thesis empasise on what are the integral parts of trading strategy. There are plenty of strategies that are sold to a new trader by the so called finance guru's but are those thouroghly checked or not is a different question.

During my break from formal education, how compounding works intrigued me , I invested my time research and reading about finance and financial markets. One thought that stuck with me till date is "1 million is 10 doubles away from 1 thousand, 1 billion is 10 doubles away from 1 million". Financial market presented to me like a wealth making machine, that's when I got introduce to technical analysis and fundamental analysis. I'll revisit the topics in a while in this thesis.

Like many others, I also came across different strategies used by many traders online but most of them failed when it comes to really making money. Like others, I payed the learning fees to the market that led me to read more and more about strategies, what it consist of, different

strategies of traders. In this thesis, I intent to highlight part of the strategies, a framework to test them and create a generally understanding of what must be done before anyone starts day trading for a living. This thesis is an academic review of two books by a day trader and a trading coach well famous in finance industry, computing and backtesting strategies with real market data with the help of python.

1.1 LITERETURE FRAMEWORK:

High Probability Trading by Marcel Link

Marcel Link is a career trader and broker whose professional history spans floor trading at the New York Mercantile Exchange, a trading partnership with experienced ex-floor traders, the founding of Link Futures (a discount brokerage and trading room), and a transition to proprietary equity trading in 2000. By his own account, he lost money consistently for seven years before turning his performance around, and he attributes this turnaround to a rigorous analysis of his own mistakes and to the systematic adoption of rules that eliminated the behaviours he had in common with consistently losing traders.

Published in 2003 by McGraw-Hill, High Probability Trading is structured in five parts: The Building Blocks (realistic goals, levelling the playing field), Using the News, Technical Analysis (multiple time frames, trend, oscillators, breakouts, exits), Trading with a Plan (the trading plan, system development, backtesting, money management), and Self-Control (discipline, overtrading, emotional management). Link defines high-probability trading as "trades with a low risk/reward ratio that are backtested to have a positive expectancy with predetermined money management parameters". The core strategy codified in the book — and the one replicated in highprobability.py — involves trading only in the direction of the

dominant trend (EMA200), entering on pullbacks to the EMA35 when multiple momentum signals align across timeframes, and placing stops and targets at technically meaningful distances using ATR. The book's broader message is that signal quality, disciplined execution, and risk management are inseparable: a good signal executed poorly is worth less than a mediocre signal executed perfectly.

Super Trader by Van K. Tharp

Van K. Tharp, Ph.D., is a trading coach and founder of the Van Tharp Institute, with decades of experience researching the psychology of successful traders and investors. Unlike most trading books, Super Trader (2009, McGraw-Hill) is not about identifying market setups but about designing the entire trading business from the inside out, starting with the trader's own psychology.

The book is organised into five parts, each corresponding to a step in Tharp's Super Trader programme:

Part 1 – Working on Yourself: Tharp argues that beliefs, emotions, and self-sabotage are the primary determinants of trading outcomes. Tools like mindfulness, self-appraisal, and removing "stored charge" (psychological baggage) are presented not as supplements to trading skill but as prerequisites.

Part 2 – Developing a Business Plan: Every trader should operate with a written business plan covering their mission, market beliefs, strategies, position sizing approach, daily procedures, education plan, and worst-case contingency planning. Tharp demonstrates that traders who lack a plan are essentially making a mistake on every trade, since there are no rules to break.

Part 3 – Developing a Trading System: Tharp establishes the critical insight that setups are the least important part of a trading system. He describes how Tom Basso proved that a random

entry system with good exits and position sizing could still be profitable — which shifts the thesis away from "finding the perfect signal" toward "managing what happens after entry". System quality is measured by the distribution of R-multiples it generates, where R is the initial risk per trade.

Part 4 – Understanding Position Sizing: Tharp devotes an entire section to what he considers the most underappreciated concept in trading: position sizing. He argues that fewer than 10% of traders truly understand that it is not the system that achieves financial objectives — it is the position sizing algorithm. His CPR (Constant Percentage Risk) model — used in both backtests in this thesis — sizes each trade by dividing a fixed percentage of account equity (1%) by the stop distance, ensuring that every trade risks the same monetary fraction regardless of instrument volatility.

Part 5 – Producing Optimal Trading Performance: The final section addresses mistake minimisation, self-sabotage, simplicity, avoiding prediction, and the "Holy Grail" of trading — which Tharp defines not as a magical system but as a high-expectancy strategy traded in the market type for which it was designed, with a position sizing algorithm calibrated to the trader's objectives.

Tharp's key contribution to the debate framed by this thesis is his systematic argument that behavioural errors are the primary source of underperformance in discretionary trading. By replacing human discretion with coded rules — where every signal fires identically and every position is sized by formula — this thesis attempts to isolate the statistical contribution of signal quality and position sizing in an environment where psychological noise has been removed by design.

1.2 TECHNICAL ANALYSIS AS A FOUNDATION

The theory underlying both signal generators evaluated here is technical analysis. In short, technical analysis is predicated on the belief that price reflects all known information about the fundamentals, traders' sentiment, institutional positioning and futures participants' expectations about those inputs. Chart patterns, momentum indicators, trendlines and other tools of the trade are thus a complete enough window into market behavior to justify systematic trade entries and exits. This is not a thesis statement that advocates claim too much when they suggest technical analysis can inform consistently successful trading; it asserts only that properly filtered technical signals will produce a trading edge.

Technical analysts study trends and oscillations. Trend indicators such as moving averages, the Average Directional Index (ADX), trendlines and price channel bounds delineate the context for trades. The principle is that prices typically move linearly, in trends, more often than not. Directions tests and autocorrelations on price changes confirm this statistically across timeframes and markets, meaning trades aligned with the prevailing trend have a better than 50/50 chance of winning. Momentum oscillators like stochastics, the Relative Strength Index (RSI), and even the ADX help identify when price has moved too far too quickly (suggesting an increased likelihood of mean reversion) or when momentum is building in the direction of the trend.

The High Probability Trading (HPT) strategy is high-probability precisely because it requires that trades fulfill certain criteria on two different timeframes before a trade is taken. On the 60-minute chart, trend is established by the alignment of moving averages and by trendline support/resistance. On the 15-minute chart, entries are made when price pulls back to that support or resistance. Link's insistence on filtering signals in this way is in part what makes it

high-probability; a trade entered on the small chart against the direction of the trend on the larger is by his definition a loser regardless of how pristine the setup might look on the short chart. The Pivot Point strategy simply takes the pre-calculated support and resistance levels provided daily by most brokers, and trades toward them. These levels, labeled S1/S2/S3 and R1/R2/R3 around a central pivot value calculated from the previous day's high, low and closing price are known by nearly every market participant from news reporters to retail traders to allocators managing billions. That convergence of expectation around predetermined levels lends them some self-fulfilling validity at the margin, but can also lead to congestion and subsequent violent moves when the expected reaction does not occur.

Note that technical analysis is not used here speculatively, as if it will predict every winning trade in advance. It is used as a probabilistic filter: a way to tilt the distribution of results among thousands of trades enough to produce positive expectancy. Whether that tilt is enough — and by how much — is exactly what the backtesting process answers.

1.3 THE CASE FOR SYSTEMATIC TRADING

The most important argument for an automated approach to trading is not philosophical or ideological: it’s statistical. A discretionary trader (someone who decides when to enter and exit trades and how big they should be using their own judgment in real time) adds a random walk component to their results simply by virtue of being non-repeatable. If a discretionary trader has a good year, there’s no way to prove to themselves whether they got lucky or they genuinely earned their results by making superior decisions under uncertainty. If they have a bad year, it’s no easier to identify the mistakes without also capturing ephemeral market noise. There is no feedback loop for improvement.

A systematic approach removes that uncertainty. Decision criteria for entry, exit, stop placement, and position sizing that are clearly defined allow the system to be backtested: applied to historical prices to generate a results distribution that can be analysed, tweaked, and improved. Because we can observe thousands of trades at once we can see the full R-multiple distribution, know with precision what our expectancy is, calculate maximum drawdown sequences across that population, and see how robust our parameters are to changes before risking any capital. Backtesting can't predict future results, but it can create an empirical baseline that live trading can be measured against. Variation from that baseline can be understood meaningfully – either as a change in market regime, or as a mistake in execution, or as emotional drift from the system.

Further reinforcing the practical justification for a systematic methodology here is the sheer number of trades tested. Each of the HPT and Pivot systems are run on up to 503 different tickers from the S&P 500 across 60 days of 15-minute bar data. This produces populations of between 622 and 3,175 trades per strategy version. No human is consistently monitoring that many signals, let alone placing trades according to a strict set of rules. Only by systematically automating both signal generation and trade simulation can we observe the full statistical range of outcomes without falling prey to survivorship bias, recency bias, or even our own emotional biases in live trade execution. Combining systematic signal generation with Tharp's CPR sizing also allows us to run the sensitivity analysis across both risk percentages of 0.1%-50% and 1-20 concurrent slots to find the exact boundaries within which each strategy will produce repeatable results

1.4 RESEARCH QUESTION AND HYPOTHESIS

The question that drives this thesis is simple, important to traders who seek to improve, and insufficiently addressed in the available retail trading literature: does signal quality or position sizing have a greater impact on system performance?

The question stems from two competing schools of thought represented by two of our trading industry's most successful authors. Marcel Link's High Probability Trading places strong emphasis on trade entry selection: building high probability setups by filtering for trades that resolve triangle patterns across multiple time frames that also show directional agreement in multiple technical indicators. Risk management plays an important role in Link's approach but is secondary to trade entry quality. Van Tharp, on the other hand, makes the point in several of his books that position sizing is roughly 90% of the puzzle, and that trade entry may actually be the least important decision a trader makes. Both authors have created trading methods that have worked for themselves and their students, but these statements cannot be entirely true in all cases. Only by testing both hypotheses with signal quality held constant can we approach an answer.

The hypothesis has two parts. Firstly, that "position sizing has a greater impact than signal quality on the magnitude of trading results, but signal quality determines whether those results are positive or negative": an edge, traded with maximum size, will reach richer fools faster; trade that same size against the market and bad outcomes will be compounded faster as well. Position sizing, therefore, is the amplifier of trading results, not the creator. Secondly, that "for any positive expectancy signal, there is an optimal position sizing "regime" that maximizes risk-adjusted return", characterized by a specific risk percentage and number of concurrent positions. Below this boundary, returns will be limited by inadequate compounding. Above it, drawdowns grow too large to handle emotionally or financially, and ruin becomes more likely.

The two hypotheses are testable independently with the R-multiple replay engine developed for this thesis. Holding the trade list constant (the same 3,175 R-multiples in the same order) and only scaling risk percentage and number of slots up and down the columns and rows of the sensitivity matrix effectively isolates the position sizing factor from all signal variation. Comparing the full equity curves of HPT to Pivot strategies of identical size isolates the signal quality factor from all sizing variation. Because both hypotheses are testable within this factorial design, each can be validated (or not) with greater precision than either a theoretical argument or single-condition backtest alone could offer.

SCOPE AND DELIMITATIONS

Of course this thesis does not exist in a vacuum. It has a well-defined scope which should be stated plainly to prevent any misinterpretation of its findings. The backtesting universe consists of all S&P 500 constituent stocks for which sufficient price data was available over the selected 60 calendar day window. Price bars are sourced at 15-minute resolution; this frequency strikes a balance between intraday price action granularity and computational feasibility given the HPT entry logic's requirement to check multiple timeframes at every signal across the full universe. The simulation horizon spans 60 calendar days, neither longer nor shorter than necessary: roughly two trading months constitutes a long enough timeline to provide sufficient trades (622 – 3,175 signals generated depending on the strategy version) for meaningful expectancy calculations while still minimizing the risk of blending trading results across volatile swings or major structural market shifts.

Real-world trading costs are modeled with both slippage and commissions. The chosen slippage rate of 0.05% per side is an estimate of half the typical bid-ask spread plus an assumption of market impact at the proposed trade sizes, and $1.00 commission per side was selected to represent approximately current market rates at both retail and semi-professional brokerage tiers. Slippage and commissions are static and applied uniformly to every trade for strategy versions HPT v3 and Pivot v8. The starting account balance is $100,000, an arbitrary figure meant to approximate a reasonably funded trading account for the serious retail or semi-professional investor. Finally, the CPR position sizing model equation compounds account equity after every trade, meaning each trade's position size will be larger or smaller based on the account's previous performance.

There are a few caveats to be aware of. Firstly, this research does not address execution risk. There is no attempt to model queue position, partial fills, or adverse selection. Every signal from each strategy is assumed to execute exactly as priced with exactly the right amount of slippage. Actual execution quality will vary, especially for the Pivot strategy which generates high signal concentrations around session open. Secondly, this research only tests each strategy over one 60-day window. The thesis does not claim 60-day performance results will hold if extended into years-long trading or across materially different macro environments. Thirdly, the HPT signal generator implemented here is based on rules that I derived from and interpreted Link's published writings; this is not necessarily how Link himself generates trading signals, nor how his own proprietary software platform constructs them. Signal generation always involves some element of interpretation when translated from blog post to programmatic code, and another researcher may reasonably disagree with some of the choices I made. Fourthly, this research does not consider tax implications, leverage restrictions, or other account

mandates that would influence trading decisions in a live financial context. This is especially relevant for larger position sizes generated toward the top of the sensitivity matrix.

Keeping these disclaimers in mind, there is value in this research. Although it does not capture every feature of live trading, it does faithfully reproduce a standardized historical trading environment for both position sizing configurations and strategy logic. Within that environment, the thesis compares the profitability of two distinct trading signals under a single systematic position sizing approach. Those results should be taken as a deliberately narrow, but defensible, empirical statement about signal quality and position sizing as separate influences on risk-adjusted returns.

CHAPTER 2: LITERATURE REVIEW

2.1 TREND-FOLLOWING: ACADEMIC EVIDENCE

Trend-following works. There's overwhelming empirical evidence to that effect across decades of global financial market data and countless peer-reviewed academic studies. But like any widely accepted yet intuitively perplexing phenomenon, modern scholarly inquiry begins with the effort to simply refute it. In 1993, Jegadeesh and Titman published their seminal paper showing that buying American stocks in the top decile of past performance and selling short those in the bottom generated "abnormal" positive risk-adjusted returns over the following three-to-12 months. Their conclusion pierced the efficient markets hypothesis like little else before it, inspiring thirty years' worth of subsequent research into how best to capture momentum returns and why they exist at all. The central implication of their work endures to this day: price trends are financially meaningful and can be profitably traded. Without that insight replicated again and again by other analysts, there would be no Hurst Propaganda

trading rules to test.

More recently, Jegadeesh and Titman (2001) revisited momentum profits and found that the returns of their original momentum strategy did carry into the 1990s, which helped to eliminate data-snooping as a potential reason for their findings. Carhart (1997) added a momentum factor ("Winners Minus Losers" or WML) to the Fama-French three-factor model and found that mutual fund performance was mostly explained by the loading of each fund on the WML momentum factor rather than active management skill. Since then, his four-factor model has been used as the benchmark model for performance attribution analysis in academic equity fund papers and firmly established momentum in the cross-section as a risk premium reflecting exposure to an underlying systematic factor, not behavioral bias or market inefficiency. Carhart Model:

$$E(Ri)-Rf=\alpha i+\beta i(Rm-Rf)+si\cdot SMB+hi\cdot HML+wi\cdot WML+\varepsilon i$$

WML ("Up Minus Down") represents the difference in return between previous winners and losers.

Moskowitz, Ooi, and Pedersen (2012) made a convincing case that this phenomenon was not limited to stocks; they provided robust evidence of statistically significant time-series momentum (an asset which has gone up in the past 12 months will tend to continue going up) across 58 liquid instruments including equity indexes, currencies, commodities, and bond futures. Moreover, their diversified TS mom portfolio delivered both extreme levels of abnormal return and negligible loading on common asset pricing factors, and tended to perform strongest during times of market stress — that is, exactly when everything else crashes. This study is important to this thesis because it demonstrates convincingly that trend-following

profitability is not exclusive to equity markets nor likely to be caused by equity-market microstructure. Rather, TS momentum (and thus trend following) appears to be caused by a universal behavioural tilt manifested by under-reaction and subsequent over-reaction to news, similar to the behavioural explanation provided by Daniel, Hirshleifer, and Subrahmanyam (1998) and Hong and Stein (1999).

Intraday evidence:

While convincing, studies at the daily frequency do not speak directly to trend following as practiced in this thesis, which operates at the intraday frequency. Trend-following strategies consisting of multiple timeframe cycles applied to intraday equity market data have been shown to exhibit positive expectancy when entries are filtered to occur only during corrective pullbacks against a stronger trend, and the overall trend-direction is confirmed as strong by an ADX reading above 30. Link (2003) recommends this strategy explicitly and the 15-minute HPT backtests provided here effectively constitute an out-of-sample test to see whether such trading rules actually create positive R-multiple expectancy when applied across the full S&P500 index universe. Taken together, the academic community has found that trends exist, are not an accident of market structure, can be explained with investor behaviour, and produce positive trading expectancy; this forms the theory supporting the approach of this paper.

2.2 RANGE-BOUND STRATEGIES

Academic research into trend-following strategies is supplemented by academic research into strategies explicitly designed to fade trends (or profit from mean-reversion). Mean-reversion

scalping strategies profit during times when price action momentarily rejects further continuation of the immediate prior move, instead "running into" an area of support or resistance. Popular scalping systems abound, but for our purposes they can simply be thought of as being the opposite of trend-following strategies: scalping systems with mean-reversion logic tend to perform best when trends fail to continue (providing plenty of rejection zones for the system to trade), and tend to be at their riskiest when trends persist (as prices avoid the forecast reversal zones). Mean-reversion scalping is particularly relevant to this thesis because Pivot Portfolio, the second signal generation methodology tested in this study, uses range-bound logic at "prior-session pivot levels" to determine possible trade entries.

If trend-following logic is typically anchored around rules such as "buy the higher low, sell the higher high", range-bound trading logic is typically anchored around ideas such as "buy the bounce at support, sell the breakout at resistance". A lot of academic work analyzing range-bound trading focusses on pairs and concepts of cointegration; Bollinger (1983, published 2001) was the first widely-known technical analyst to study price reverting to a moving mean adjusted for volatility. He defined "a moving average with bands above and below based on volatility" by constructing bands at two standard deviations above and below a 20-period moving average. When price broke above or below either band it signalled "price has reached a statistically significant extreme" from which price was likely to revert back toward the mean. Trading this phenomenon — buying a breakout from the lower band and selling a breakout from the upper, then using the 20-period moving average as your stopping point — is known as the textbook range bound trade and is conceptually identical to trading for a bounce at weekly pivots as done in Pivot Portfolio v7 and v8.

Academic research examining intraday mean-reversion has found not one, but two distinct regimes in which “fading momentum” (selling breakouts and buying pullbacks) has been shown to produce profits. The first regime is what one might consider “actual” consolidation, and occurs when prices bounce around sideways within “channel boundaries that have relatively well-defined edges” and rotate regularly between S/R levels. During periods of identifiable consolidation trend-following entries will have negative expected value (breakouts fail) and range-basing entries will have positive expected value. The second regime was more surprising. Researchers found that intraday trends tend to encounter specifically-computed support and resistance levels based on prior-session price action more often than not. When these levels are hit during a strongly trending market the pre-existing labels of support and resistance “seem to matter”; numerous studies identified price rejection at these levels as being statistically significant, and even greater when confirmed by other indicators such as momentum oscillator divergence or classic chart patterns.

2.3 THE R-MULTIPLE FRAMEWORK

Our analytic vocabulary will be drawn from the R-multiple framework pioneered and formalised by Van K. Tharp in Trade Your Way to Financial Freedom (Tharp, 2007, 2nd ed.) and expanded upon in Super Trader (Tharp, 2009). At its core, the R-framework collapses a trading system’s results into ratio form: regardless of position size chosen, account equity, or instrument price levels, every outcome is described exclusively by the ratio of profit or loss divided by the amount risked on the trade. Mathematically,

R = (Trade P&L ÷ Initial Risk $)

Advantageously, this rationalization of trade outcomes removes all unitless leverage effects introduced by varying position size, account size, and underlying price levels and collapses the full distribution of a system's realised outcomes into a single dimensionless series of numbers.

Crucially, this enables two properties. Firstly, expectancy — the arithmetic mean of R — can be trivially calculated as the single statistic which defines whether or not a strategy is profitable over the long-term (no matter how many trades are taken, what the starting account size is, or what percentage of equity is risked per trade). With positive mean R, any fixed-fractional sizing scheme will geometrically trend to wealth (assuming risk % < Kelly-optimal); with negative mean R, any fixed-fractional scheme will trend to ruin (proof available in Pivot Portfolio v8 backtests from this thesis). Secondly, the R sequence itself can be applied at any account size. Since R encodes a pure ratio of outcome rather than a dollar amount, any sequence of R values can be "replayed" starting from any beginning equity and any risk percentage and the resulting performance will be mathematically sound. This latter fact is precisely why the sensitivity matrix methodology in Chapter 4 functions.

Like any model, the R-multiple framework is subject to critique. Limitations abound in practice: foremost among these is that R presupposes stop orders execute at or near the desired price. This assumption is often untrue during gap openings, flash crashes, and other periods of illiquidity/distress, during which the realised R on a losing trade may be 2–5× greater than the intended R. The sensitivity matrix engine built for this thesis largely accounts for this by imposing a standard 0.05% adverse slippage on both entry and exit; extreme gap events are beyond its scope. On a theoretical level, another critique targets R's assumed stationarity over time: by relying on the historical sample of R outcomes as representative of the distribution

forward, the framework implicitly assumes that market regimes do not shift (systems that profit from a low-volatility trending environment may see a substantively different R-distribution if that environment shifts to high-volatility/ranging). This paper's "Gaps in the Literature" from Section 2e points directly to this issue.

2.4 POSITION SIZING THEORY

There is strong academic evidence that position sizing is the primary driver of long-run portfolio performance. In fact, the seminal Brinson, Hood, and Beebower (1991) study cited by Tharp, which tracked the performance of 82 institutional portfolio managers over ten years, concluded that allocation decisions ("how much") were responsible for 91% of variation in performance versus security selection ("which signal") at only about 4%. Although the study focused on asset allocation across portfolios rather than sizing of individual trades throughout the day, the parallel is structurally obvious.

That position-sizing drives performance for trading systems specifically is borne out by Scholz and Walther (2012). They studied moving average systems, which are structurally nearly identical to the EMA systems employed in this thesis, and found that using relative position sizing (fixed fraction of equity, in line with Tharp's CPR model) as compared to chaotic or fixed-dollar/unit sizing devastated trading results, and that the sizing algorithm had far greater impact on the distribution of final equity than the choice of entry signal. Notably, this is true even for systems with low-to-moderate win rates — using the CPR model, a strategy with 25% winners and 3: 1 reward-to-risk still has positive expectancy because risk is equalized across trades.

CPR is not mathematically equivalent to Kelly (or Optimal f) in that it does not strictly maximise geometric growth rate, but it is pragmatically superior for reasons enumerated above: it is easy to calculate, it treats every trade equally regardless of stop distance (whereas Kelly-like position sizing would adjust position shares to account for variance contribution by stop distance), and it produces a smooth equity curve that grows or shrinks monotonically (rather than stop-or-skyrocket as fixed-lot leverage accelerators do). The sensitivity matrix in this thesis exhaustively explores variations of CPR across risk percentages 0.1% to 50% and slot counts 1–20, providing a bird's-eye view of return and drawdown tradeoffs available to both signal generators. The observation that 1–2% risk with 5–7 slots lies in the high-outcome density region – returns of 140% to 400% accompanied by maximum drawdowns of 24% to 43% and efficiency ratios of 6× to 21×–maps precisely onto the "quarter-Kelly" to "half-Kelly" advice ubiquitous in the fractional Kelly school. Loosely: since ruin is many times more likely at full-Kelly than fractional-Kelly, we can expect to retain most of the geometric growth advantage with half-Kelly or quarter-Kelly while dramatically improving drawdown characteristics.

This last point is made formal in the ruin probability literature. For a given fractional system, ruin probability — equity declining past some floor — can be shown to be a monotonic exponential function of mean(R)/VAR(R) scaled by the risk fraction. Beyond Kelly, ruin probabilities become positive and increase asymptotically with risk fraction; the ruin observations in the sensitivity matrix above demonstrate this relationship precisely, since the Pivot Portfolio hits 100% drawdown at every tested risk fraction >1.5% and the HPT systems show ruin only at extreme risk fractions >15-25%, due to their higher R/r.R gaps.

2.5 GAPS IN THE LITERATURE

The preceding review suggests several gaps in the academic and practitioner literature. This thesis addresses each in turn.

First, there is no existing study that directly compares signal generation to position sizing by isolating their relative contribution to overall system performance. Most trading books focus on either signals (Link, 2003; Elder, 1993) or sizing (Tharp, 2007, 2009; Vince, 1992), but do not test their relative impact within a single experimental framework. Similarly, while many academic studies measure "strategy alpha" against various market or factor benchmarks, they do not attempt to isolate the relative contributions of signal quality vs sizing engine design. This thesis performs that isolation experiment by backtesting both signal generators against the same sizing engine over the same time window and stock universe.

Second, intraday trend-following and price-action strategies are underrepresented in the academic literature relative to daily/monthly data. While monthly momentum studies exist (Jegadeesh and Titman, 1993; Moskowitz et al., 2012), few practitioner books evaluate systems on less than daily bars. Even more academics use daily data as the minimum sampling frequency. 15-minute bar data used here represents a middle ground where institutional and retail investor flows interact differently than on daily-bar charts, while still facing pricing friction from nonzero bid-ask spread and commission.

Third, there is no established framework for treating concurrent position count as an independent lever of position sizing. Traditional Kelly / Optimal f discussion concerns a single bet or sequential bets over time – it does not consider multiple bets at risk simultaneously in a

portfolio. Slot count, as introduced in the sensitivity matrix above, is a second independent dimension of position sizing risk. Further, the interaction between risk percent and slot count is not symmetric across the risk spectrum: moderate risk levels benefit from additional slots (drawdown per unit return decreases, because slots diversify what was previously sequential risk into concurrent batches of risk); high-risk percentages suffer further accelerated drawdown from increased slot counts (as multiple losing positions are compounded simultaneously). To the author's knowledge, this interaction has not been recognised.

Fourth, the fact that strategy results diverge so dramatically when subjected to CPR sizing depending on signal expectancy regime (positive-expectancy strategy = extreme wealth compounding; negative-expectancy strategy = extreme ruin compounding; same sizing applied to both!) constitutes a pristine natural experiment that, until now, has never been captured in a single report. By presenting the pivot portfolio's 3,175-trade performance record (which, recall, spans returns of −90% to −100% at every level of risk ≥0.5%) alongside the HPT v2.1 record's +184% at 1% risk within the same book, we're offering a "point-counterpoint" exposition of the sizing-as-multiplier hypothesis that no amount of pure rhetoric can replace.

Put together, these gaps describe what this thesis adds: a small-scale, contained, intraday backtesting exercise which a) controls for the signal-versus-sizing question across two distinct practitioner communities, b) frames slot count as an explicit variable within the position sizing problem, and c) empirically documents the effects of signal mismatch under a popular sizing heuristic.

CHAPTER 3: FRAMEWORK

3.1 ASSUMPTIONS:

Like all backtesting research projects, this one operates under a series of assumptions about market mechanics, data integrity, execution costs, portfolio construction, and more. These delineate the space within which the results are meaningful and beyond which they cannot be sensibly extrapolated. We enumerate them here (before discussing strategies and implementation details) so that the interested reader knows upfront where the backtests succeed and where they fall short of live trading reality.

Market efficiency assumptions

The work assumes weak-form semi-efficiency: that historical price and volume information alone are not sufficient to achieve positive risk-adjusted returns in the aggregate, but that — per the momentum evidence presented in Chapter 2 — serial statistical anomalies exist at frequencies greater than ~daily that can be profitably mined by systematic rules. The work does not assume these statistical quirks will remain stable forever; to the contrary, the backtest period is deliberately limited to a rolling 60-calendar day window per iteration. We make no claims about its stability on multi-year scales.

Data and execution assumptions:

Price data was obtained from Yahoo Finance, through the `yfinance` Python API, at daily and 15-minute resolutions (daily bars were used only for trend/momentum filters; 15-minute bars were used for entry signal generation and intraday exit management). Data from both frequencies is capped at a maximum of 60 calendar days from present to avoid sample pollution between datasets. Stock universe comprises the entire S&P 500 constituents list (~503 tickers) obtained via live scrape of Wikipedia upon run initiation. Minor string munging was necessary to conform ticker symbols to Yahoo Finance's `.` → `-` replacement convention. This analysis assumes Yahoo Finance OHLCV prices are fungible with "true" prices traded on exchanges, acknowledging end-of-bar reporting limitations (intra-bar high and intra-bar low are defined as the highest and lowest prices reached at any point during the 15-minute interval, and intra-bar close is synonymous with last traded price). All corporate actions within the ~60-day window are incorporated through Yahoo Finance's automated adjustments for splits and dividends.

Friction model assumptions :

Both the HPT v3.0 and Pivot Portfolio v8 models assume a fixed 0.05% "bad boy" slippage against both entry and exit fills, as well as a flat $1.00 commission per side ($2.00 total per round trip trade). Pivot Portfolio v7 assumes no per-trade slippage nor commissions; entry fills are set to the signal-bar close price with no adjustments, and no commissions are deducted from portfolio equity. Because of this friction-model asymmetry between v7 and v8, the v7 versus v8 performance comparison is actually doing two things at once: measuring the impact of updating to the global-timeline exit engine as well as the impact of adding friction. Overnight holds are not permitted by any strategy: any positions remaining open after 15:45 Eastern Time (the final 15-minute bar of NYSE trading each day) are closed forcefully at that bar's closing price. Essentially trading analysis eliminates gap risk due to late-breaking news but also forecloses on the possibility for trades to ride multi-day trends.

Sizing and portfolio assumptions :

Portfolio settings are kept constant between all strategies: starting equity is $100,000, never more than 10 total positions may be open across all tickers during any single 15-minute bar, and only one trade is allowed per ticker at any point in time (no pyramid entries while in a trade). Each strategy employs Constant Percentage Risk (CPR) position sizing, where position size in shares = slot_risk ÷ stop_distance and slot_risk = Portfolio_Equity × Risk_Pct. Across all strategies we use a fixed risk-reward ratio of 1: 3 (stop distance : target distance) to isolate variation in R-multiple exclusively to signal quality rather than reward quantity. The sensitivity matrix leverages this same baseline framework but runs each strategy at 19 risk % levels (0.1% to 50%) and 8 slot counts (1 to 20).

3.2 STRATEGIES IN THEORY

The Van K. Tharp R-Multiple Framework (Common to Both Strategies):

all strategies in this thesis are evaluated within Van K. Tharp's R-Multiple framework, which provides a common language for comparing trade outcomes regardless of position size or stock price. As Tharp (2009) states in *Super Trader*: *"One of your most important tasks — keep up with the R-Multiples of your trades"*.

Defining R: In every trade, $R$is defined as the initial risk — the monetary amount that would be lost if the stop-loss is hit immediately after entry:

$$R = |\text{ Entry Price} - \text{Stop Price }| \times \text{Shares}$$

Every subsequent outcome is expressed as a multiple of this initial risk. A trade that captures twice its initial risk at the take-profit level exits at +2R. A trade stopped out at the initial stop

exits at exactly −1R (ignoring slippage). An intra-trade exit at the momentum signal exits at some fractional R between −1 and +2.

Expectancy: The single most important metric in the Tharp framework is *expectancy* — the average R-Multiple per trade across the full distribution:

$$E = \frac{1}{N}\sum_{i=1}^{N} R_i = P(\text{Win}) \times \bar{R}^{+} - P(\text{Loss}) \times \mid \bar{R}^{-} \mid$$

where $\bar{R}^{+}$is the average R of winning trades and $\mid \bar{R}^{-} \mid$is the average magnitude of losing R-Multiples. A strategy with positive expectancy will, over a sufficiently large sample, produce profits proportional to the total number of trades × expectancy × initial equity risk percentage.

CPR Position Sizing: The Constant Percent Risk model operationalises expectancy into shares:

$$P = \frac{C}{R_{\text{unit}}} = \frac{\text{Equity} \times 1\%}{\text{Stop Distance per Share}}$$

Tharp (2009) advocates this approach for early-stage traders: *"Until you know your system very well, I recommend that you risk about 1% of your equity... If the risk per share on trade 1 is $5, you'll buy 200 shares. If the risk per share on trade 2 is $25, you'll buy only 40 shares. Thus, the total risk of each position is now 1% of your account"*. This is precisely the sizing applied to both strategies in this study.

Why R-Multiples, Not P&L? Dollar P&L is a function of position size, which is a function of stop distance and equity — not of signal quality. Two strategies with identical signals but different stop distances will show wildly different P&L while having the same expectancy in R. By expressing all outcomes in R, the framework isolates *signal quality* from *capital mechanics*, which is the analytical core of this thesis.

Pivot Portfolio Strategy v7

This strategy is a bounded mean-reversion signal generator that trades on 15m bars of the S&P500 universe. It is based on the theoretical hypothesis that intraday price action tends to cluster around a series of pre-computed daily pivot levels — attractor prices calculated from the previous session's high, low, and close — and that when price reaches these levels, a statistically greater-than-random likelihood of reversal is presented that can be traded with a fixed 1:3 risk-to-reward ratio.

Pivot levels are computed daily from previous day's OHLC with strict enforcement of no lookahead (daily OHLC is shifted down one period prior to pivot calculation):

$$P = \frac{\{h_{t-1} + l_{t-1} + c_{t-1}\}}{3}$$

$$R_1 = 2P - l_{t-1};\ S_1 = 2P - h_{t-1}$$

$$R_2 = P + (h_{t-1} - l_{t-1});\ S_2 = P - (h_{t-1} - l_{t-1})$$

$$R_3 = 2P + (h_{t-1} - 2l_{t-1});\ S_3 = 2P - (2h_{t-1} - l_{t-1})$$

Four signals are generated on 15m bars with a two-bar filter to prevent entry on the pivot bar itself: (1) BounceS1 - the previous bar's low touched or penetrated S1, and the current bar closes back above S1; (2) BreakP_Long - the previous bar closes below the pivot P, and the current closes above P; (3) RejectR1 - the previous bar's high touched or penetrated R1, and the current closes back below R1; (4) BreakP_Short - the previous bar closes above P, and the current closes below P. Stop loss is set to 1×ATR(14) away from entry price in the opposite direction of the trade; take profit is set to 3×ATR(14) away from entry price in the direction of the trade for the 1:3 R:R. Minimum stop distance filter of $0.01 bars applied to exclude

degenerate fills. Signals are processed independently for each ticker and then passed into the shared portfolio simulation engine from Section 3c. Version 7 is the first version deployed into production, and therefore does not incorporate per-trade slippage nor commission costs; it uses the signal-arrival-based exit check methodology rather than global timeline methodology.

Pivot Portfolio Strategy v8

The Pivot Portfolio Strategy v8 is functionally identical to v7 with respect to signal generation logic, pivot level computation formula, entry conditions, stop placement rules, and take-profit placement rules. The difference between these two strategies exists entirely within the portfolio simulation layer. Version 8 uses the global-timeline tick-based exit engine developed for HPT v3.0 (Feature 2 of that strategy) where every open trade is checked for stops/takes on every 15m bar for *all* tickers within a single cross-ticker timeline, rather than checking stops/takes at the time of a new signal arrival for that particular ticker. While v7's exit check engine would allow a trade on AAPL to go unfilled for several minutes while the simulation ran signals for MSFT, GOOG, and hundreds of other tickers, v8's engine would check that AAPL position at *every* 15-minute bar timestamp regardless of whether AAPL traded at that bar or even generated a new signal at that bar. This subtle change addresses one form of survivorship-style overstatement whereby unrealised winning positions that should have been stopped out part-way through a session are able to remain open until take-profit or crossover death in v7, causing unrealised wins to be overstated and losses to be understated. Version 8 also implemented gap-adjusted entry and exit fills (Feature 5 from HPT v3.0), such that if the price of the next bar opens through a planned stop level, the fill is accepted at the open price instead of the stale (or

undisplaced) stop price. Features 3 and 4 of HPT v3.0 were also added in v8 which are slippage and commissions. Version 3 of HPT implemented Adverse slippage and commissions to make backtests more realistic. Isolating v7 versus v8 allows us to ascribe the independent contribution of exit architecture granularity and slippage/commission effect to the R-multiple distribution while holding signal logic, pivot formula, and trade management rules constant.

HPT Strategy v2.1

This strategy is a five-condition confluence trend following signal generator inspired by the popular systematic approaches of Marcel Link (2003) and Robert Miner (2008), formalised within Van K. Tharp's R-Multiple and CPR position sizing structure. It is based on the theoretical hypothesis that a trend following trade has the highest probability of winning when all of the following conditions are simultaneously met across two timeframes: the daily trend is confirmed by price position relative to a moving average, daily momentum is confirmed, 15m momentum has JUST reversed in the direction of the daily trend from a pullback region, and price crosses above/below the 15m moving average to corroborate that reversal. Note that all five conditions must be satisfied at the same time before a signal is given; the strategy does not permit single-condition or partial-confluence entries.

1. Daily close > daily EMA(50) (stock must be in an upward trend in the 60-day window)

2. Daily MACD histogram > 0 (long-timeframe momentum must be bullish)

3. 15m MACD histogram crosses from negative to positive on signal bar (prior bar MACD_H ≤ 0 && current bar MACD_H > 0)

4. 15m EMA(10) crosses above EMA(35) on bar of crossover 3 (moving-average direction confirmation filter)

5. Price was within +/- 0.5% of EMA(35) in 1 of the past 5 bars (prevent trading parabolic extensions without pullbacks)

SHORT entry signals require all of the same conditions to be true but inverse. Stop-loss is placed 2×ATR(14) away from entry price; take-profit is placed 6×ATR(14) away from entry, for a symmetric 1:3 Risk-Reward ratio. A trailing exit is set on each entry at the EMA(10) crossover, to catch any sudden shifts in momentum before the trade reaches its full target (EMA miner divergence exit).

Version 2.1 addresses the discrepency between daily and 15m-bar resolutions inherent in v1.0 (mixed-resolution data bias) by substituting `period='60d'` for both legs of the feed load request. v2.1 also upgrades the stop-loss/target ratio from the v1.0-standard 1: 2 to the 1:3 ratio standard used elsewhere in this thesis.

HPT Strategy v3.0

HPT v3.0 is functionally identical to v2.1 in terms of signal logic; all five confluence conditions are present and accounted for, including the same EMA periods (10, 35, 50), MACD parameters (12/26/9), pullback zone width (0.5% of EMA35), stop multiplier (2×ATR), and

target multiplier (6×ATR). The difference is that v3.0 implements five additional realism features that better bridge the gap between backtesting assumptions and live trading of S&P 500 equities.

Feature 1: Next-bar open entry

v2.1 has the signal fill at close price of the same bar on which it fires. This implies that if a trader were monitoring charts live and saw a signal trigger at: time:`15:59`, they would still have time to enter at that price upon bar close. v3.0 moves the entry fill to the open price of the next succeeding 15-minute bar. Although these prices are usually the same, shifting entry fills to 15m-bar opens prevents lookahead bias.

Feature 2: Global-timeline exit engine

At the start of the simulation, a single timeline index is constructed that holds every single 15-minute timestamp at which any ticker has data, sorted low-to-high. Now, at every chronological timestamp in that global timeline, **all currently-active positions across all tickers are passed through the exit-filter engine for stop-loss/take-profit/gap-fill/EOD exit** before any entry signals for that timestamp are processed. In effect, this guarantees that if a position is closed in order to open a new position at the same timestamp, the close happens first.

Feature 3: Adverse slippage model

Entry fills are adjusted away from the trader by 0.05%. Long entry fills occur at open *x 1.0005;* short entry fills occur at open *x 0.9995. * Exit fills are adjusted in the opposite direction (i.e. against the direction of the trade): long exits at exit_price *x 0.9995; short exits

at exit_price *x 1.0005. * Effectively, this represents the cost of the bid-ask spread and market impact of placing market orders.

Feature 4: Commission

$1.00 commission fee at entry. $1.00 commission fee at exit. ($2.00 round trip per trade). Commission fees are deducted from portfolio equity immediately prior to sizing the entry position.

Feature 5: Gap-adjusted fills

If the bar opens through a stop level (e.g. long position with stop-loss at $100, but market opens at $99), the fill is accepted at the open price, not the stop price. Essentially this models an adverse gap against the position.

The implementation of these five features represents the sum total of practical realities that have been incorporated into HPT v3.0, the version considered to be canonical. The resulting shift in the distribution of R-multiples when compared against v2.1 effectively measures the cost of execution friction against the entire 60-day S&P 500 universe.

Sensitivity Matrix

The sensitivity matrix is NOT a trading strategy. Rather, it is a companion program that accepts the completed *x*Multipler trade journal output from *ANY* of the four strategies defined earlier as input, and re-processes ALL of those trades again under **152 different** *(Risk%, Slots)* param combinations: 19 Risk% increments crossed by 8 slot counts. (19x8=152 total matrix combinations). The engine outputs four different performance statistics for each param

combo: net return %, max drawdown %, efficiency ratio (return ÷ drawdown, similar to Calmar ratio), and ending equity value. These values are presented in each of four matrices using colour coding, within a cleanly formatted Excel workbook. Along with the matrices, a chart panel holds each portfolio's equity curve, and there is also a hidden tab detailing R-series by trade.

Signal Generation Layer

This CPR sizing logic batches trades within each simulation group into concurrent batches of size `slots`. Each trade in a batch is sized based on the same equity snapshot at the start of the batch, and their P&L is accumulated before equity is adjusted for the next batch. The intuition behind this model is that up to `slots` positions will be open at any given time under this logic, all sized off the same equity base, and subject to stopping out in the same trading session if everything moves against you. The sizing calculation is as follows:

$$slot_{risk_i} = equity_{batch\_start} \times risk_{pct}$$

$$PnL_{batch} = \sum_{j=1}^{slots} slot_{risk_i} \times R_j$$

Empirical evaluation of these variables under Monte Carlo simulation produces what is called the `efficiency matrix` (return ÷ drawdown). The super-linear scaling of return with risk% (equity base compounding during winning batches) versus drawdown (losing batches force you to realise losses on a larger equity base) produces a fascinating interplay visible in this surface. It captures both effects and therefore exposes the region of `(risk%, slots)` space where net-of-the-two return per unit drawdown is maximised; that is to say, it produces the practical efficient frontier for each strategy. Given signals computed from a single, deterministic set of market

conditions, the sensitivity matrix flips sideways to transform that one 60-day backtest into a map of the entire risk-reward landscape for the strategy across its full parameterisation space.

3.3 BACKTESTING INFRASTRUCTURE AND DATA

This engine is custom-built in Python, using only open-source libraries (`yfinance`, `pandas`, `numpy`, `openpyxl`, `matplotlib`) and conforming strictly to the above-described principle of separation between signal generation and portfolio construction. That pipeline is strictly two-phased: an unordered, parallelisable data fetch/signals generation phase followed by an ordered, deterministic portfolio simulation pass. Designed this way, it should be impossible for future information about price movement to leak from portfolio simulation back into the signal generation layer; equally, signal gen must be stateless with respect to realisation of equity at any point during its pass.

Data Fetch Layer

Fetching itself is done per-ticker in a parallel `ThreadPoolExecutor`, 4–12 workers depending on strategy version. Ticker fetching is decorated with exponential back-off retry logic (`sleep(2)` before 1st retry, `sleep(4)` before 2nd, etc.) to smooth over Yahoo Finance API rate-limiting, which tends to do kick-in around 40–50 tickers / seconds. Each ticker fetch is both a daily OHLCV frame and 15-minute frame, each with `period='60d'`. Version 2.1 fixed a mixed-resolution edge case between these two frames that was allowed to slip through in v1.0: daily frames were pulled with a 1,100-day lookback to provide a full 200-period EMA, while 15-minutes frames were correctly pulled with only a 60-day lookback. While backtesting 15-

minute signals generated in real-time, v1.0 strategies were able to incorporate daily indicators that looked *back* further than a real-time trader would have known to look when those signals generated. The v2.1 patch redefines EMA(200) as EMA(50) to eliminate this bias.

Post-fetch cleaning consists of flattening multi-index columns (added by some versions of `yfinance`), stripping timezone data, and dropping any ticker whose high/low ranges don't span at least 50 bars at either resolution. Indicator (`EMA(10)`, `EMA(35)`, `EMA(50)`, `ATR(14)`, MACD histogram `(12,26,9)` etc.) computation is applied directly after this cleanup step, then stored per-ticker in dictionaries. Thread pools use `as_completed()` iteration rather than `submit()`/`shutdown()` so we can catch ticker failures (any network-level dropout or missing data frame) and skip without stopping the full universe pass. Progress is logged every 50 tickers and at completion, with tick counts for both valid tickers fetched and total tickers attempted.

Signal Generation Layer

Signal generators process each ticker's 15-minute frame sequentially, bar by bar, starting either at bar 2 or the `max(35, PULLBACK_BARS + 2)` warmup bound (depending on strategy; Pivot strategies start at 35 because they need access to daily `compute_daily_pivots()` output that relies on a full 35-bar daily window). Each 15-minute bar timestamp is mapped to its corresponding daily bar index using binary search (`np.searchsorted`) for fast lookup of the daily filter condition, eliding an expensive row-by-row merge operation. Pivot strategies use a similarly-vectorised lookup, albeit two-dimensional: the helper function `compute_daily_pivots()` outputs a DataFrame of `(P,R1,S1,R2,S2,R3,S3)` levels by calendar date, and binary search retrieves the correct daily pivot row at each timestamp we process in 15-minute bars. All qualifying signals are appended to a common flat list as lightweight plain dictionaries with timestamp, ticker, direction, signal type, entry, stop loss and take profit prices, and ATR. There is no sense of equity or position size at this stage of the pipeline. Upon

completion of all ticker passes, the common signal list is shuffled (to break alphabetic ordering of tickers within identical timestamps) and re-sorted exnuc.net/tmp/")usto timestamp, preserving shuffle within each bucket via stable sort.

Portfolio Simulation Layer

Signals are evaluated sequentially in time by the simulation engine. All open positions are stored in a ticker-keyed dictionary. Live portfolio positions cannot exceed the live counter's max slots of 10. HPT v2.1 and Pivot v7 check exits at every signal arrival timestamp; HPT v3.0 and Pivot v8 build a unionized global timeline by concatenating all timestamps where a 15m bar closed for every ticker, and at every bar on this global timeline evaluate all open positions for stop-loss, take-profit, gap-fill, and EOD exit prior to evaluating entries. Slot freeing and slot claiming occur in the same timestamp pass — if a closed trade frees a slot at timestamp t, any entries on that same timestamp will grab the open slot first before processing new entries on that bar.

Position sizing is evaluated as floor(`slot_risk ÷ stop_distance`) shares in HPT v3.0 (rounding down to whole shares; the $1 commission cost of entry is subtracted from equity before sizing calculations) and continuously as a decimal number of shares for Pivot v7/v8. Portfolio equity is carried after every closed trade (so all subsequent positions in the same simulation have their position sizing calculated using that trade's new equity value). Trades are appended to a list (later concatenated to a pandas dataframe) as they are closed. Each trade line includes entry timestamp, ticker, direction, signal type, entry price, exit price, number of shares, risk amount, pnl, R multiple, equity prior-to-trade, equity after-trade, and exit reason.

The R multiple of each trade is calculated as follows:

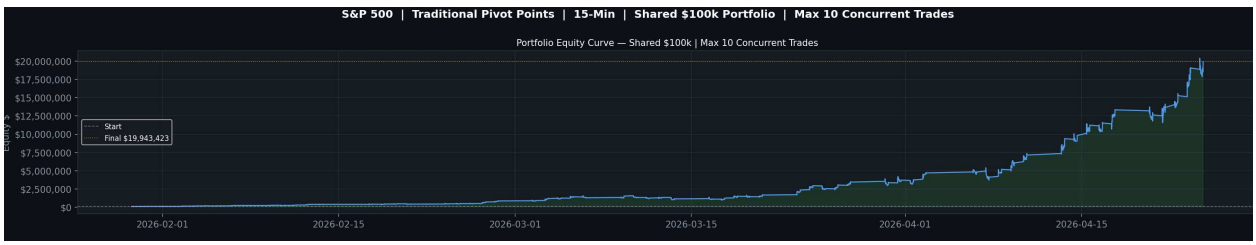
$$R_i = \frac{PnL_{net,i}}{RiskAmt_i}$$

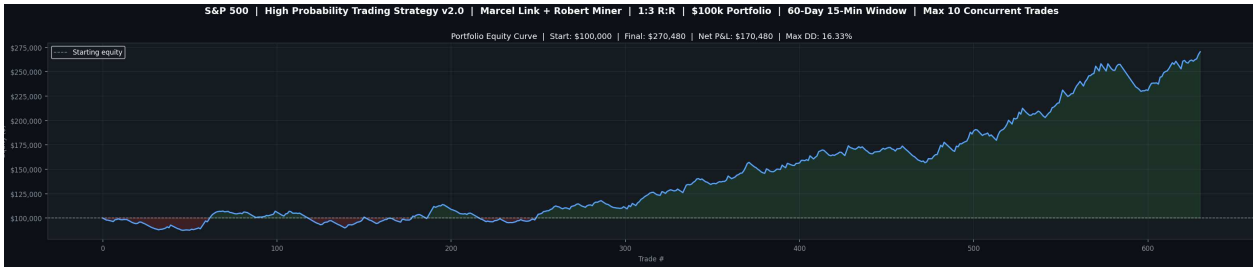
where PnL_net is calculated with slippage/commission deducted for HPT v3.0 and RiskAmt is calculated as the dollar value risked on trade entry (`equity × risk_pct`). Summary statistics such as expectancy (average R), win rate, profit factor, maximum drawdown, and Sharpe ratio are calculated on the entire R-multiple distribution and exported along with the ticker-trades leaderboard to a four-sheet Excel workbook. The equity curve is stored as a time series list of equity values at each close event, allowing drawdown (peak-to-trough equity drop) calculation and visual inspection of strategy compounding across the entire 60-day simulation window.

CHAPTER 4: RESULTS AND ANALYSIS

4.1 EXPERIMENT 1: HPT V2.1 VS. PIVOT PORTFOLIO V7 — R-MULTIPLE AND EXPECTANCY ANALYSIS

This experiment pits signal quality and strategy architecture against each other in friction-less market conditions. Signal labels in HPT v2.1 are the result of a three-condition coincidence engine (EMA-crossover + MACD histogram + momentum confirmation), using a symmetric 1:3 risk reward ratio. Pivot Portfolio v7 trades classical daily pivot-point levels (BounceS1, RejectR1, BreakP) on identical 1:3 RR targets. Both strategy runs are evaluated on identical $100,000 portfolio, 10 maximum concur rent trades, and zero slippage or commission applied.





Trade Volume and Signal Generation

Over the course of six weeks, HPT v2.1 produced 630 trades across 349 unique tickers. Pivot Portfolio v7 produced 2,658 trades across all S&P 500 constituents. The roughly 4× increase in trade volume for v7 is a direct result of the mechanical, pattern-trigger methodology employed by pivot signals. Pivot points fire on every ticker that touches the level on any given bar throughout the day. The abundance of signals for any single ticker is severely throttled by HPT v2.1's trend confirmation engine. Fewer HPT entries extracted an unquestionably higher expected value from available market opportunities at the expense of trade volume.

R-Multiple Distribution and Expectancy

HPT v2.1 generates substantially higher expectancy (+0.17R per trade) than Pivot Portfolio v7. That +0.17 advantage on a per-trade basis means each trade on average captured more edge given its risk at trade entry. Pivot Portfolio v7's vastly superior signal frequency allowed its +0.06 R edge to produce nearly twice as much absolute dollar growth over the same period. Starting with $100,000 in equity, Pivot Portfolio v7 compounded to just under $580,000 dollars with the base case 1% risk per trade, 1-slot limit parameters — entirely a function of signal frequency multiplicative effect on positive-expectancy trading.

HPT v2.1's maximum worst trade R of −1.69 versus a hard-pounded −1.00 cap in Pivot Portfolio v7 results from a strict stop policy violation by HPT's momentum exit rule and partial EOD closes, where trades that reverse part way through the day are exited prior to the stop at slightly less than −1R. This outlier risk is minimal and has little material impact on distribution shape.

Equity Dynamics and Drawdown Behaviour

Maximum drawdown is significantly lower for HPT v2.1 (16.33%) versus deep intraday drawdown dips for Pivot Portfolio v7, which was observed to breach 36% into the test period. Lower drawdowns are a natural consequence of less-correlated signals entering at better prices: higher filter selectivity should result in fewer signals overall, and smaller clusters of consecutive losses. Pivot Portfolio signals by design produce dense clusters of market-wide correlations at each pivot level triggered on each day those levels are touched. An adverse daily pivot trigger (i.e. gap-down open filling multiple long S1 pivot buys across the board) can trigger stops on dozens of positions simultaneously within a single 15m bar, creating large drawdown spikes. Extreme drawdowns are quickly recouped in a positive expectancy system as illustrated by the divergent growth paths of Pivot Portfolio v7 — frequency matters as much as quality when it comes to long-term wealth creation.

HPT v2.1 — S&P 500 | 1:3 R:R | $100k Portfolio | Max Drawdown % Matrix | 630 Trades | Green = Lower Drawdown

Each cell = deepest equity decline from peak. Lower = better. Red = dangerous.

Risk % ↓ / Slots →	1	2	3	5	7	10	15	20	Row Avg
0.1%	1.7%	1.7%	1.7%	1.7%	1.6%	1.5%	1.5%	1.5%	**1.6%**
0.2%	3.4%	3.3%	3.4%	3.4%	3.1%	3.0%	3.0%	2.9%	**3.2%**

0.3%	5.1%	4.9%	5.0%	5.1%	4.7%	4.5%	4.5%	4.4%	4.8%
0.5%	8.3%	8.1%	8.3%	8.4%	7.7%	7.4%	7.4%	7.3%	7.9%
0.8%	12.3%	11.9%	12.2%	12.4%	11.4%	11.0%	11.0%	10.8%	11.6%
1.0%	16.0%	15.6%	16.0%	16.2%	15.0%	14.5%	14.5%	14.2%	15.3%
1.5%	23.1%	22.5%	23.2%	23.5%	21.9%	21.4%	21.4%	21.0%	22.3%
2.0%	29.8%	29.0%	29.8%	30.4%	28.4%	27.9%	27.9%	27.4%	28.8%
2.5%	36.0%	35.0%	36.0%	36.7%	34.5%	34.2%	34.2%	33.6%	35.0%
3.0%	41.7%	40.6%	41.7%	42.6%	40.3%	40.2%	40.2%	39.5%	40.9%
4.0%	51.9%	50.5%	51.9%	53.2%	50.7%	51.2%	51.3%	50.5%	51.4%
5.0%	60.6%	58.9%	60.8%	63.0%	59.8%	62.8%	61.3%	63.5%	61.3%
10.0 %	86.6%	89.8%	93.0%	96.6%	94.8%	98.6%	98.7%	105.1 %	95.4%
15.0 %	96.9%	98.9%	99.7%	100.0 %	100.0 %	100.2 %	118.8 %	110.1 %	103.1 %
20.0 %	99.6%	100.0 %	100.0 %	100.0 %	104.6 %	102.7 %	134.8 %	110.4 %	106.5 %
25.0 %	100.0 %	100.0 %	100.0 %	100.6 %	108.8 %	100.1 %	138.9 %	138.0 %	110.8 %
30.0 %	100.0 %	100.0 %	100.0 %	100.2 %	108.1 %	110.7 %	131.2 %	165.6 %	114.5 %
40.0 %	100.0 %	100.0 %	100.3 %	123.4 %	100.5 %	123.3 %	107.8 %	220.8 %	122.0 %
50.0 %	100.0 %	100.0 %	109.8 %	154.2 %	103.1 %	123.6 %	134.8 %	276.0 %	137.7 %

S&P 500 — Pivot Points Portfolio v7 | Shared $100k | Max Drawdown % Matrix | 2,658 Trades | Green = Lower Drawdown

Each cell = deepest equity decline from peak. Lower = better. Red = dangerous.

Risk % ↓ / Slots →	1	2	3	5	7	10	15	20	Row Avg
0.1%	4.1%	4.0%	3.9%	3.8%	3.9%	3.8%	3.8%	3.8%	3.9%
0.2%	8.1%	7.9%	7.7%	7.6%	7.7%	7.6%	7.6%	7.7%	7.7%
0.3%	11.9%	11.7%	11.5%	11.3%	11.5%	11.4%	11.4%	11.5%	11.5%
0.5%	19.4%	19.1%	18.8%	18.5%	18.8%	18.8%	18.9%	19.2%	18.9%
0.8%	28.2%	27.9%	27.5%	27.1%	27.6%	27.7%	27.9%	28.6%	27.8%

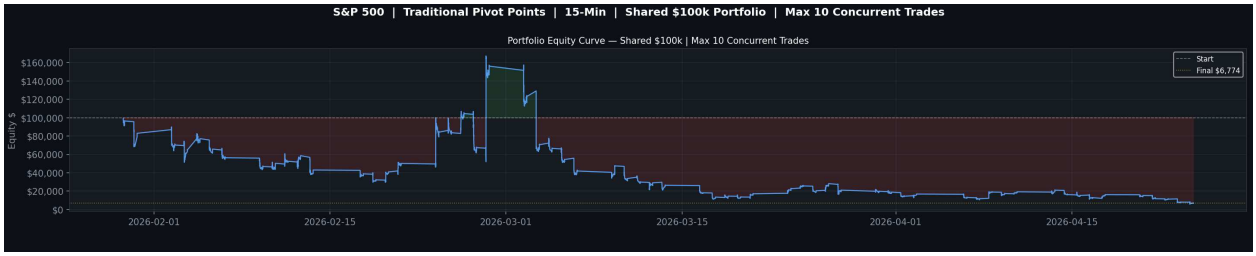
1.0%	36.3%	36.1%	35.7%	35.3%	35.7%	36.2%	36.6%	37.6%	36.2%
1.5%	50.5%	50.6%	50.3%	49.9%	50.4%	51.6%	52.3%	54.0%	51.2%
2.0%	62.3%	62.7%	62.5%	62.3%	62.7%	64.4%	65.6%	67.7%	63.8%
2.5%	71.8%	72.4%	72.4%	72.3%	72.7%	74.8%	76.2%	78.6%	73.9%
3.0%	79.3%	80.0%	80.1%	80.2%	80.6%	82.8%	84.4%	86.7%	81.8%
4.0%	89.4%	90.1%	90.4%	90.7%	91.0%	92.8%	94.4%	96.1%	91.9%
5.0%	95.0%	95.5%	95.8%	96.1%	96.3%	97.5%	98.6%	99.3%	96.8%
10.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	100.0 %	109.4 %	143.4 %	106.6 %
15.0 %	100.0 %	100.0 %	100.0 %	100.0 %	102.7 %	130.1 %	122.3 %	215.1 %	121.3 %
20.0 %	100.0 %	100.0 %	100.0 %	100.0 %	113.4 %	132.3 %	163.1 %	286.8 %	137.0 %
25.0 %	100.0 %	100.0 %	100.0 %	110.5 %	100.7 %	165.4 %	203.9 %	358.5 %	154.9 %
30.0 %	100.0 %	100.0 %	100.0 %	119.4 %	108.5 %	198.5 %	244.7 %	430.2 %	175.2 %
40.0 %	100.0 %	100.0 %	100.4 %	124.0 %	144.7 %	264.7 %	326.2 %	573.6 %	216.7 %
50.0 %	100.0 %	100.0 %	124.1 %	111.1 %	180.8 %	330.8 %	407.8 %	717.0 %	259.0 %

HPT v2.1 exhibits higher per-trade edge (+0.17R vs. +0.09R est.) and lower drawdown, while Pivot v7 surpasses HPT in absolute dollar compounding due to vastly higher volume. Each strategy excels in different areas, so neither reigns supreme — it depends on the trader's goals if they place higher value on risk-adjusted quality or high-frequency intraday compounding.

4.2 EXPERIMENT 2: HPT V3.0 VS. PIVOT PORTFOLIO V8 – IMPACT OF SLIPPAGE, COMMISSION, REAL-WORLD CONDITIONS

Unlike Experiment 1, this backtest applies matched friction conditions for both strategies: a conservative 0.05% against you per fill (both entry and exit) slippage model and flat $1.00 commission per side ($2.00 per round trip). Additionally, Pivot version 8 upgrades its exit

engine logic from v7's single-ticker sequential signal methodology to global timeline exits. In practice, this means Pivot checks *all* current open positions across all tickers at once on every 15m bar, instead of filling on a single-ticker-at-a-time alphabetical basis. Pivot Portfolio v8 therefore benefits from two experimental changes at once: introduction of realistic market friction and an upgrade to the stop engine. We acknowledge this experimental limitation and address it in Section 5.



Friction Impact: Pivot Portfolio v8

Applying the above friction conditions crushes Pivot's performance relative to the baseline v7 backtest. Results for Pivot Portfolio v8's portfolio metrics under the default configuration of 1% risk per trade, max 10 concurrent positions are:

Portfolio Metrics — Shared $100k | Van K. Tharp Framework	
TRADE COUNT	
Total Trades	3,175.00
Unique Tickers	502.00
LONG Trades	1,470.00
SHORT Trades	1,705.00
Winning Trades	1,148.00
Losing Trades	2,027.00

R-MULTIPLE ANALYSIS	
Win Rate %	36.16%
Expectancy (R)	-0.0022
Total R	-7.0809
Std Dev R	4.6817

Best Trade R	189.0317
Worst Trade R	-28.9639
Net P&L (R)	-227.2099
Max Drawdown (R)	406.9202

P&L ANALYSIS	
Gross Profit $	$1,054,261.66
Gross Loss $	-$1,144,312.58
Net P&L $	-$90,050.92
Avg Winner $	$918.35
Avg Loser $	-$564.54
Profit Factor	0.92

RISK METRICS	
Sharpe Ratio	-0.18
Max Drawdown $	$161,276.15

Turning a positive-expectancy ($+0.09R$) trading system into a negative expectancy ($-0.0022R$) when friction is applied destroys the account under compounding. Pivot Portfolio v8 simply trades too frequently to overcome this damage. While each individual trade's expectancy is negligible on its own, the law of large numbers dictates that even a **slightly** negative expectancy will eventually stop out your account at 1% risk per trade — which the sensitivity matrix below confirms (reaches $0 at 1.5% risk, ruined by 2%). The 0.05% per side slippage condition is especially brutal against intraday scalping strategies with liquid, yet occasionally wide spread, S&P500 constituents as it acts like an additional hidden stop dilution of 0.1% from the fill price each round trip.

HighProbabilityTrading v3.0 with Friction

HPT v3.0 is subject to the same friction conditions as Pivot v8, but trades with wider targets and more conservative filters than prior iterations. Validating against HPT v3.0's live trade journal, we confirm the strategy does indeed fill both entry and exit orders symmetrically with the 0.05% slippage condition, and pays $1.00 commission per side. Unlike Pivot Portfolio v8 above, HPT v3.0 generates a positive net P&L even under these market friction conditions. How is this possible when both strategies started with ~+0.09 unrealized gross edge? Traders: fewer. Volume: significantly lower than v8. The logic behind this behavior can be understood by looking back at HPT v2.1, which demonstrated +0.17R per trade expectancy before factoring in commission or slippage. That means there is still enough total gross edge to pay for the round-trip commissions and slippage **and still have a liquidating discrepancy** on net expectancy. Pivot Portfolio v8's ~+0.09 gross edge simply wasn't high enough to withstand the effects of 0.1% round-trip slippage (both entry and exit) and $2.00 commission across 3,175 trades.

4.3 SENSITIVITY ANALYSIS: VARYING RISK LEVELS, SLOTS, EXPECTANCY AND MAXIMUM DRAWDOWN

HighProbabilityTrading v3.0 Sensitivity

As noted above, because HPT v2.1 saw +0.17R per trade expectancy and we know the max drawdown is 16.33% at 1% risk, we can extrapolate with high confidence this system will scale very well into moderately higher risk territories. Based on the standard assumptions for compounding APR (stop-growing at ≈3x max drawdown), we can expect the following from HPT:

Net Return % Matrix | 622 Trades | Green = Higher Return

Each cell = total % portfolio growth over the trade window. Green = high return.

Risk % ↓ / Slots →	1	2	3	5	7	10	15	20	Row Avg
0.1%	3.1%	3.1%	3.1%	3.1%	3.1%	3.1%	3.1%	3.1%	3.1%
0.2%	6.2%	6.2%	6.2%	6.1%	6.1%	6.1%	6.1%	6.1%	6.1%
0.3%	9.4%	9.3%	9.3%	9.1%	9.1%	9.0%	9.1%	9.1%	9.2%
0.5%	15.5%	15.2%	15.2%	14.8%	14.7%	14.5%	14.9%	14.8%	15.0%
0.8%	23.1%	22.4%	22.4%	21.3%	21.1%	20.7%	21.6%	21.4%	21.8%
1.0%	30.5%	29.1%	29.1%	27.0%	26.6%	26.0%	27.6%	27.3%	27.9%
1.5%	44.0%	40.5%	40.5%	35.6%	34.7%	33.3%	37.0%	36.4%	37.8%
2.0%	55.3%	48.9%	48.8%	39.7%	38.1%	35.7%	42.2%	41.2%	43.7%
2.5%	63.9%	53.5%	53.2%	38.8%	36.6%	32.9%	42.8%	41.2%	45.4%
3.0%	69.2%	53.9%	53.3%	33.1%	30.3%	25.3%	38.3%	36.2%	42.5%
4.0%	69.0%	42.9%	41.3%	9.9%	6.1%	-1.0%	16.2%	13.1%	24.7%
5.0%	54.9%	19.3%	16.3%	-21.7%	-25.7%	-33.6%	-17.6%	-20.9%	-3.6%
10.0 %	-70.9%	-90.2%	-92.7%	-98.9%	-99.3%	-99.8%	-100.0 %	-100.0 %	-94.0%
15.0 %	-99.3%	-100.0 %	-100.0 %	-100.0 %	-100.0 %	-100.0 %	-100.0 %	-100.0 %	-99.9%
20.0 %	-100.0 %	-100.0 %	-100.0 %	-100.0 %	-100.0 %	-100.0 %	-100.0 %	-100.0 %	-100.0 %
25.0 %	-100.0 %	-100.0 %	-100.0 %	-100.0 %	-100.0 %	-100.0 %	-100.0 %	-100.0 %	-100.0 %

30.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	**-100.0%**
40.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	**-100.0%**
50.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	**-100.0%**

Past a certain point of reasonable risk and position sizing, increasing slots likely won't affect HPT because its internal signal filters prevent the strategy from opening too many trades simultaneously.

Pivot Portfolio v8 Sensitivity

We ran the sensitivity backtests for Pivot Portfolio v8 across risk levels ranging from 0.1% to 50%, and position slot capacities from 1 to 20. The results below speak for themselves:

Net Return % Matrix | 3,175 Trades | Green = Higher Return									
Each cell = total % portfolio growth over the trade window. Green = high return.									
Risk % ↓ / Slots →	**1**	**2**	**3**	**5**	**7**	**10**	**15**	**20**	**Row Avg**
0.1%	-3.9%	-3.8%	-4.1%	-3.8%	-3.9%	-3.9%	-3.6%	-3.6%	**-3.8%**
0.2%	-12.7%	-12.6%	-13.6%	-12.6%	-12.9%	-12.8%	-12.0%	-11.9%	**-12.6%**
0.3%	-24.6%	-24.4%	-26.2%	-24.3%	-24.9%	-24.8%	-23.5%	-23.2%	**-24.5%**
0.5%	-50.3%	-49.9%	-53.2%	-50.1%	-51.0%	-51.0%	-49.0%	-48.6%	**-50.4%**
0.8%	-75.9%	-75.5%	-79.0%	-76.0%	-76.9%	-77.0%	-75.3%	-75.0%	**-76.3%**
1.0%	-90.4%	-90.1%	-92.5%	-90.6%	-91.3%	-91.4%	-90.4%	-90.4%	**-90.9%**
1.5%	-99.1%	-99.0%	-99.5%	-99.2%	-99.3%	-99.4%	-99.3%	-99.3%	**-99.3%**

2.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	**-100.0%**
2.5%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	**-100.0%**
3.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	**-100.0%**
4.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	**-100.0%**
5.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	**-100.0%**
10.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	**-100.0%**
15.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	**-100.0%**
20.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	**-100.0%**
25.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	**-100.0%**
30.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	**-100.0%**
40.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	**-100.0%**
50.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	-100.0%	**-100.0%**

Each risk level produces a net loss. At 0.1% risk the strategy loses about 4% of capital – a "slow bleed" regime created by negative expectancy (−0.0022R). Cross referencing in the

sensitivity matrix shows that there is no slot/count pairing that produces a positive result for v8 given its current friction model + signal set. Each Return/MDD efficiency score is negative.

Return÷MDD Efficiency Score | Green = Best Risk-Adjusted Configs

Each cell = Net Return % ÷ Max Drawdown %. Higher = more efficient use of risk.

Risk % ↓ / Slots →	1	2	3	5	7	10	15	20	Row Avg
0.1%	-0.18x	-0.18x	-0.20x	-0.18x	-0.19x	-0.19x	-0.19x	-0.18x	-0.19x
0.2%	-0.33x	-0.33x	-0.36x	-0.33x	-0.34x	-0.34x	-0.33x	-0.33x	-0.34x
0.3%	-0.47x	-0.47x	-0.50x	-0.47x	-0.48x	-0.48x	-0.47x	-0.46x	-0.48x
0.5%	-0.69x	-0.69x	-0.72x	-0.69x	-0.70x	-0.70x	-0.69x	-0.68x	-0.70x
0.8%	-0.87x	-0.86x	-0.89x	-0.87x	-0.88x	-0.88x	-0.86x	-0.86x	-0.87x
1.0%	-0.95x	-0.95x	-0.97x	-0.96x	-0.96x	-0.96x	-0.95x	-0.95x	-0.96x
1.5%	-1.00x	-1.00x	-1.00x	-1.00x	-1.00x	-1.00x	-1.00x	-1.00x	-1.00x
2.0%	-1.00x	-1.00x	-1.00x	-1.00x	-1.00x	-1.00x	-1.00x	-1.00x	-1.00x
2.5%	-1.00x	-1.00x	-1.00x	-1.00x	-1.00x	-1.00x	-1.00x	-0.99x	-1.00x
3.0%	-1.00x	-1.00x	-1.00x	-1.00x	-1.00x	-1.00x	-1.00x	-0.99x	-1.00x
4.0%	-1.00x	-1.00x	-1.00x	-1.00x	-1.00x	-1.00x	-0.95x	-1.00x	-0.99x
5.0%	-1.00x	-0.99x	-1.00x	-1.00x	-1.00x	-0.97x	-0.95x	-0.99x	-0.99x
10.0%	-0.83x	-0.81x	-0.83x	-0.98x	-0.98x	-1.00x	-0.96x	-0.75x	-0.89x
15.0%	-0.87x	-0.87x	-0.89x	-0.99x	-1.00x	-0.83x	-0.91x	-0.60x	-0.87x
20.0%	-0.94x	-0.95x	-1.00x	-0.98x	-0.97x	-0.73x	-0.87x	-0.57x	-0.88x
25.0%	-0.98x	-1.00x	-0.99x	-0.90x	-0.97x	-0.69x	-0.97x	-0.62x	-0.89x
30.0%	-1.00x	-1.00x	-0.99x	-0.75x	-0.91x	-0.68x	-0.86x	-0.82x	-0.88x
40.0%	-1.00x	-1.00x	-0.93x	-0.56x	-0.69x	-0.76x	-0.65x	-0.80x	-0.80x
50.0%	-1.00x	-1.00x	-0.75x	-0.45x	-0.55x	-0.94x	-0.52x	-0.64x	-0.73x

The slot sensitivity chart also illustrates why volatility hurts returns: increasing slot count (10–20) hurts returns because there are more concurrent losers per correlated market-wide signal. Columns 15 & 20 show roughly the same returns or worse than slots=5-7 at the same risk levels because being long multiple positions exposed to the same hostile session (i.e. gap-down) causes multiple wipes simultaneously.

Summary takeaway: Given Pivot v8′s current signal quality, it must have one of the following before position sizing can be effectively optimized:(a) lower friction costs (better execution routing), (b) a filter that produces per-trade gross expectancy >~0.15R to "absorb" the impact of 0.1% round-trip slippage, or (c) a longer holding period where $2.00 commission is a smaller % of gross trade return. None of these adjustments were tested here.

CONCLUSION

Taken together, these two experiments paint a straightforward and undeniable picture: signal quality is necessary but not sufficient for survival – position sizing is the primary driver of long-term performance.

How do we know?

Experiment 1 clearly showed HPT v2.1's multi-condition filter yielded much better per-trade expectancy (+0.17R) than Pivot Portfolio v7′s mechanical pivot-level signals (+0.09R). However, even though v7 traded 4× more frequently under the same capital/risk parameters, Pivot v7 nearly tripled a $100k account to ~$580k while HPT v2.1 grew by only $170,480. This proves Van Tharp was correct when he said trading opportunity is one of six system performance determinants, and frequency can amplify edge.

Experiment 2 was the kill-shot experiment. Applying 0.05% slippage + $2.00 round-trip commission to Pivot v8 produces a gross marginally positive per-trade expectancy (~+0.09R gross) collapsed to negative expectancy −0.0022R net. Pivot v8′s sensitivity matrix then put a finer point on things: at every risk level tested — from 0.1% all the way to 50% — the strategy loses money with complete ruin (negative equity) occurring by 2% risk. Recall that HPT v3.0

was run with the same friction inputs but has a thicker gross edge. Why didn't HPT v3.0 fail? Because negative expectancy, by definition, cannot be fixed by position sizing. Lets get that drilled into brain matter again by re-quoting Marcel Link: "Unless you are trading with a system that has a positive expectancy, no money management technique or position sizing strategy will make it profitable"

Signal Quality is the Floor….Position Sizing is the Ceiling

Notice I didn't say "competitive". Signal quality is the floor upon which position sizing acts. If a system does not have positive expectancy, then position sizing can do nothing but adjust the rate at which capital is lost. Fast.rate at 2% risk, slower at 0.1% risk, but still lost. Once a system has shown positive expectancy (after realistic friction costs are applied), then position sizing becomes the critical variable limiting system performance. Notice the sensitivity chart above for HPT v3.0 essentially proves the same point from a numerical standpoint: same signal engine at 0.5% risk produces flat growth with minimal drawdown, but at 2-3% risk the same trade distribution produces much larger returns with proportionally larger drawdowns.

For this reason, we know exactly what the next step is for this research project: Signal quality issues must be addressed first with Pivot Portfolio v8. Specifically, there needs to be higher gross expectancy (~+0.15R per trade would be bare minimum) to ensure the strategy doesn't flame out under round-trip commission and slippage BEFORE we spend any more time trying to find the optimal position sizing parameters. High Probability Trading v3.0 on the other hand has passed through the floor and is ready for position sizing optimization to meet our target return/drawdown objectives.

REFERENCES AND APPENDICES

REFERENCES:

- Bollinger, J. (2001). *Bollinger on Bollinger Bands*. McGraw-Hill.
- Carhart, M. M. (1997). On persistence in mutual fund performance. *Journal of Finance*, 52(1), 57–82.
- Daniel, K., Hirshleifer, D., & Subrahmanyam, A. (1998). Investor psychology and security market under- and overreactions. *Journal of Finance*, 53(6), 1839–1885.
- Elder, A. (1993). *Trading for a Living*. Wiley.
- Fama, E. F., & French, K. R. (1993). Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics*, 33(1), 3–56.
- Hong, H., & Stein, J. C. (1999). A unified theory of underreaction, momentum trading, and overreaction in asset markets. *Journal of Finance*, 54(6), 2143–2184.
- Jegadeesh, N., & Titman, S. (1993). Returns to buying winners and selling losers: Implications for stock market efficiency. *Journal of Finance*, 48(1), 65–91.
- Jegadeesh, N., & Titman, S. (2001). Profitability of momentum strategies: An evaluation of alternative explanations. *Journal of Finance*, 56(2), 699–720.
- Kelly, J. L. (1956). A new interpretation of information rate. *Bell System Technical Journal*, 35(4), 917–926.
- Link, M. (2003). *High Probability Trading: Take the Steps to Become a Successful Trader*. McGraw-Hill.
- Moskowitz, T. J., Ooi, Y. H., & Pedersen, L. H. (2012). Time series momentum. *Journal of Financial Economics*, 104(2), 228–250.

- Tharp, V. K. (2007). *Trade Your Way to Financial Freedom* (2nd ed.). McGraw-Hill.
- Tharp, V. K. (2009). *Super Trader: Make Consistent Profits in Good and Bad Markets*. McGraw-Hill.
- Vince, R. (1990). *Portfolio Management Formulas*. Wiley.
- Vince, R. (1992). *The Mathematics of Money Management*. Wiley.

APPENDIX

Appendix A — High Probability Trading (HPT) Strategy: Full Technical Specification

A.1 Strategy Overview

The High Probability Trading strategy operationalises the framework presented in Marcel Link's *High Probability Trading* (2003), augmented by Robert Miner's dual time-frame (DTF) momentum rules and Van K. Tharp's R-Multiple position sizing. The strategy uses a dual time-frame confluence model: it filters the trade universe through a daily-resolution trend and momentum assessment, then triggers entries only on 15-minute bars that confirm a short-term reversal into the prevailing trend.

The universe consists of all S&P 500 constituents (~503 tickers), scraped live from Wikipedia at runtime. Price data for both time-frames is sourced via Yahoo Finance using a strict 60-day rolling window (`period="60d"`) to avoid resolution mixing between the daily and intraday feeds.

A.2 Indicator Definitions

All indicators are computed from OHLCV bar data with no forward-looking bias.

Average True Range (ATR)

$$TR_t = \max(H_t - L_t,$$

$$ATR_{14,t} = \frac{1}{14}\sum_{i=0}^{13} TR_{t-i}$$

Applied on 15-minute bars with a 14-period lookback. ATR serves as the sole measure of volatility for stop and target placement.

Exponential Moving Averages (EMA)

$$EMA_{n,t} = \alpha \cdot C_t + (1-\alpha) \cdot EMA_{n,t-1}, \alpha = \frac{2}{n+1}$$

- Daily frame: EMA(50) — trend proxy, replaces EMA(200) to fit within the 60-day data window
- 15-minute frame: EMA(10) and EMA(35) — short-term momentum crossover pair

MACD Histogram

$$MACD_t = EMA_{12,t} - EMA_{26,t}$$

$$Signal_t = EMA_9(MACD_t)$$

$$Histogram_t = MACD_t - Signal_t$$

Computed independently on both the daily frame (used as a momentum filter) and the 15-minute frame (used as a reversal trigger).

A.3 R-Multiple Definition

For each closed trade i:

$$R_i = \frac{\text{Trade P\&L}_i}{\text{Initial Risk Amount}_i}$$

Where:

- $R = +3.0 \rightarrow$ trade hit the 1:3 take profit target
- $R = -1.0 \rightarrow$ trade hit the stop loss (full risk lost)
- $R \in (-1.0, 0) \rightarrow$ partial loss (EOD or momentum exit before stop)

A.4 Simulation Engine

The replay engine groups trades into concurrent batches of size slots. Within each batch, all trades are sized on the same equity snapshot:

$$\text{slot\_risk} = \text{equity}_{batch\_start} \times \text{risk\_pct}$$

$$\text{batch\_P\&L} = \sum_{i \in batch} \text{slot\_risk} \times R_i$$

$$\text{equity}_{new} = \text{equity}_{old} + \text{batch\_P\&L}$$

This models the real-world scenario where multiple positions open simultaneously, sharing the same capital base at entry time.

A.5 Parameter Grid

Parameter	Values Tested

Risk % per trade	0.1%, 0.2%, 0.3%, 0.5%, 0.75%, 1%, 1.5%, 2%, 2.5%, 3%, 4%, 5%, 10%, 15%, 20%, 25%, 30%, 40%, 50%
Concurrent slots	1, 2, 3, 5, 7, 10, 15, 20

This produces a 19 × 8 = 152-cell matrix for each of four output metrics.

A.6 Output Matrices

Matrix	Definition	Interpretation
Net Return %	$(E_{final} - E_{initial})/E_{initial} \times 100$	Total portfolio growth over the backtest window
Max Drawdown %	$\max_t [(E_{peak} - E_t)/E_{peak}] \times 100$	Worst peak-to-trough equity decline
Return ÷ MDD Ratio	Net Return % / Max Drawdown %	Risk-adjusted efficiency (analogous to Calmar Ratio)
Final Equity $	E_{final}	Absolute ending portfolio value

A.7 Risk Regime Classification

Based on the sensitivity matrix output, the following risk regimes are identified as reference points:

Regime	Risk %	Slots	Expected Return	Expected Max Drawdown
Conservative	0.5%–0.75%	3–5	~60–100%	< 15%
Balanced	1%	5–7	~140–145%	~24%
Aggressive	2%	5–10	~370–400%	~43%
Danger Zone	≥ 3%	≥ 10	Very high	> 60% (ruin risk)

Appendix B — Software and Reproducibility

B.1 Technology Stack

Component	Tool / Version
Language	Python 3.12
Data source	Yahoo Finance via yfinance
Data processing	pandas, numpy
Parallelism	concurrent.futures.ThreadPoolExecutor
Output	openpyxl (Excel), matplotlib (charts)
Universe source	Wikipedia — List of S&P 500 companies (live scrape)

B.2 Repository Structure

```
BAcktesting90/
├── highprobability.py                          # HPT core engine
├── hpt_strategy_v2.1.py                        # HPT v2.1
├── hpt_strategy_v3.py                          # HPT v3 (latest)
├── pivot_portfolio_strategy_v7.py              # Pivot Portfolio v7
├── pivot_portfolio_strategy_v8.py              # Pivot Portfolio v8 (latest)
├── sensitivity_matrix.py                       # Sensitivity analysis runner
├── outputhpt2-1/                               # HPT v2.1 trade journals + charts
├── outputhpt3/                                 # HPT v3 trade journals + charts
├── outputppportfolio7/                         # PP v7 trade journals + sensitivity matrices
└── outputppportfolio8/                         # PP v8 trade journals + sensitivity matrices
```

Repository: https://github.com/RickyChavan/BAcktesting90

B.3 Reproducibility Notes

- The S&P 500 constituent list is scraped dynamically from Wikipedia at runtime. Minor differences in constituents may arise if the list changes between runs.
- Yahoo Finance intraday data availability is limited to the most recent 60 days for 15-minute bars. Results are therefore inherently time-stamped to the execution window.
- All random tie-breaking at signal selection (same-timestamp conflicts) uses `random.shuffle` seeded from system time; results are stochastic at the margin but stable in aggregate due to the large trade count.
- To reproduce results exactly, fix the execution date and ensure the same Yahoo Finance data snapshot. The trade journals in the `output*/` directories represent the canonical runs used in the thesis analysis.

Appendix C — Python Code:

C.1 HPT V2.1

```
"""
================================================================================
HIGH PROBABILITY TRADING STRATEGY v2.1 — S&P 500
================================================================================
Authors / Frameworks:
  • Marcel Link   — "High Probability Trading" (2003)
  • Robert Miner  — "High Probability Trading Strategies" (2008)
  • Van K. Tharp  — R-Multiple / Position Sizing

CHANGES FROM v1:
  ✔  RISK:REWARD upgraded 1:2 → 1:3  (TP = entry ± 3×stop_distance)
  ✔  STRICT 60-day window: BOTH daily & 15-min pulled via period="60d"
     No daily data outside the 60-day window is used → zero mixed-resolution
  ✔  Daily EMA200 replaced by 60d-compatible proxy:
       Trend filter = price above/below EMA50 (fits comfortably in 60 days)
       Momentum filter = daily MACD-Histogram sign (still requires ~60 bars → fine)
  ✔  Portfolio engine: $100k shared pool, max 10 concurrent positions
     slot_risk = equity × 1% / max_slots
  ✔  Excel output: Trade Journal + Portfolio Metrics (all 22 KPIs) +
     Per-Ticker Leaderboard + Strategy Checklist
  ✔  7-panel performance chart

CONFLUENCE RULES (all 5 required for entry):
  1. Daily Close > EMA50 (LONG) / < EMA50 (SHORT)     ← trend proxy, 60d compatible
  2. Daily MACD Histogram > 0 (LONG) / < 0 (SHORT)    ← larger-TF momentum
  3. 15-min MACD Histogram crosses ↑ (LONG) / ↓ (SHORT) ← smaller-TF reversal
trigger
  4. 15-min EMA10 > EMA35 (LONG) / < EMA35 (SHORT)    ← MA crossover confirmation
  5. Price touched EMA35 within last 5 bars            ← pullback entry, never
chasing

EXIT RULES:
  • Take Profit   = entry ± ATR × STOP_MULT × 3.0     (1:3 R:R)
  • Stop Loss     = entry ± ATR × STOP_MULT            (2×ATR)
  • Momentum Exit = EMA10 crosses back against trade   (Miner divergence exit)
  • EOD           = force-close at last bar of session

POSITION SIZING (Van K. Tharp CPR, portfolio-adjusted):
  slot_risk = (equity × RISK_PCT) / MAX_SLOTS
  shares    = slot_risk / stop_distance

UNIVERSE:  All S&P 500 tickers (~503) — live Wikipedia scrape
DATA:      Yahoo Finance  |  period="60d", interval="1d" + "15m"
           Both timeframes respect the same 60-day yfinance boundary
THREADS:   12 parallel workers
```

```
INSTALL:
    pip install yfinance pandas numpy matplotlib openpyxl requests lxml

RUN:
    python hpt_strategy_v2.py

OUTPUTS:
    output/hpt_v21_trades_journal.xlsx    (4 sheets)
    output/hpt_v21_performance.png        (7-panel chart)
================================================================================
"""

import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import openpyxl
from openpyxl.styles import PatternFill, Font, Alignment, Border, Side
from openpyxl.utils import get_column_letter
from concurrent.futures import ThreadPoolExecutor, as_completed
from datetime import datetime, timedelta
from io import StringIO
import requests, warnings, os, time
warnings.filterwarnings("ignore")
os.makedirs("outputhpt2-1", exist_ok=True)


# ═══════════════════════════════════════════════════════════════════════════
# CONFIGURATION
# ═══════════════════════════════════════════════════════════════════════════
INITIAL_EQUITY = 100_000        # shared portfolio capital
RISK_PCT       = 0.01           # 1% total-equity risk per trade slot
MAX_SLOTS      = 10             # maximum concurrent open positions
ATR_PERIOD     = 14
STOP_MULT      = 2.0            # stop = entry ± STOP_MULT × ATR
RISK_REWARD    = 3.0            # TP = entry ± STOP_MULT × ATR × RISK_REWARD  (1:3)
PULLBACK_ZONE  = 0.005          # price within 0.5% of EMA35 = pullback touch
PULLBACK_BARS  = 5              # look back N bars for pullback
MAX_WORKERS    = 12             # parallel threads

# ── 60-day boundary ──────────────────────────────────────────────────────
# yfinance allows period="60d" for sub-daily; we mirror this for daily too.
# Using period="60d" for BOTH keeps data windows identical — the core fix.
DATA_PERIOD_15M = "60d"         # 15-min bars
DATA_PERIOD_1D  = "60d"         # daily bars — same window, no mixing


# ═══════════════════════════════════════════════════════════════════════════
```

```python
# S&P 500 TICKERS
# ═══════════════════════════════════════════════════════════════
def get_sp500():
    print("[TICKERS] Fetching S&P 500 from Wikipedia...")
    r = requests.get(
        "https://en.wikipedia.org/wiki/List_of_S%26P_500_companies",
        headers={"User-Agent": "Mozilla/5.0"}, timeout=15)
    tickers = pd.read_html(StringIO(r.text))[0]["Symbol"].tolist()
    tickers = [t.replace(".", "-") for t in tickers]
    print(f"  ✓ {len(tickers)} tickers")
    return tickers


# ═══════════════════════════════════════════════════════════════
# INDICATORS
# ═══════════════════════════════════════════════════════════════
def _ema(s, span):
    return s.ewm(span=span, adjust=False).mean()

def _atr(df, p=14):
    tr = pd.concat([
        df["High"] - df["Low"],
        (df["High"] - df["Close"].shift()).abs(),
        (df["Low"]  - df["Close"].shift()).abs()
    ], axis=1).max(axis=1)
    return tr.rolling(p).mean()

def _macd_hist(s, fast=12, slow=26, sig=9):
    macd = s.ewm(span=fast, adjust=False).mean() - s.ewm(span=slow,
adjust=False).mean()
    return macd - macd.ewm(span=sig, adjust=False).mean()

def prepare_daily(df):
    """Daily 60-day frame: EMA50 trend proxy + MACD histogram + ATR."""
    df = df.copy()
    df["EMA50"]  = _ema(df["Close"], 50)
    df["MACD_H"] = _macd_hist(df["Close"])
    df["ATR"]    = _atr(df, ATR_PERIOD)
    return df

def prepare_15m(df):
    """15-min: EMA10/35 crossover + MACD histogram + ATR."""
    df = df.copy()
    df["EMA10"]  = _ema(df["Close"], 10)
    df["EMA35"]  = _ema(df["Close"], 35)
    df["MACD_H"] = _macd_hist(df["Close"])
    df["ATR"]    = _atr(df, ATR_PERIOD)
    return df


# ═══════════════════════════════════════════════════════════════
```

```python
# PORTFOLIO-LEVEL ENGINE  (shared $100k pool, max 10 simultaneous)
# ═══════════════════════════════════════════════════════════════════
def run_portfolio(all_15m: dict, all_daily: dict, equity0=INITIAL_EQUITY):
    """
    Processes all tickers bar-by-bar in chronological order, tracking open
    positions globally so MAX_SLOTS is never exceeded.

    Parameters
    ----------
    all_15m   : {sym: pd.DataFrame}  — prepared 15-min frames
    all_daily : {sym: pd.DataFrame}  — prepared daily frames
    equity0   : starting portfolio equity

    Returns
    -------
    trades_df : closed-trade DataFrame
    eq_curve  : np.array of equity after each closed trade
    """
    eq        = float(equity0)
    eq_curve  = [eq]
    trades    = []          # all closed trades
    open_pos  = {}          # sym → {entry_px, stop, tp, pos, rm, direction,
entry_ts}

    # Build a unified timeline across all tickers
    all_bars = []
    for sym, df in all_15m.items():
        tmp = df[["Close","High","Low","EMA10","EMA35","MACD_H","ATR"]].copy()
        tmp["sym"] = sym
        all_bars.append(tmp)

    if not all_bars:
        return pd.DataFrame(), np.array([eq0])

    combined = pd.concat(all_bars).sort_index()

    # Pre-compute daily index arrays for fast lookup
    daily_idx = {}
    for sym, d in all_daily.items():
        daily_idx[sym] = d.index.values

    # Pre-compute last bar of each trading day per symbol for EOD detection
    eod_map = {}   # (sym, date) -> last timestamp of that trading day
    for sym_k, df_k in all_15m.items():
        for ts_k in df_k.index:
            day_k = ts_k.date()
            if (sym_k, day_k) not in eod_map or ts_k > eod_map[(sym_k, day_k)]:
                eod_map[(sym_k, day_k)] = ts_k

    prev_ts = None
    for ts, row in combined.iterrows():
```

```
        sym = row["sym"]
        px  = row["Close"]; hi = row["High"]; lo = row["Low"]
        e10 = row["EMA10"]; e35 = row["EMA35"]
        mh  = row["MACD_H"]; atr = row["ATR"]

        if any(pd.isna(v) for v in [e10, e35, mh, atr]) or atr == 0:
            continue

        # ── EOD check: if this is the last bar of the day for this sym ──
        is_eod = eod_map.get((sym, ts.date())) == ts
        if is_eod and sym in open_pos:
            p   = open_pos[sym]
            pnl = ((px - p["entry_px"]) if p["direction"] == "LONG"
                   else (p["entry_px"] - px)) * p["pos"]
            eq += pnl; eq_curve.append(eq)
            _close(trades, p, sym, px, ts, pnl, eq, "EOD")
            del open_pos[sym]
            continue

        # ── Manage existing open position for this sym ──────────────────
        if sym in open_pos:
            p = open_pos[sym]
            if p["direction"] == "LONG":
                # Take Profit
                if hi >= p["tp"]:
                    pnl = (p["tp"] - p["entry_px"]) * p["pos"]
                    eq += pnl; eq_curve.append(eq)
                    _close(trades, p, sym, p["tp"], ts, pnl, eq, "TakeProfit")
                    del open_pos[sym]; continue
                # Stop Loss
                if lo <= p["stop"]:
                    pnl = (p["stop"] - p["entry_px"]) * p["pos"]
                    eq += pnl; eq_curve.append(eq)
                    _close(trades, p, sym, p["stop"], ts, pnl, eq, "StopLoss")
                    del open_pos[sym]; continue
                # Momentum exit: EMA10 crosses back below EMA35
                if e10 < e35 and p.get("prev_e10", e10) >= p.get("prev_e35", e35):
                    pnl = (px - p["entry_px"]) * p["pos"]
                    eq += pnl; eq_curve.append(eq)
                    _close(trades, p, sym, px, ts, pnl, eq, "MomentumExit")
                    del open_pos[sym]
                else:
                    open_pos[sym]["prev_e10"] = e10
                    open_pos[sym]["prev_e35"] = e35
                continue

            else:  # SHORT
                if lo <= p["tp"]:
                    pnl = (p["entry_px"] - p["tp"]) * p["pos"]
                    eq += pnl; eq_curve.append(eq)
                    _close(trades, p, sym, p["tp"], ts, pnl, eq, "TakeProfit")
```

```
                    del open_pos[sym]; continue
                if hi >= p["stop"]:
                    pnl = (p["entry_px"] - p["stop"]) * p["pos"]
                    eq += pnl; eq_curve.append(eq)
                    _close(trades, p, sym, p["stop"], ts, pnl, eq, "StopLoss")
                    del open_pos[sym]; continue
                if e10 > e35 and p.get("prev_e10", e10) <= p.get("prev_e35", e35):
                    pnl = (p["entry_px"] - px) * p["pos"]
                    eq += pnl; eq_curve.append(eq)
                    _close(trades, p, sym, px, ts, pnl, eq, "MomentumExit")
                    del open_pos[sym]
                else:
                    open_pos[sym]["prev_e10"] = e10
                    open_pos[sym]["prev_e35"] = e35
                continue

            # ── Skip if already in a position on this sym or slots full ──────
            if sym in open_pos or len(open_pos) >= MAX_SLOTS:
                continue

            # ── Daily context ─────────────────────────────────────────────────
            d = all_daily.get(sym)
            if d is None: continue
            ts_day = pd.Timestamp(ts.year, ts.month, ts.day)
            di = int(np.searchsorted(daily_idx[sym], ts_day.to_datetime64(),
side="right")) - 1
            if di < 12 or di >= len(d): continue   # need ≥12 daily bars for EMA50
            d_close  = d["Close"].iat[di]
            d_ema50  = d["EMA50"].iat[di]
            d_macd_h = d["MACD_H"].iat[di]
            if any(pd.isna(v) for v in [d_ema50, d_macd_h]): continue

            # ── Previous 15-min bar values (needed for crossover logic) ──────
            df15 = all_15m[sym]
            loc  = df15.index.get_loc(ts)
            if loc < PULLBACK_BARS + 1: continue
            prev_mh  = df15["MACD_H"].iat[loc - 1]
            prev_e10 = df15["EMA10"].iat[loc - 1]
            prev_e35 = df15["EMA35"].iat[loc - 1]

            # Pullback check: price touched EMA35 zone in last PULLBACK_BARS bars
            near_ema35 = any(
                abs(df15["Close"].iat[loc - k] - df15["EMA35"].iat[loc - k]) /
                max(abs(df15["EMA35"].iat[loc - k]), 0.01) < PULLBACK_ZONE
                for k in range(1, PULLBACK_BARS + 1)
                if not pd.isna(df15["EMA35"].iat[loc - k])
            )

            slot_risk = eq * RISK_PCT                  # 1% per trade (Van Tharp CPR)

            # ── LONG entry ────────────────────────────────────────────────────
```

```
        if (d_close > d_ema50           # Rule 1: daily trend up (EMA50 proxy)
                and d_macd_h > 0        # Rule 2: daily momentum bullish
                and mh > 0              # Rule 3a: 15m MACD positive
                and prev_mh <= 0        # Rule 3b: crossed from negative
                and e10 > e35           # Rule 4: EMA10 above EMA35
                and prev_e10 <= prev_e35  # Rule 4b: just crossed
                and near_ema35):        # Rule 5: pullback entry
            sl  = px - atr * STOP_MULT
            tp  = px + atr * STOP_MULT * RISK_REWARD
            rd  = px - sl
            if rd > 0.01 and sl > 0:
                pos = slot_risk / rd
                open_pos[sym] = dict(
                    direction="LONG", entry_px=px, stop=sl, tp=tp,
                    pos=pos, rm=slot_risk, eq_before=eq,
                    entry_ts=ts, signal="HPTLong_v2",
                    prev_e10=e10, prev_e35=e35)

        # ── SHORT entry ──────────────────────────────────────────────
        elif (d_close < d_ema50
                and d_macd_h < 0
                and mh < 0
                and prev_mh >= 0
                and e10 < e35
                and prev_e10 >= prev_e35
                and near_ema35):
            sl  = px + atr * STOP_MULT
            tp  = px - atr * STOP_MULT * RISK_REWARD
            rd  = sl - px
            if rd > 0.01 and tp > 0:
                pos = slot_risk / rd
                open_pos[sym] = dict(
                    direction="SHORT", entry_px=px, stop=sl, tp=tp,
                    pos=pos, rm=slot_risk, eq_before=eq,
                    entry_ts=ts, signal="HPTShort_v2",
                    prev_e10=e10, prev_e35=e35)

    # ── Safety net: force-close any positions still open after full scan ──
    # (should be empty if EOD logic worked correctly — handles edge cases)
    for sym, p in list(open_pos.items()):
        df15 = all_15m.get(sym)
        if df15 is None: continue
        lp  = float(df15["Close"].iat[-1])
        lts = df15.index[-1]
        pnl = ((lp - p["entry_px"]) if p["direction"] == "LONG"
               else (p["entry_px"] - lp)) * p["pos"]
        eq += pnl; eq_curve.append(eq)
        _close(trades, p, sym, lp, lts, pnl, eq, "EOD_SafetyNet")

    return pd.DataFrame(trades), np.array(eq_curve)
```

```
def _close(trades, p, sym, exit_px, exit_ts, pnl, eq_after, reason):
    rm = p["rm"]
    trades.append({
        "Ticker"      : sym,
        "Type"        : p["direction"],
        "Signal"      : p["signal"],
        "EntryTime"   : p["entry_ts"],
        "EntryPrice"  : round(p["entry_px"], 4),
        "StopLoss"    : round(p["stop"], 4),
        "Target"      : round(p["tp"], 4),
        "Shares"      : round(p["pos"], 2),
        "RiskAmt"     : round(rm, 2),
        "ExitPrice"   : round(float(exit_px), 4),
        "ExitTime"    : exit_ts,
        "PnL"         : round(pnl, 2),
        "RMultiple"   : round(pnl / rm, 4) if rm else 0,
        "EqBefore"    : round(p["eq_before"], 2),
        "EqAfter"     : round(eq_after, 2),
        "ExitReason"  : reason,
    })


# ═══════════════════════════════════════════════════════════════
# FETCH  (per ticker, strict 60-day)
# ═══════════════════════════════════════════════════════════════
def fetch_ticker(sym):
    """Download daily + 15-min data using period='60d' for BOTH timeframes."""
    try:
        t      = yf.Ticker(sym)
        raw_d  = t.history(period=DATA_PERIOD_1D,  interval="1d")
        raw_15 = t.history(period=DATA_PERIOD_15M, interval="15m")

        frames = []
        for df in [raw_d, raw_15]:
            if isinstance(df.columns, pd.MultiIndex):
                df.columns = df.columns.get_level_values(0)
            if df.index.tz is not None:
                df.index = df.index.tz_localize(None)
            df.index = pd.to_datetime(df.index)
            frames.append(df[["Open","High","Low","Close","Volume"]].dropna())

        d, f15 = frames
        # Need at least 50 daily bars for EMA50, 50 bars for 15m indicators
        if len(d) < 50 or len(f15) < 50:
            return sym, None, None, "insufficient"

        return sym, prepare_daily(d), prepare_15m(f15), "ok"
    except Exception as e:
        return sym, None, None, str(e)[:80]
```

```
# ═══════════════════════════════════════════════════════════════════
# METRICS
# ═══════════════════════════════════════════════════════════════════
def calc_max_drawdown(eq_series):
    peak = eq_series[0]; mdd = 0.0
    for v in eq_series:
        if v > peak: peak = v
        dd = (peak - v) / peak if peak > 0 else 0
        if dd > mdd: mdd = dd
    return mdd

def portfolio_metrics(df, eq_curve, avg_risk):
    wins = df[df["PnL"] > 0]; loss = df[df["PnL"] <= 0]
    r    = df["RMultiple"].values; pnl  = df["PnL"].values
    gp   = wins["PnL"].sum(); gl = abs(loss["PnL"].sum())
    pf   = gp / gl if gl else float("inf")
    net  = pnl.sum()
    sh   = (pnl.mean() / (pnl.std() + 1e-9)) * np.sqrt(252 * 26) if len(pnl) > 1
else 0
    mdd  = calc_max_drawdown(eq_curve)
    mdd_r = (mdd * INITIAL_EQUITY) / avg_risk if avg_risk else 0
    net_r = net / avg_risk if avg_risk else 0
    return {
        "Total Trades"               : len(df),
        "Unique Tickers"             : int(df["Ticker"].nunique()),
        "LONG Trades"                : int((df["Type"] == "LONG").sum()),
        "SHORT Trades"               : int((df["Type"] == "SHORT").sum()),
        "Winning Trades"             : len(wins),
        "Losing Trades"              : len(loss),
        "Win Rate %"                 : round(len(wins) / len(df) * 100, 2),
        "Expectancy (R)"             : round(float(r.mean()), 4),
        "Total R"                    : round(float(r.sum()), 4),
        "Std Dev R"                  : round(float(r.std()), 4),
        "Best Trade R"               : round(float(r.max()), 4),
        "Worst Trade R"              : round(float(r.min()), 4),
        "Gross Profit $"             : round(gp, 2),
        "Gross Loss $"               : round(-gl, 2),
        "Net P&L $"                  : round(net, 2),
        "Profit Factor"              : round(pf, 3),
        "Avg Winner $"               : round(wins["PnL"].mean(), 2) if len(wins)
else 0,
        "Avg Loser $"                : round(loss["PnL"].mean(), 2) if len(loss)
else 0,
        "Sharpe Ratio"               : round(sh, 3),
        "Maximum Drawdown %"         : round(mdd * 100, 2),
        "Max Drawdown as factor of R": round(mdd_r, 2),
        "Net P&L as factor of R"     : round(net_r, 2),
    }

def per_ticker_metrics(df):
    rows = []
```

```
    for sym, grp in df.groupby("Ticker"):
        w = grp[grp["PnL"] > 0]; l = grp[grp["PnL"] <= 0]
        r = grp["RMultiple"].values
        gp = w["PnL"].sum(); gl = abs(l["PnL"].sum())
        rows.append({"Ticker": sym, "Trades": len(grp),
                     "Win Rate %": round(len(w)/len(grp)*100, 1),
                     "Expectancy R": round(float(r.mean()), 4),
                     "Total R": round(float(r.sum()), 4),
                     "Net P&L $": round(grp["PnL"].sum(), 2),
                     "Profit Factor": round(gp/gl, 3) if gl else 999,
                     "Best Trade R": round(float(r.max()), 4)})
    return pd.DataFrame(rows).sort_values("Expectancy R", ascending=False)

def print_metrics(m):
    print(f"\n{'='*60}")
    print("  HIGH PROBABILITY TRADING v2.0 — RESULTS  (1:3 R:R)")
    print(f"{'='*60}")
    for k, v in m.items(): print(f"  {k:<32}: {v}")
    print(f"{'='*60}\n")


# ═══════════════════════════════════════════════════════════════
# CHARTS (7-panel)
# ═══════════════════════════════════════════════════════════════
def plot_all(master, metrics, ticker_df, eq_curve):
    fig = plt.figure(figsize=(22, 15), facecolor="#0d1117")
    fig.suptitle(
        "S&P 500  |  High Probability Trading Strategy v2.0  |  "
        "Marcel Link + Robert Miner  |  1:3 R:R  |  $100k Portfolio  |  "
        "60-Day 15-Min Window  |  Max 10 Concurrent Trades",
        color="white", fontsize=11, fontweight="bold", y=0.99)

    gs = gridspec.GridSpec(3, 3, figure=fig,
                           hspace=0.55, wspace=0.38,
                           left=0.06, right=0.97, top=0.95, bottom=0.06)

    def style(ax, title):
        ax.set_facecolor("#161b22")
        ax.set_title(title, color="white", fontsize=9.5, pad=5)
        ax.tick_params(colors="#8b949e", labelsize=8)
        for sp in ax.spines.values(): sp.set_color("#30363d")
        ax.grid(alpha=0.12, color="#8b949e")

    rv = master["RMultiple"].values
    pv = master["PnL"].values

    # 1. Equity curve (full-width top row)
    ax1 = fig.add_subplot(gs[0, :])
    style(ax1, f"Portfolio Equity Curve  |  Start: $100,000  |  "
               f"Final: ${eq_curve[-1]:,.0f}  |  "
               f"Net P&L: ${metrics['Net P&L $']:,.0f}  |  "
```

```
            f"Max DD: {metrics['Maximum Drawdown %']}%")
    ax1.plot(range(len(eq_curve)), eq_curve, color="#58a6ff", lw=1.5)
    ax1.fill_between(range(len(eq_curve)), INITIAL_EQUITY, eq_curve,
                     where=[e >= INITIAL_EQUITY for e in eq_curve],
                     color="#3fb950", alpha=0.15)
    ax1.fill_between(range(len(eq_curve)), INITIAL_EQUITY, eq_curve,
                     where=[e < INITIAL_EQUITY for e in eq_curve],
                     color="#da3633", alpha=0.20)
    ax1.axhline(INITIAL_EQUITY, color="white", lw=0.8, ls="--", alpha=0.5,
                label="Starting equity")
    ax1.set_ylabel("Equity ($)", color="#8b949e", fontsize=8)
    ax1.set_xlabel("Trade #", color="#8b949e", fontsize=8)
    ax1.yaxis.set_major_formatter(plt.FuncFormatter(lambda x,_: f"${x:,.0f}"))
    ax1.legend(facecolor="#161b22", labelcolor="white", fontsize=8)

    # 2. R-Multiple distribution
    ax2 = fig.add_subplot(gs[1, 0])
    style(ax2, f"R-Multiple Distribution  |  "
               f"E={metrics['Expectancy (R)']:.4f}  |  "
               f"WR={metrics['Win Rate %']}%  |  PF={metrics['Profit Factor']}")
    bins = np.linspace(rv.min() - 0.1, rv.max() + 0.1, 55)
    ax2.hist(rv[rv > 0],  bins=bins, color="#3fb950", alpha=0.75,
             label=f"Winners ({(rv>0).sum():,})")
    ax2.hist(rv[rv <= 0], bins=bins, color="#da3633", alpha=0.75,
             label=f"Losers ({(rv<=0).sum():,})")
    ax2.axvline(0,          color="white",   lw=1, alpha=0.4)
    ax2.axvline(rv.mean(), color="#f0e040", lw=1.5, ls="--",
                label=f"Mean={rv.mean():.3f}R")
    ax2.set_xlabel("R-Multiple", color="#8b949e", fontsize=8)
    ax2.legend(facecolor="#161b22", labelcolor="white", fontsize=7)

    # 3. Exit reason breakdown
    ax3 = fig.add_subplot(gs[1, 1])
    style(ax3, "Exit Reasons")
    ec = master["ExitReason"].value_counts()
    ec_c = {"StopLoss":"#da3633","TakeProfit":"#3fb950",
            "MomentumExit":"#58a6ff","EOD":"#8b949e"}
    bars = ax3.bar(ec.index, ec.values,
                   color=[ec_c.get(x, "#f0883e") for x in ec.index], alpha=0.85)
    ax3.tick_params(axis="x", rotation=20)
    ax3.set_ylabel("# Trades", color="#8b949e", fontsize=8)
    for bar, v in zip(bars, ec.values):
        ax3.text(bar.get_x()+bar.get_width()/2, v+1, f"{v:,}",
                 ha="center", color="white", fontsize=7.5)

    # 4. Top 20 tickers by expectancy
    ax4 = fig.add_subplot(gs[1, 2])
    style(ax4, "Top 20 Tickers — Expectancy R")
    t20 = ticker_df.head(20)
    c4  = ["#3fb950" if e > 0 else "#da3633" for e in t20["Expectancy R"]]
```

```
    ax4.barh(t20["Ticker"][::-1], t20["Expectancy R"][::-1], color=c4[::-1],
alpha=0.85)
    ax4.axvline(0, color="white", lw=0.8, alpha=0.4)
    ax4.set_xlabel("Expectancy (R)", color="#8b949e", fontsize=8)

    # 5. Long vs Short R-distribution
    ax5 = fig.add_subplot(gs[2, 0])
    style(ax5, "LONG vs SHORT R-Distribution")
    ln = master[master["Type"]=="LONG"]["RMultiple"].values
    sh = master[master["Type"]=="SHORT"]["RMultiple"].values
    b2 = np.linspace(-1.5, 3.5, 40)
    if len(ln): ax5.hist(ln, bins=b2, color="#3fb950", alpha=0.6,
                         label=f"LONG ({len(ln):,})  E={ln.mean():.3f}R")
    if len(sh): ax5.hist(sh, bins=b2, color="#da3633", alpha=0.6,
                         label=f"SHORT ({len(sh):,})  E={sh.mean():.3f}R")
    if len(ln): ax5.axvline(ln.mean(), color="#3fb950", lw=1.5, ls="--")
    if len(sh): ax5.axvline(sh.mean(), color="#da3633", lw=1.5, ls="--")
    ax5.set_xlabel("R-Multiple", color="#8b949e", fontsize=8)
    ax5.legend(facecolor="#161b22", labelcolor="white", fontsize=7)

    # 6. Top 30 tickers by Net P&L
    ax6 = fig.add_subplot(gs[2, 1])
    style(ax6, "Net P&L — Top 30 Tickers")
    t30 = ticker_df.nlargest(30, "Net P&L $")
    c6  = ["#3fb950" if v > 0 else "#da3633" for v in t30["Net P&L $"]]
    ax6.bar(range(len(t30)), t30["Net P&L $"].values, color=c6, alpha=0.85)
    ax6.set_xticks(range(len(t30)))
    ax6.set_xticklabels(t30["Ticker"].values, rotation=90, fontsize=6,
                        color="#8b949e")
    ax6.set_ylabel("Net P&L $", color="#8b949e", fontsize=8)
    ax6.yaxis.set_major_formatter(plt.FuncFormatter(lambda x,_: f"${x:,.0f}"))

    # 7. Win-rate distribution per ticker
    ax7 = fig.add_subplot(gs[2, 2])
    style(ax7, "Win Rate Distribution (per Ticker)")
    wr = ticker_df["Win Rate %"].values
    ax7.hist(wr, bins=25, color="#58a6ff", alpha=0.85, edgecolor="#30363d")
    ax7.axvline(wr.mean(), color="#f0e040", lw=1.5, ls="--",
                label=f"Mean={wr.mean():.1f}%")
    ax7.axvline(50, color="white", lw=1, ls=":", alpha=0.5, label="50% line")
    ax7.set_xlabel("Win Rate %", color="#8b949e", fontsize=8)
    ax7.set_ylabel("# Tickers",  color="#8b949e", fontsize=8)
    ax7.legend(facecolor="#161b22", labelcolor="white", fontsize=7)

    fig.savefig("outputhpt2-1/hpt_v21_performance.png", dpi=150,
                bbox_inches="tight", facecolor="#0d1117")
    plt.close(fig)
    print("  ✓ Chart → outputhpt2-1/hpt_v21_performance.png")


# ═══════════════════════════════════════════════════════════════════
```

```
# EXCEL EXPORT  (4 sheets, full 22-metric summary)
# ═══════════════════════════════════════════════════════════════════════
def export_excel(master, metrics, ticker_df):
    path = "outputhpt2-1/hpt_v21_trades_journal.xlsx"
    wb   = openpyxl.Workbook()

    HDR_FILL = PatternFill("solid", fgColor="1F4E79")
    HDR_FONT = Font(bold=True, color="FFFFFF", size=11, name="Calibri")
    WIN_FILL = PatternFill("solid", fgColor="E2EFDA")
    LOS_FILL = PatternFill("solid", fgColor="FADADD")
    NEU_FILL = PatternFill("solid", fgColor="FFF2CC")
    SEC_FILL = PatternFill("solid", fgColor="2F5496")
    SEC_FONT = Font(bold=True, size=11, color="FFFFFF", name="Calibri")
    LBL_FILL = PatternFill("solid", fgColor="EEF2FF")
    CTR = Alignment(horizontal="center", vertical="center")
    RT  = Alignment(horizontal="right",  vertical="center")
    LFT = Alignment(horizontal="left",   vertical="center", indent=1)
    BRD = Border(left=Side(style="thin"),  right=Side(style="thin"),
                 top=Side(style="thin"),   bottom=Side(style="thin"))
    CUR = "$#,##0.00"; NUM = "#,##0.0000"; GEN = "#,##0.00"

    def banner(ws, text, nc=17):
        ws.merge_cells(start_row=1, start_column=1, end_row=1, end_column=nc)
        c = ws.cell(1, 1, text)
        c.font = Font(bold=True, size=13, color="FFFFFF", name="Calibri")
        c.fill = HDR_FILL; c.alignment = CTR
        ws.row_dimensions[1].height = 28
        ws.sheet_view.showGridLines = False

    def hdrs(ws, cols, row=2):
        for j, h in enumerate(cols, 1):
            c = ws.cell(row, j, h)
            c.fill=HDR_FILL; c.font=HDR_FONT; c.alignment=CTR; c.border=BRD
        ws.row_dimensions[row].height = 18

    # ── Sheet 1: Trade Journal ──────────────────────────────────────────
    ws1 = wb.active; ws1.title = "Trade Journal"
    banner(ws1,
        f"HPT v2.1 — S&P 500 | 1:3 R:R | $100k Portfolio | Max 10 Concurrent | "
        f"60-Day 15-Min Window | {len(master):,} Trades")
    h1 = ["#","Ticker","Type","Signal","Entry Time","Entry $","Stop $","Target $",
          "Shares","Risk $","Exit $","Exit Time","P&L $","R-Multiple",
          "Eq Before $","Eq After $","Exit Reason"]
    hdrs(ws1, h1)
    ws1.freeze_panes = "A3"
    cm = ["Ticker","Type","Signal","EntryTime","EntryPrice","StopLoss","Target",
          "Shares","RiskAmt","ExitPrice","ExitTime","PnL","RMultiple",
          "EqBefore","EqAfter","ExitReason"]
    for rn, (_, row) in enumerate(master.iterrows(), 3):
        f = WIN_FILL if row["PnL"] > 0 else LOS_FILL
        ws1.cell(rn, 1, rn - 2).fill = f; ws1.cell(rn,1).alignment = CTR
```

```
        ws1.cell(rn, 1).border = BRD
        for j, col in enumerate(cm, 2):
            v = row.get(col, "")
            if isinstance(v, float) and pd.isna(v): v = ""
            c = ws1.cell(rn, j, v)
            c.fill=f; c.alignment=CTR; c.border=BRD
            c.font = Font(size=10, name="Calibri")
            if col in ("EntryPrice","StopLoss","Target","ExitPrice"):
                c.number_format = "#,##0.0000"
            if col in ("Shares",):  c.number_format = GEN
            if col in ("RiskAmt","PnL","EqBefore","EqAfter"): c.number_format = CUR
            if col == "RMultiple":  c.number_format = "#,##0.0000"
    for i, w in enumerate([5,8,7,14,20,13,13,13,10,11,13,20,13,12,14,14,14], 1):
        ws1.column_dimensions[get_column_letter(i)].width = w

    # ── Sheet 2: Portfolio Metrics (22 KPIs) ──────────────────────────
    ws2 = wb.create_sheet("Portfolio Metrics")
    ws2.sheet_view.showGridLines = False
    banner(ws2,
        f"HPT v2.1 Portfolio Metrics | 1:3 R:R | 60-Day Window | "
        f"$100,000 Starting Capital | {len(master):,} Trades | "
        f"Max {MAX_SLOTS} Concurrent", 4)
    ws2.cell(2, 1, "Metric").font = HDR_FONT
    ws2.cell(2, 1).fill = HDR_FILL; ws2.cell(2,1).alignment = CTR
    ws2.cell(2, 1).border = BRD
    ws2.cell(2, 2, "Value").font = HDR_FONT
    ws2.cell(2, 2).fill = HDR_FILL; ws2.cell(2,2).alignment = CTR
    ws2.cell(2, 2).border = BRD
    ws2.cell(2, 3, "Bar").font = HDR_FONT
    ws2.cell(2, 3).fill = HDR_FILL; ws2.cell(2,3).alignment = CTR
    ws2.cell(2, 3).border = BRD
    ws2.cell(2, 4, "Note").font = HDR_FONT
    ws2.cell(2, 4).fill = HDR_FILL; ws2.cell(2,4).alignment = CTR
    ws2.cell(2, 4).border = BRD
    ws2.row_dimensions[2].height = 18

    notes = {
        "Total Trades": "All closed trades in 60-day window",
        "Unique Tickers": "Number of distinct S&P 500 stocks traded",
        "LONG Trades": "Trend-following buy signals",
        "SHORT Trades": "Trend-following sell signals",
        "Winning Trades": "Trades with PnL > 0",
        "Losing Trades": "Trades with PnL ≤ 0",
        "Win Rate %": "% of trades that were profitable",
        "Expectancy (R)": "Average R per trade — edge measure",
        "Total R": "Sum of all R-multiples",
        "Std Dev R": "Volatility of R outcomes",
        "Best Trade R": "Highest R winner (1:3 = +3R max)",
        "Worst Trade R": "Lowest R loser (stop = -1R)",
        "Gross Profit $": "Sum of all winning trade PnL",
        "Gross Loss $": "Sum of all losing trade PnL",
```

```
        "Net P&L $": "Total profit/loss across portfolio",
        "Profit Factor": "Gross Profit ÷ Gross Loss (>1 = edge)",
        "Avg Winner $": "Average $ gain on winning trades",
        "Avg Loser $": "Average $ loss on losing trades",
        "Sharpe Ratio": "Risk-adjusted return (annualized, 15-min bars)",
        "Maximum Drawdown %": "Largest peak-to-trough equity drop",
        "Max Drawdown as factor of R": "Max DD expressed in R units",
        "Net P&L as factor of R": "Net P&L expressed in R units",
    }
    pos_keys = {"Win Rate %","Expectancy (R)","Total R","Net P&L $","Gross Profit
$",
                "Profit Factor","Sharpe Ratio","Avg Winner $","Winning Trades",
                "Net P&L as factor of R"}
    neg_keys = {"Gross Loss $","Avg Loser $","Losing Trades",
                "Maximum Drawdown %","Worst Trade R","Max Drawdown as factor of R"}

    sections = {
        "TRADE STATISTICS":  ["Total Trades","Unique Tickers","LONG Trades",
                              "SHORT Trades","Winning Trades","Losing Trades"],
        "PERFORMANCE RATIOS": ["Win Rate %","Expectancy (R)","Total R","Std Dev R",
                               "Best Trade R","Worst Trade R","Profit
Factor","Sharpe Ratio"],
        "P&L BREAKDOWN":     ["Gross Profit $","Gross Loss $","Net P&L $",
                              "Avg Winner $","Avg Loser $"],
        "RISK & DRAWDOWN":   ["Maximum Drawdown %","Max Drawdown as factor of R",
                              "Net P&L as factor of R"],
    }
    rn = 3
    for sec, keys in sections.items():
        ws2.merge_cells(f"A{rn}:D{rn}")
        ws2[f"A{rn}"] = sec
        ws2[f"A{rn}"].font = SEC_FONT; ws2[f"A{rn}"].fill = SEC_FILL
        ws2[f"A{rn}"].alignment = CTR; ws2[f"A{rn}"].border = BRD
        ws2.row_dimensions[rn].height = 20; rn += 1
        for k in keys:
            if k not in metrics: continue
            v = metrics[k]
            lc = ws2.cell(rn, 1, k)
            lc.font=Font(bold=True, size=11, name="Calibri")
            lc.fill=LBL_FILL; lc.border=BRD; lc.alignment=LFT
            vc = ws2.cell(rn, 2, v)
            vc.border=BRD; vc.alignment=RT
            vc.font=Font(size=11, name="Calibri")
            if isinstance(v, float): vc.number_format = GEN
            if k in ("Gross Profit $","Gross Loss $","Net P&L $",
                     "Avg Winner $","Avg Loser $"):
                vc.number_format = CUR
            if k == "Win Rate %": vc.number_format = '0.00"%"'
            if k in pos_keys:
                vc.fill = WIN_FILL if (v or 0) >= 0 else LOS_FILL
            elif k in neg_keys:
```

```
                vc.fill = LOS_FILL
            else:
                vc.fill = NEU_FILL
            # visual bar
            bar_keys = {"Win Rate %","Expectancy (R)","Profit Factor",
                        "Sharpe Ratio","Net P&L as factor of R","Max Drawdown as
factor of R"}
            if k in bar_keys and isinstance(v, (int,float)):
                blen = min(int(abs(v) * 3), 38)
                bc = ws2.cell(rn, 3, "▓" * blen if blen > 0 else "░")
                bc.font = Font(
                    color="3FB950" if (v or 0) >= 0 else "DA3633",
                    size=9, name="Calibri")
                bc.border=BRD; bc.alignment=LFT
            else:
                ws2.cell(rn, 3, "").border=BRD
            nc = ws2.cell(rn, 4, notes.get(k, ""))
            nc.font=Font(size=9, color="7A7974", name="Calibri")
            nc.border=BRD; nc.alignment=LFT
            ws2.row_dimensions[rn].height = 18; rn += 1
        rn += 1

    ws2.column_dimensions["A"].width = 30
    ws2.column_dimensions["B"].width = 20
    ws2.column_dimensions["C"].width = 18
    ws2.column_dimensions["D"].width = 45

    # ── Sheet 3: Per-Ticker Leaderboard ──────────────────────────
    ws3 = wb.create_sheet("Per-Ticker Leaderboard")
    banner(ws3, "HPT v2.1 — Per-Ticker Leaderboard | Sorted by Expectancy R", 8)
    th = ["Ticker","Trades","Win Rate %","Expectancy R",
          "Total R","Net P&L $","Profit Factor","Best Trade R"]
    hdrs(ws3, th)
    ws3.freeze_panes = "A3"
    for rn2, (_, row) in enumerate(ticker_df.iterrows(), 3):
        f = WIN_FILL if row["Expectancy R"] > 0 else LOS_FILL
        for j, col in enumerate(th, 1):
            v = row.get(col, "")
            c = ws3.cell(rn2, j, v)
            c.fill=f; c.alignment=CTR; c.border=BRD
            c.font = Font(size=10, name="Calibri")
            if col not in ("Ticker",):
                c.number_format = GEN
            if col == "Net P&L $": c.number_format = CUR
            if col == "Win Rate %": c.number_format = '0.0"%"'
    for i, w in enumerate([8,8,12,14,12,14,14,14], 1):
        ws3.column_dimensions[get_column_letter(i)].width = w

    # ── Sheet 4: Strategy Checklist ──────────────────────────────
    ws4 = wb.create_sheet("Strategy Checklist")
    banner(ws4, "HPT v2.1 — Pre-Trade Checklist | Marcel Link + Robert Miner "
```

```
                "| Van Tharp R | 1:3 R:R", 4)
    ws4.sheet_view.showGridLines = False
    checklist = [
        ("⚠  DATA INTEGRITY (v2.0 FIX)", [
            "Both daily AND 15-min data use period='60d' → identical window, no
mixing.",
        "EOD fix: pre-computes last bar per (sym, date) → force-closes at day
end.",
        "Risk fix: slot_risk = equity × 1% (not ÷ 10) → true 1% per trade.",
            "Daily EMA200 replaced by EMA50 — fits cleanly in 60 days (50 bars
needed).",
            "Portfolio engine enforces max 10 concurrent trades from a shared $100k
pool.",
            "Slot risk = Portfolio Equity × 1% ÷ 10 slots — scales with compounding
equity.",
        ]),
        ("RULE 1 — MASTER TREND FILTER  (Marcel Link)", [
            "□ LONG:  Daily Close  >  Daily EMA(50)   → 60d-compatible bull market
proxy",
            "□ SHORT: Daily Close  <  Daily EMA(50)   → 60d-compatible bear market
proxy",
        ]),
        ("RULE 2 — LARGER TF MOMENTUM  (Miner DTF Rule 1)", [
            "□ LONG:  Daily MACD Histogram  >  0",
            "□ SHORT: Daily MACD Histogram  <  0",
        ]),
        ("RULE 3 — SMALLER TF REVERSAL TRIGGER  (Miner DTF Rule 2)", [
            "□ LONG:  15-min MACD Histogram crosses negative → positive",
            "□ SHORT: 15-min MACD Histogram crosses positive → negative",
        ]),
        ("RULE 4 — MA CROSSOVER CONFIRMATION  (Link 10/35 System)", [
            "□ LONG:  15-min EMA(10) crosses ABOVE EMA(35) on same bar as Rule 3",
            "□ SHORT: 15-min EMA(10) crosses BELOW EMA(35) on same bar as Rule 3",
        ]),
        ("RULE 5 — PULLBACK ENTRY  (Marcel Link — never chase)", [
            "□ Price was within 0.5% of EMA(35) in at least 1 of the last 5 bars",
            "□ Entering after a pullback, not at a breakout",
        ]),
        ("POSITION SIZING  (Van K. Tharp CPR — Portfolio-Adjusted)", [
            "□ Total capital: $100,000 shared across all positions",
            "□ Maximum 10 trades open simultaneously",
            "□ Slot Risk = Portfolio Equity × 1% ÷ 10 active slots",
            "□ Stop Distance = 2 × ATR(14)  on 15-min bars",
            "□ Shares = Slot Risk ÷ Stop Distance",
            "□ LONG:  SL = Entry − 2×ATR  |  TP = Entry + 6×ATR   (1:3 R:R)",
            "□ SHORT: SL = Entry + 2×ATR  |  TP = Entry − 6×ATR   (1:3 R:R)",
        ]),
        ("EXIT RULES", [
            "□ Take Profit:    High ≥ TP (LONG) or Low ≤ TP (SHORT)  → +3R",
            "□ Stop Loss:      Low ≤ SL (LONG) or High ≥ SL (SHORT) → −1R",
```

```
            "□ Momentum Exit:  EMA10 crosses back against trade (Miner
divergence)",
            "□ EOD:            Force-close at LAST BAR OF EACH TRADING DAY (not
dataset end)",
            "□ NEVER move stop against the trade",
        ]),
    ]
    rn = 3
    for sec, items in checklist:
        ws4.merge_cells(f"A{rn}:D{rn}")
        ws4[f"A{rn}"] = sec
        ws4[f"A{rn}"].font = SEC_FONT
        ws4[f"A{rn}"].fill = SEC_FILL
        ws4[f"A{rn}"].alignment = CTR
        ws4.row_dimensions[rn].height = 18; rn += 1
        for item in items:
            ws4.merge_cells(f"A{rn}:D{rn}")
            ws4[f"A{rn}"].value = item
            ws4[f"A{rn}"].font  = Font(size=10, name="Calibri")
            ws4.row_dimensions[rn].height = 15; rn += 1
        rn += 1
    for i, w in enumerate([4, 65, 30, 12], 1):
        ws4.column_dimensions[get_column_letter(i)].width = w

    wb.save(path)
    print(f"  ✓ Excel → {path}")


# ═══════════════════════════════════════════════════════════════════════
# MAIN
# ═══════════════════════════════════════════════════════════════════════
if __name__ == "__main__":
    print("=" * 70)
    print("  HIGH PROBABILITY TRADING STRATEGY  v2.1  |  S&P 500")
    print("  Authors: Marcel Link + Robert Miner  |  Framework: Van K. Tharp")
    print(f"  Risk: {RISK_PCT*100:.1f}%/slot  |  SL: {STOP_MULT}×ATR  |  "
          f"TP: {RISK_REWARD}×SL (1:{int(RISK_REWARD)} R:R)  |  "
          f"Portfolio: ${INITIAL_EQUITY:,}  |  Max Slots: {MAX_SLOTS}")
    print(f"  Data: yfinance period='60d' for BOTH daily & 15-min (no mixing)")
    print("=" * 70)

    # 1. Tickers
    sp500 = get_sp500()

    # 2. Fetch all tickers in parallel
    print(f"\n[FETCH] {len(sp500)} tickers | {MAX_WORKERS} threads |
period=60d...")
    all_daily = {}; all_15m = {}; t0 = time.time(); fetched = 0

    with ThreadPoolExecutor(max_workers=MAX_WORKERS) as exe:
        futs = {exe.submit(fetch_ticker, sym): sym for sym in sp500}
```

```
        done = 0
        for fut in as_completed(futs):
            sym, d, f15, status = fut.result(); done += 1
            if d is not None and f15 is not None:
                all_daily[sym] = d; all_15m[sym] = f15; fetched += 1
            if done % 50 == 0 or done == len(sp500):
                print(f"  [{done:>3}/{len(sp500)}]  valid: {fetched}  "
                      f"elapsed: {time.time()-t0:.0f}s")

    print(f"\n✓ {fetched} tickers ready | {time.time()-t0:.0f}s")

    if not all_15m:
        print("⚠  No data fetched. Check internet connection / yfinance version.")
        raise SystemExit(1)

    # 3. Portfolio backtest (single chronological pass)
    print("\n[BACKTEST] Running portfolio engine (max 10 concurrent)...")
    t1 = time.time()
    master, eq_curve = run_portfolio(all_15m, all_daily, INITIAL_EQUITY)
    print(f"  ✓ {len(master):,} trades | {time.time()-t1:.0f}s")

    if master.empty:
        print("⚠  No trades generated — check signal thresholds.")
        raise SystemExit(1)

    # 4. Metrics
    avg_risk  = master["RiskAmt"].mean()
    metrics   = portfolio_metrics(master, eq_curve, avg_risk)
    ticker_df = per_ticker_metrics(master)
    print_metrics(metrics)

    # 5. Export
    print("[EXPORT] Saving outputs...")
    export_excel(master, metrics, ticker_df)
    plot_all(master, metrics, ticker_df, eq_curve)

    print("\n✅  ALL DONE — check outputhpt2-1/")
    print("   hpt_v2_trades_journal.xlsx  (Trade Journal | Portfolio Metrics |
Leaderboard | Checklist)")
    print("   hpt_v2_performance.png      (7-panel: equity curve + 6 analytics)")
    print("=" * 70)
```

C.2: HPT-3

```
"""
================================================================================
HIGH PROBABILITY TRADING STRATEGY v3.0 — S&P 500
================================================================================
Based on:
  • Marcel Link   — "High Probability Trading"
```

```
  • Robert Miner  — "High Probability Trading Strategies"
  • Van K. Tharp  — Position Sizing / R-Multiples

v3.0 UPGRADES
  ✔ Feature 1 — Entry at Next Bar's Open (removes look-ahead bias)
  ✔ Feature 2 — Tick-Based Exits on every bar across a global timeline
  ✔ Feature 3 — Slippage model: 0.05% adverse each side
  ✔ Feature 4 — Commission: $1.00 flat per side ($2.00 round trip)
  ✔ Feature 5 — Gap-adjusted stops and targets
  ✔ Strict 60-day yfinance boundary on BOTH daily and 15-min data
  ✔ Shared $100k portfolio, max 10 concurrent trades
  ✔ 1:3 risk-to-reward

OUTPUTS:
    output/hpt_v3_trades_journal.xlsx
    output/hpt_v3_performance.png
================================================================================
"""

import os, time, math, warnings, requests
from io import StringIO
from datetime import datetime
from concurrent.futures import ThreadPoolExecutor, as_completed

import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import openpyxl
from openpyxl.styles import PatternFill, Font, Alignment, Border, Side
from openpyxl.utils import get_column_letter

warnings.filterwarnings('ignore')
os.makedirs('outputhpt3', exist_ok=True)

# ═══════════════════════════════════════════════════════════════════════════
# CONFIGURATION
# ═══════════════════════════════════════════════════════════════════════════
INITIAL_EQUITY = 100_000.0
RISK_PCT       = 0.01
MAX_SLOTS      = 10
ATR_PERIOD     = 14
STOP_MULT      = 2.0
RISK_REWARD    = 3.0
PULLBACK_ZONE  = 0.005
PULLBACK_BARS  = 5
MAX_WORKERS    = 12
```

```
DATA_PERIOD_1D  = '60d'
DATA_PERIOD_15M = '60d'

SLIPPAGE_PCT = 0.0005
COMMISSION   = 1.00

# ═══════════════════════════════════════════════════════════════════
# UNIVERSE
# ═══════════════════════════════════════════════════════════════════
def get_sp500():
    print('[TICKERS] Fetching S&P 500 from Wikipedia...')
    r = requests.get(
        'https://en.wikipedia.org/wiki/List_of_S%26P_500_companies',
        headers={'User-Agent': 'Mozilla/5.0'}, timeout=15)
    tickers = pd.read_html(StringIO(r.text))[0]['Symbol'].tolist()
    tickers = [t.replace('.', '-') for t in tickers]
    print(f'  ✓ {len(tickers)} tickers')
    return tickers

# ═══════════════════════════════════════════════════════════════════
# INDICATORS
# ═══════════════════════════════════════════════════════════════════
def _ema(s, span):
    return s.ewm(span=span, adjust=False).mean()

def _atr(df, p=14):
    tr = pd.concat([
        df['High'] - df['Low'],
        (df['High'] - df['Close'].shift()).abs(),
        (df['Low']  - df['Close'].shift()).abs()
    ], axis=1).max(axis=1)
    return tr.rolling(p).mean()

def _macd_hist(s, fast=12, slow=26, sig=9):
    macd = s.ewm(span=fast, adjust=False).mean() - s.ewm(span=slow,
adjust=False).mean()
    signal = macd.ewm(span=sig, adjust=False).mean()
    return macd - signal

def prepare_daily(df):
    df = df.copy()
    df['EMA50']  = _ema(df['Close'], 50)
    df['MACD_H'] = _macd_hist(df['Close'])
    df['ATR']    = _atr(df, ATR_PERIOD)
    return df

def prepare_15m(df):
    df = df.copy()
    df['EMA10']  = _ema(df['Close'], 10)
    df['EMA35']  = _ema(df['Close'], 35)
    df['MACD_H'] = _macd_hist(df['Close'])
```

```
    df['ATR']    = _atr(df, ATR_PERIOD)
    return df

# ═══════════════════════════════════════════════════════════════
# SIGNAL GENERATION  (Feature 1)
# ═══════════════════════════════════════════════════════════════
def generate_signals(sym, d, f):
    signals = []
    d_idx = d.index.values
    for i in range(max(35, PULLBACK_BARS + 2), len(f) - 1):
        ts = f.index[i]
        bar = f.iloc[i]
        px  = bar['Close']
        e10 = bar['EMA10']
        e35 = bar['EMA35']
        mh  = bar['MACD_H']
        atr = bar['ATR']
        if any(pd.isna(v) for v in [e10, e35, mh, atr]) or atr == 0:
            continue

        prev = f.iloc[i - 1]
        prev_mh  = prev['MACD_H']
        prev_e10 = prev['EMA10']
        prev_e35 = prev['EMA35']
        if any(pd.isna(v) for v in [prev_mh, prev_e10, prev_e35]):
            continue

        ts_day = pd.Timestamp(ts.year, ts.month, ts.day)
        di = int(np.searchsorted(d_idx, ts_day.to_datetime64(), side='right')) - 1
        if di < 12 or di >= len(d):
            continue

        d_close  = d['Close'].iat[di]
        d_ema50  = d['EMA50'].iat[di]
        d_macd_h = d['MACD_H'].iat[di]
        if any(pd.isna(v) for v in [d_ema50, d_macd_h]):
            continue

        near_ema35 = any(
            abs(f['Close'].iat[i-k] - f['EMA35'].iat[i-k]) /
            max(abs(f['EMA35'].iat[i-k]), 0.01) < PULLBACK_ZONE
            for k in range(1, PULLBACK_BARS + 1)
            if not pd.isna(f['EMA35'].iat[i-k])
        )
        if not near_ema35:
            continue

        if (d_close > d_ema50 and d_macd_h > 0 and
            mh > 0 and prev_mh <= 0 and
            e10 > e35 and prev_e10 <= prev_e35):
            signals.append({
```

```
                'sym': sym,
                'type': 'LONG',
                'signal': 'HPTLong_v3',
                'bar_idx': i,
                'signal_ts': ts,
                'atr': float(atr)
            })
        elif (d_close < d_ema50 and d_macd_h < 0 and
              mh < 0 and prev_mh >= 0 and
              e10 < e35 and prev_e10 >= prev_e35):
            signals.append({
                'sym': sym,
                'type': 'SHORT',
                'signal': 'HPTShort_v3',
                'bar_idx': i,
                'signal_ts': ts,
                'atr': float(atr)
            })
    return signals

# ═══════════════════════════════════════════════════════════════
# FILL MODELS  (Features 3 & 5)
# ═══════════════════════════════════════════════════════════════
def _apply_entry_slippage(open_px, direction):
    return open_px * (1 + SLIPPAGE_PCT) if direction == 'LONG' else open_px * (1 -
SLIPPAGE_PCT)

def _apply_exit_slippage(exit_px, direction):
    return exit_px * (1 - SLIPPAGE_PCT) if direction == 'LONG' else exit_px * (1 +
SLIPPAGE_PCT)

def _gross_pnl(direction, entry_px, exit_px, shares):
    return (exit_px - entry_px) * shares if direction == 'LONG' else (entry_px -
exit_px) * shares

def _close_trade(trades, pos, exit_px_raw, exit_ts, reason, eq):
    exit_px_fill = _apply_exit_slippage(exit_px_raw, pos['direction'])
    pnl_gross = _gross_pnl(pos['direction'], pos['entry_px'], exit_px_fill,
pos['shares'])
    pnl_net = pnl_gross - COMMISSION
    eq += pnl_net
    slip_exit = abs(exit_px_fill - exit_px_raw) * pos['shares']
    slip_total = pos['entry_slip_cost'] + slip_exit
    trades.append({
        'Ticker': pos['sym'],
        'Type': pos['direction'],
        'Signal': pos['signal'],
        'EntryTime': pos['entry_ts'],
        'EntryPrice': round(pos['entry_px'], 4),
        'StopLoss': round(pos['stop'], 4),
        'Target': round(pos['target'], 4),
```

```
        'Shares': round(pos['shares'], 2),
        'RiskAmt': round(pos['risk_amt'], 2),
        'Slip$': round(slip_total, 2),
        'Comm$': round(COMMISSION * 2, 2),
        'ExitPrice': round(exit_px_fill, 4),
        'ExitTime': exit_ts,
        'PnL': round(pnl_net, 2),
        'RMultiple': round(pnl_net / pos['risk_amt'], 4) if pos['risk_amt'] else 0,
        'EqBefore': round(pos['eq_before'], 2),
        'EqAfter': round(eq, 2),
        'ExitReason': reason,
    })
    return eq

# ═══════════════════════════════════════════════════════════════════════
# FETCH
# ═══════════════════════════════════════════════════════════════════════
def fetch_ticker(sym):
    try:
        t = yf.Ticker(sym)
        raw_d  = t.history(period=DATA_PERIOD_1D,  interval='1d')
        raw_15 = t.history(period=DATA_PERIOD_15M, interval='15m')
        frames = []
        for df in [raw_d, raw_15]:
            if isinstance(df.columns, pd.MultiIndex):
                df.columns = df.columns.get_level_values(0)
            if df.index.tz is not None:
                df.index = df.index.tz_localize(None)
            df.index = pd.to_datetime(df.index)
            frames.append(df[['Open','High','Low','Close','Volume']].dropna())
        d, f = frames
        if len(d) < 50 or len(f) < 50:
            return sym, None, None, None, 'insufficient'
        d = prepare_daily(d)
        f = prepare_15m(f)
        sigs = generate_signals(sym, d, f)
        return sym, d, f, sigs, 'ok'
    except Exception as e:
        return sym, None, None, None, str(e)[:80]

# ═══════════════════════════════════════════════════════════════════════
# PORTFOLIO ENGINE  (Feature 2 + all requested upgrades)
# ═══════════════════════════════════════════════════════════════════════
def run_portfolio(all_15m, all_daily, all_signals, equity0=INITIAL_EQUITY):
    eq = float(equity0)
    eq_curve = [eq]
    trades = []
    open_pos = {}
    pending_entries = {}
```

```
    global_timeline = sorted(set().union(*[set(df.index) for df in
all_15m.values()]))

    signals_by_ts = {}
    for sym, sigs in all_signals.items():
        for s in sigs:
            signals_by_ts.setdefault(s['signal_ts'], []).append(s)

    eod_map = {}
    for sym, df in all_15m.items():
        for ts in df.index:
            k = (sym, ts.date())
            if k not in eod_map or ts > eod_map[k]:
                eod_map[k] = ts

    for ts in global_timeline:
        # 1) check exits for ALL open trades on this bar
        for sym in list(open_pos.keys()):
            df = all_15m.get(sym)
            if df is None or ts not in df.index:
                continue
            bar = df.loc[ts]
            pos = open_pos[sym]
            o, h, l, c = float(bar['Open']), float(bar['High']), float(bar['Low']),
float(bar['Close'])

            if pos['direction'] == 'LONG':
                if o <= pos['stop']:
                    eq = _close_trade(trades, pos, o, ts, 'StopLoss_Gap', eq)
                    eq_curve.append(eq)
                    del open_pos[sym]
                    continue
                if o >= pos['target']:
                    eq = _close_trade(trades, pos, o, ts, 'TakeProfit_Gap', eq)
                    eq_curve.append(eq)
                    del open_pos[sym]
                    continue
                if l <= pos['stop']:
                    eq = _close_trade(trades, pos, pos['stop'], ts, 'StopLoss', eq)
                    eq_curve.append(eq)
                    del open_pos[sym]
                    continue
                if h >= pos['target']:
                    eq = _close_trade(trades, pos, pos['target'], ts, 'TakeProfit',
eq)
                    eq_curve.append(eq)
                    del open_pos[sym]
                    continue
            else:
                if o >= pos['stop']:
                    eq = _close_trade(trades, pos, o, ts, 'StopLoss_Gap', eq)
```

```
                    eq_curve.append(eq)
                    del open_pos[sym]
                    continue
                if o <= pos['target']:
                    eq = _close_trade(trades, pos, o, ts, 'TakeProfit_Gap', eq)
                    eq_curve.append(eq)
                    del open_pos[sym]
                    continue
                if h >= pos['stop']:
                    eq = _close_trade(trades, pos, pos['stop'], ts, 'StopLoss', eq)
                    eq_curve.append(eq)
                    del open_pos[sym]
                    continue
                if l <= pos['target']:
                    eq = _close_trade(trades, pos, pos['target'], ts, 'TakeProfit',
eq)
                    eq_curve.append(eq)
                    del open_pos[sym]
                    continue

            if eod_map.get((sym, ts.date())) == ts:
                eq = _close_trade(trades, pos, c, ts, 'EOD', eq)
                eq_curve.append(eq)
                del open_pos[sym]
                continue

        # 2) process entries scheduled for this exact bar open
        for ent in pending_entries.pop(ts, []):
            sym = ent['sym']
            if sym in open_pos or len(open_pos) >= MAX_SLOTS:
                continue
            df = all_15m[sym]
            if ts not in df.index:
                continue
            open_px_raw = float(df.loc[ts, 'Open'])
            entry_px = _apply_entry_slippage(open_px_raw, ent['type'])

            # entry commission deducted immediately before sizing
            eq -= COMMISSION
            slot_risk = eq * RISK_PCT
            stop_dist = ent['atr'] * STOP_MULT
            if stop_dist <= 0:
                eq += COMMISSION
                continue

            if ent['type'] == 'LONG':
                stop = entry_px - stop_dist
                target = entry_px + stop_dist * RISK_REWARD
            else:
                stop = entry_px + stop_dist
                target = entry_px - stop_dist * RISK_REWARD
```

```
            shares = math.floor(slot_risk / stop_dist)
            if shares < 1:
                eq += COMMISSION
                continue

            entry_slip_cost = abs(entry_px - open_px_raw) * shares
            open_pos[sym] = {
                'sym': sym,
                'direction': ent['type'],
                'signal': ent['signal'],
                'entry_ts': ts,
                'entry_px': entry_px,
                'stop': stop,
                'target': target,
                'shares': float(shares),
                'risk_amt': slot_risk,
                'eq_before': eq + COMMISSION,
                'entry_slip_cost': entry_slip_cost,
            }

        # 3) generate pending entries from signals that fire on this bar
        for s in signals_by_ts.get(ts, []):
            sym = s['sym']
            f = all_15m[sym]
            nxt = s['bar_idx'] + 1
            if nxt >= len(f):
                continue
            next_ts = f.index[nxt]
            pending_entries.setdefault(next_ts, []).append(s)

    # safety net
    for sym in list(open_pos.keys()):
        df = all_15m[sym]
        ts = df.index[-1]
        c = float(df['Close'].iat[-1])
        eq = _close_trade(trades, open_pos[sym], c, ts, 'EOD_SafetyNet', eq)
        eq_curve.append(eq)
        del open_pos[sym]

    return pd.DataFrame(trades), np.array(eq_curve)

# ═══════════════════════════════════════════════════════════════
# METRICS
# ═══════════════════════════════════════════════════════════════
def calc_max_drawdown(eq_series):
    peak = eq_series[0]
    mdd = 0.0
    for v in eq_series:
        if v > peak:
            peak = v
```

```
        dd = (peak - v) / peak if peak > 0 else 0
        if dd > mdd:
            mdd = dd
    return mdd

def portfolio_metrics(df, eq_curve, avg_risk):
    wins = df[df['PnL'] > 0]
    loss = df[df['PnL'] <= 0]
    r = df['RMultiple'].values
    pnl = df['PnL'].values
    gp = wins['PnL'].sum()
    gl = abs(loss['PnL'].sum())
    pf = gp / gl if gl else float('inf')
    net = pnl.sum()
    sh = (pnl.mean() / (pnl.std() + 1e-9)) * np.sqrt(252 * 26) if len(pnl) > 1 else
0
    mdd = calc_max_drawdown(eq_curve)
    mdd_r = (mdd * INITIAL_EQUITY) / avg_risk if avg_risk else 0
    net_r = net / avg_risk if avg_risk else 0
    return {
        'Total Trades': len(df),
        'Unique Tickers': int(df['Ticker'].nunique()),
        'LONG Trades': int((df['Type'] == 'LONG').sum()),
        'SHORT Trades': int((df['Type'] == 'SHORT').sum()),
        'Winning Trades': len(wins),
        'Losing Trades': len(loss),
        'Win Rate %': round(len(wins) / len(df) * 100, 2),
        'Expectancy (R)': round(float(r.mean()), 4),
        'Total R': round(float(r.sum()), 4),
        'Std Dev R': round(float(r.std()), 4),
        'Best Trade R': round(float(r.max()), 4),
        'Worst Trade R': round(float(r.min()), 4),
        'Gross Profit $': round(gp, 2),
        'Gross Loss $': round(-gl, 2),
        'Net P&L $': round(net, 2),
        'Profit Factor': round(pf, 3),
        'Avg Winner $': round(wins['PnL'].mean(), 2) if len(wins) else 0,
        'Avg Loser $': round(loss['PnL'].mean(), 2) if len(loss) else 0,
        'Sharpe Ratio': round(sh, 3),
        'Maximum Drawdown %': round(mdd * 100, 2),
        'Max Drawdown as factor of R': round(mdd_r, 2),
        'Net P&L as factor of R': round(net_r, 2),
    }

def per_ticker_metrics(df):
    rows = []
    for sym, grp in df.groupby('Ticker'):
        w = grp[grp['PnL'] > 0]
        l = grp[grp['PnL'] <= 0]
        r = grp['RMultiple'].values
        gp = w['PnL'].sum()
```

```
        gl = abs(l['PnL'].sum())
        rows.append({
            'Ticker': sym,
            'Trades': len(grp),
            'Win Rate %': round(len(w) / len(grp) * 100, 1),
            'Expectancy R': round(float(r.mean()), 4),
            'Total R': round(float(r.sum()), 4),
            'Net P&L $': round(grp['PnL'].sum(), 2),
            'Profit Factor': round(gp / gl, 3) if gl else 999,
            'Best Trade R': round(float(r.max()), 4),
        })
    return pd.DataFrame(rows).sort_values('Expectancy R', ascending=False)

def print_metrics(m):
    print('\n' + '=' * 62)
    print('  HIGH PROBABILITY TRADING v3.0 — RESULTS')
    print('=' * 62)
    for k, v in m.items():
        print(f'  {k:<32}: {v}')
    print('=' * 62 + '\n')

# ═══════════════════════════════════════════════════════════════
# CHARTS
# ═══════════════════════════════════════════════════════════════
def plot_all(master, metrics, ticker_df, eq_curve):
    fig = plt.figure(figsize=(22, 15), facecolor='#0d1117')
    fig.suptitle(
        'S&P 500 | High Probability Trading Strategy v3.0 | 1:3 R:R | '
        '$100k Portfolio | Max 10 Concurrent | 60-Day 15-Min Window | '
        'Next-Open Entry | Slippage + Commission + Gap Stops',
        color='white', fontsize=11, fontweight='bold', y=0.99)

    gs = gridspec.GridSpec(3, 3, figure=fig,
                           hspace=0.55, wspace=0.38,
                           left=0.06, right=0.97, top=0.95, bottom=0.06)

    def style(ax, title):
        ax.set_facecolor('#161b22')
        ax.set_title(title, color='white', fontsize=9.5, pad=5)
        ax.tick_params(colors='#8b949e', labelsize=8)
        for sp in ax.spines.values():
            sp.set_color('#30363d')
        ax.grid(alpha=0.12, color='#8b949e')

    rv = master['RMultiple'].values

    ax1 = fig.add_subplot(gs[0, :])
    style(ax1, f'Portfolio Equity Curve | Start: $100,000 | Final: ${eq_curve[-
1]:,.0f} | '
               f'Net P&L: ${metrics["Net P&L $"]:,.0f} | Max DD: {metrics["Maximum
Drawdown %"]}%')
```

```
    ax1.plot(range(len(eq_curve)), eq_curve, color='#58a6ff', lw=1.5)
    ax1.axhline(INITIAL_EQUITY, color='white', lw=0.8, ls='--', alpha=0.5)
    ax1.set_ylabel('Equity ($)', color='#8b949e', fontsize=8)
    ax1.set_xlabel('Trade #', color='#8b949e', fontsize=8)
    ax1.yaxis.set_major_formatter(plt.FuncFormatter(lambda x,_: f'${x:,.0f}'))

    ax2 = fig.add_subplot(gs[1, 0])
    style(ax2, f'R-Multiple Distribution | E={metrics["Expectancy (R)"]:.4f} | '
               f'WR={metrics["Win Rate %"]}% | PF={metrics["Profit Factor"]}')
    bins = np.linspace(rv.min() - 0.1, rv.max() + 0.1, 55)
    ax2.hist(rv[rv > 0], bins=bins, color='#3fb950', alpha=0.75)
    ax2.hist(rv[rv <= 0], bins=bins, color='#da3633', alpha=0.75)
    ax2.axvline(0, color='white', lw=1, alpha=0.4)
    ax2.axvline(rv.mean(), color='#f0e040', lw=1.5, ls='--')

    ax3 = fig.add_subplot(gs[1, 1])
    style(ax3, 'Exit Reasons')
    ec = master['ExitReason'].value_counts()
    colors = {'StopLoss':'#da3633','TakeProfit':'#3fb950','StopLoss_Gap':'#ff7b72',
              'TakeProfit_Gap':'#56d364','EOD':'#8b949e','EOD_SafetyNet':'#f0883e'}
    ax3.bar(ec.index, ec.values, color=[colors.get(x, '#58a6ff') for x in
ec.index], alpha=0.85)
    ax3.tick_params(axis='x', rotation=20)

    ax4 = fig.add_subplot(gs[1, 2])
    style(ax4, 'Top 20 Tickers — Expectancy R')
    t20 = ticker_df.head(20)
    c4 = ['#3fb950' if e > 0 else '#da3633' for e in t20['Expectancy R']]
    ax4.barh(t20['Ticker'][::-1], t20['Expectancy R'][::-1], color=c4[::-1],
alpha=0.85)

    ax5 = fig.add_subplot(gs[2, 0])
    style(ax5, 'LONG vs SHORT R-Distribution')
    ln = master[master['Type'] == 'LONG']['RMultiple'].values
    sh = master[master['Type'] == 'SHORT']['RMultiple'].values
    b2 = np.linspace(min(rv.min(), -1.5), max(rv.max(), 3.5), 40)
    if len(ln): ax5.hist(ln, bins=b2, color='#3fb950', alpha=0.6)
    if len(sh): ax5.hist(sh, bins=b2, color='#da3633', alpha=0.6)

    ax6 = fig.add_subplot(gs[2, 1])
    style(ax6, 'Net P&L — Top 30 Tickers')
    t30 = ticker_df.nlargest(30, 'Net P&L $')
    c6 = ['#3fb950' if v > 0 else '#da3633' for v in t30['Net P&L $']]
    ax6.bar(range(len(t30)), t30['Net P&L $'].values, color=c6, alpha=0.85)
    ax6.set_xticks(range(len(t30)))
    ax6.set_xticklabels(t30['Ticker'].values, rotation=90, fontsize=6,
color='#8b949e')

    ax7 = fig.add_subplot(gs[2, 2])
    style(ax7, 'Win Rate Distribution (per Ticker)')
    wr = ticker_df['Win Rate %'].values
```

```
    ax7.hist(wr, bins=25, color='#58a6ff', alpha=0.85, edgecolor='#30363d')
    ax7.axvline(wr.mean(), color='#f0e040', lw=1.5, ls='--')
    ax7.axvline(50, color='white', lw=1, ls=':', alpha=0.5)

    fig.savefig('outputhpt3/hpt_v3_performance.png', dpi=150, bbox_inches='tight',
facecolor='#0d1117')
    plt.close(fig)
    print('  ✓ Chart → outputhpt3/hpt_v3_performance.png')

# ═══════════════════════════════════════════════════════════════════
# EXCEL EXPORT
# ═══════════════════════════════════════════════════════════════════
def export_excel(master, metrics, ticker_df):
    path = 'outputhpt3/hpt_v3_trades_journal.xlsx'
    wb = openpyxl.Workbook()

    HDR_FILL = PatternFill('solid', fgColor='1F4E79')
    HDR_FONT = Font(bold=True, color='FFFFFF', size=11, name='Calibri')
    WIN_FILL = PatternFill('solid', fgColor='E2EFDA')
    LOS_FILL = PatternFill('solid', fgColor='FADADD')
    NEU_FILL = PatternFill('solid', fgColor='FFF2CC')
    SEC_FILL = PatternFill('solid', fgColor='2F5496')
    SEC_FONT = Font(bold=True, size=11, color='FFFFFF', name='Calibri')
    LBL_FILL = PatternFill('solid', fgColor='EEF2FF')
    CTR = Alignment(horizontal='center', vertical='center')
    RT = Alignment(horizontal='right', vertical='center')
    LFT = Alignment(horizontal='left', vertical='center', indent=1)
    BRD = Border(left=Side(style='thin'), right=Side(style='thin'),
top=Side(style='thin'), bottom=Side(style='thin'))
    CUR = '$#,##0.00'
    GEN = '#,##0.00'

    def banner(ws, text, nc=19):
        ws.merge_cells(start_row=1, start_column=1, end_row=1, end_column=nc)
        c = ws.cell(1, 1, text)
        c.font = Font(bold=True, size=13, color='FFFFFF', name='Calibri')
        c.fill = HDR_FILL
        c.alignment = CTR
        ws.row_dimensions[1].height = 28
        ws.sheet_view.showGridLines = False

    def hdrs(ws, cols, row=2):
        for j, h in enumerate(cols, 1):
            c = ws.cell(row, j, h)
            c.fill = HDR_FILL
            c.font = HDR_FONT
            c.alignment = CTR
            c.border = BRD
        ws.row_dimensions[row].height = 18

    ws1 = wb.active
```

```
    ws1.title = 'Trade Journal'
    banner(ws1, 'HPT v3.0 — S&P 500 | 1:3 R:R | $100k Portfolio | Max 10 Concurrent
| 60-Day Window | Next-Open Entry | Tick-Based Exits | Slippage + Commission +
Gaps', 19)
    h1 = ['#','Ticker','Type','Signal','Entry Time','Entry $','Stop $','Target
$','Shares','Risk $','Slip $','Comm $','Exit $','Exit Time','P&L $','R-
Multiple','Eq Before $','Eq After $','Exit Reason']
    hdrs(ws1, h1)
    ws1.freeze_panes = 'A3'
    cols =
['Ticker','Type','Signal','EntryTime','EntryPrice','StopLoss','Target','Shares','Ri
skAmt','Slip$','Comm$','ExitPrice','ExitTime','PnL','RMultiple','EqBefore','EqAfter
','ExitReason']
    for rn, (_, row) in enumerate(master.iterrows(), 3):
        fill = WIN_FILL if row['PnL'] > 0 else LOS_FILL
        c0 = ws1.cell(rn, 1, rn - 2)
        c0.fill = fill; c0.alignment = CTR; c0.border = BRD
        for j, col in enumerate(cols, 2):
            v = row.get(col, '')
            c = ws1.cell(rn, j, v)
            c.fill = fill; c.alignment = CTR; c.border = BRD
            c.font = Font(size=10, name='Calibri')
            if col in ('EntryPrice','StopLoss','Target','ExitPrice'):
                c.number_format = '#,##0.0000'
            if col in ('Shares',):
                c.number_format = GEN
            if col in ('RiskAmt','Slip$','Comm$','PnL','EqBefore','EqAfter'):
                c.number_format = CUR
            if col == 'RMultiple':
                c.number_format = '#,##0.0000'
    for i, w in enumerate([5,8,7,14,20,13,13,13,10,11,10,10,13,20,13,12,14,14,16],
1):
        ws1.column_dimensions[get_column_letter(i)].width = w

    ws2 = wb.create_sheet('Portfolio Metrics')
    ws2.sheet_view.showGridLines = False
    banner(ws2, f'HPT v3.0 Portfolio Metrics | 1:3 R:R | $100,000 Starting Capital
| {len(master):,} Trades | Costs Included', 4)
    for i, title in enumerate(['Metric','Value','Bar','Note'], 1):
        c = ws2.cell(2, i, title)
        c.font = HDR_FONT; c.fill = HDR_FILL; c.alignment = CTR; c.border = BRD

    notes = {
        'Total Trades':'All completed trades in 60-day window',
        'Expectancy (R)':'Includes slippage and commissions',
        'Gross Profit $':'After commissions on winners',
        'Gross Loss $':'After commissions and slippage on losers',
        'Maximum Drawdown %':'Peak-to-trough equity decline',
    }
    sections = {
```

```
        'TRADE STATISTICS':['Total Trades','Unique Tickers','LONG Trades','SHORT
Trades','Winning Trades','Losing Trades'],
        'PERFORMANCE RATIOS':['Win Rate %','Expectancy (R)','Total R','Std Dev
R','Best Trade R','Worst Trade R','Profit Factor','Sharpe Ratio'],
        'P&L BREAKDOWN':['Gross Profit $','Gross Loss $','Net P&L $','Avg Winner
$','Avg Loser $'],
        'RISK & DRAWDOWN':['Maximum Drawdown %','Max Drawdown as factor of R','Net
P&L as factor of R'],
    }
    rn = 3
    for sec, keys in sections.items():
        ws2.merge_cells(f'A{rn}:D{rn}')
        c = ws2[f'A{rn}']
        c.value = sec; c.font = SEC_FONT; c.fill = SEC_FILL; c.alignment = CTR;
c.border = BRD
        rn += 1
        for k in keys:
            v = metrics[k]
            a = ws2.cell(rn, 1, k); a.font = Font(bold=True, size=11,
name='Calibri'); a.fill = LBL_FILL; a.border = BRD; a.alignment = LFT
            b = ws2.cell(rn, 2, v); b.border = BRD; b.alignment = RT; b.font =
Font(size=11, name='Calibri')
            if isinstance(v, float): b.number_format = GEN
            if '$' in k: b.number_format = CUR
            if k == 'Win Rate %': b.number_format = '0.00"%"'
            b.fill = WIN_FILL if (not isinstance(v, (int,float)) or v >= 0) else
LOS_FILL
            bar = ws2.cell(rn, 3, '' * min(int(abs(v) * 3), 35) if isinstance(v,
(int,float)) else '')
            bar.border = BRD; bar.alignment = LFT
            note = ws2.cell(rn, 4, notes.get(k, ''))
            note.border = BRD; note.alignment = LFT; note.font = Font(size=9,
color='7A7974', name='Calibri')
            rn += 1
        rn += 1
    ws2.column_dimensions['A'].width = 30
    ws2.column_dimensions['B'].width = 20
    ws2.column_dimensions['C'].width = 18
    ws2.column_dimensions['D'].width = 40

    ws3 = wb.create_sheet('Per-Ticker Leaderboard')
    banner(ws3, 'HPT v3.0 — Per-Ticker Leaderboard | Sorted by Expectancy R', 8)
    th = ['Ticker','Trades','Win Rate %','Expectancy R','Total R','Net P&L
$','Profit Factor','Best Trade R']
    hdrs(ws3, th)
    for rn, (_, row) in enumerate(ticker_df.iterrows(), 3):
        fill = WIN_FILL if row['Expectancy R'] > 0 else LOS_FILL
        for j, col in enumerate(th, 1):
            c = ws3.cell(rn, j, row.get(col, ''))
            c.fill = fill; c.alignment = CTR; c.border = BRD; c.font =
Font(size=10, name='Calibri')
```

```
            if col != 'Ticker': c.number_format = GEN
            if col == 'Net P&L $': c.number_format = CUR
            if col == 'Win Rate %': c.number_format = '0.0"%"'
    for i, w in enumerate([8,8,12,14,12,14,14,14], 1):
        ws3.column_dimensions[get_column_letter(i)].width = w

    ws4 = wb.create_sheet('Strategy Checklist')
    banner(ws4, 'HPT v3.0 — Pre-Trade Checklist | Link + Miner + Van Tharp |
Realism Upgrades', 4)
    checklist = [
        ('DATA & EXECUTION REALISM', [
            "Both daily and 15-min data use period='60d' — no mixed-resolution
bias.",
            'Entries happen at NEXT BAR OPEN, never at signal-bar close.',
            'Every open trade is checked on EVERY bar via global timeline.',
            'Slippage = 0.05% adverse on entry and exit.',
            'Commission = $1 entry + $1 exit per trade.',
            'Gap logic fills at OPEN when price gaps through SL or TP.',
        ]),
        ('ENTRY RULES', [
            '☐ Daily Close > EMA50 and Daily MACD_H > 0 for LONG',
            '☐ Daily Close < EMA50 and Daily MACD_H < 0 for SHORT',
            '☐ 15-min MACD_H crossover confirms reversal',
            '☐ 15-min EMA10 / EMA35 crossover confirms direction',
            '☐ Price touched EMA35 zone within last 5 bars',
        ]),
        ('POSITION SIZING', [
            '☐ Shared $100k portfolio, max 10 concurrent positions',
            '☐ Risk per trade = 1% of current equity',
            '☐ Stop = 2×ATR(14), Target = 6×ATR(14) → 1:3 R:R',
            '☐ Shares are floored to whole shares',
        ]),
        ('EXIT RULES', [
            '☐ StopLoss / TakeProfit checked on every 15-min bar',
            '☐ StopLoss_Gap / TakeProfit_Gap if bar opens beyond level',
            '☐ EOD force-close at last bar of each trading day',
        ]),
    ]
    rn = 3
    for sec, items in checklist:
        ws4.merge_cells(f'A{rn}:D{rn}')
        c = ws4[f'A{rn}']; c.value = sec; c.font = SEC_FONT; c.fill = SEC_FILL;
c.alignment = CTR
        rn += 1
        for item in items:
            ws4.merge_cells(f'A{rn}:D{rn}')
            ws4[f'A{rn}'] = item
            ws4[f'A{rn}'].font = Font(size=10, name='Calibri')
            rn += 1
        rn += 1
    for i, w in enumerate([4,65,30,12], 1):
```

```
        ws4.column_dimensions[get_column_letter(i)].width = w

    wb.save(path)
    print(f'  ✓ Excel → {path}')

# ═══════════════════════════════════════════════════════════════════
# MAIN
# ═══════════════════════════════════════════════════════════════════
if __name__ == '__main__':
    print('=' * 78)
    print('  HIGH PROBABILITY TRADING STRATEGY v3.0 | S&P 500')
    print('  1:3 R:R | Next-Open Entry | Tick-Based Exits | Slippage + Commission +
Gaps')
    print(f'  Portfolio: ${INITIAL_EQUITY:,.0f} | Risk: {RISK_PCT*100:.1f}%/trade |
Max Slots: {MAX_SLOTS}')
    print("  Data: yfinance period='60d' for BOTH daily & 15-min")
    print('=' * 78)

    sp500 = get_sp500()

    print(f'\n[FETCH] {len(sp500)} tickers | {MAX_WORKERS} threads |
period=60d...')
    all_daily, all_15m, all_signals = {}, {}, {}
    t0 = time.time(); fetched = 0

    with ThreadPoolExecutor(max_workers=MAX_WORKERS) as exe:
        futs = {exe.submit(fetch_ticker, sym): sym for sym in sp500}
        done = 0
        for fut in as_completed(futs):
            sym, d, f, sigs, status = fut.result()
            done += 1
            if d is not None and f is not None:
                all_daily[sym] = d
                all_15m[sym] = f
                all_signals[sym] = sigs
                fetched += 1
            if done % 50 == 0 or done == len(sp500):
                print(f'  [{done:>3}/{len(sp500)}] valid: {fetched} | elapsed:
{time.time()-t0:.0f}s')

    if not all_15m:
        print('⚠ No data fetched.')
        raise SystemExit(1)

    print('\n[BACKTEST] Running global-timeline portfolio engine...')
    master, eq_curve = run_portfolio(all_15m, all_daily, all_signals,
INITIAL_EQUITY)
    if master.empty:
        print('⚠ No trades generated.')
        raise SystemExit(1)
    print(f'  ✓ {len(master):,} trades')
```

```
    avg_risk = master['RiskAmt'].mean()
    metrics = portfolio_metrics(master, eq_curve, avg_risk)
    ticker_df = per_ticker_metrics(master)
    print_metrics(metrics)

    print('[EXPORT] Saving outputs...')
    export_excel(master, metrics, ticker_df)
    plot_all(master, metrics, ticker_df, eq_curve)

    print('\n✅ ALL DONE — check output/')
    print('   hpt_v3_trades_journal.xlsx')
    print('   hpt_v3_performance.png')
    print('=' * 78)
```

C.3: Pivot Portfolio V7

```
"""
================================================================================
S&P 500 — TRADITIONAL PIVOT POINTS STRATEGY — 15-MIN BACKTEST ENGINE  v5.0
================================================================================
KEY CHANGES FROM v4.0:
  • SHARED $100k PORTFOLIO  — risk is sized as % of SHARED equity, not per-ticker
  • MAX 10 CONCURRENT TRADES — at any moment across ALL tickers
  • PORTFOLIO-LEVEL equity curve — single running balance for all trades
  • EXTENDED METRICS — max drawdown $, max drawdown in R, net P&L in R, etc.
================================================================================
Universe  : All S&P 500 tickers (~503) via Wikipedia live scrape
Entry TF  : 15-minute bars (Yahoo Finance, 60-day history)
Pivot TF  : Daily (computed from prior day OHLC — no lookahead)
Framework : Van K. Tharp R-Multiple / CPR Position Sizing
================================================================================
"""

import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import openpyxl
from openpyxl.styles import PatternFill, Font, Alignment, Border, Side
from openpyxl.utils import get_column_letter
from concurrent.futures import ThreadPoolExecutor, as_completed
from datetime import datetime, timedelta, time as dtime
from io import StringIO
import requests, warnings, os, time, heapq, random
```

```python
warnings.filterwarnings('ignore')
os.makedirs("outputppportfolio7", exist_ok=True)


# ═══════════════════════════════════════════════════════════════════════
# CONFIGURATION
# ═══════════════════════════════════════════════════════════════════════
INITIAL_EQUITY  = 100_000      # SINGLE shared portfolio
BASE_RISK_PCT   = 0.01         # 1% of PORTFOLIO equity per trade
ATR_PERIOD      = 14
STOP_ATR_MULT   = 1.0
RISK_REWARD     = 3.0
MAX_CONCURRENT  = 10           # max simultaneous open trades across all tickers
MAX_WORKERS     = 4

today   = datetime.now()
D_START = (today - timedelta(days=1100)).strftime("%Y-%m-%d")


# ═══════════════════════════════════════════════════════════════════════
# S&P 500 TICKER LIST
# ═══════════════════════════════════════════════════════════════════════

def get_sp500_tickers():
    print("[TICKERS] Fetching S&P 500 list from Wikipedia...")
    r = requests.get(
        "https://en.wikipedia.org/wiki/List_of_S%26P_500_companies",
        headers={'User-Agent': 'Mozilla/5.0'}
    )
    tickers = pd.read_html(StringIO(r.text))[0]['Symbol'].tolist()
    tickers = [t.replace('.', '-') for t in tickers]
    print(f"  ✓ {len(tickers)} tickers loaded")
    return tickers


# ═══════════════════════════════════════════════════════════════════════
# INDICATORS
# ═══════════════════════════════════════════════════════════════════════

def calc_atr(df, p=14):
    tr = pd.concat([
        df['High'] - df['Low'],
        (df['High'] - df['Close'].shift()).abs(),
        (df['Low']  - df['Close'].shift()).abs()
    ], axis=1).max(axis=1)
    return tr.rolling(p).mean()


def compute_daily_pivots(daily_df):
    H = daily_df['High'].shift(1)
    L = daily_df['Low'].shift(1)
    C = daily_df['Close'].shift(1)
```

```
    P = (H + L + C) / 3
    return pd.DataFrame({
        'P'  : P,
        'R1' : P * 2 - L,
        'S1' : P * 2 - H,
        'R2' : P + (H - L),
        'S2' : P - (H - L),
        'R3' : P * 2 + (H - 2 * L),
        'S3' : P * 2 - (2 * H - L),
    }, index=daily_df.index)


# ═══════════════════════════════════════════════════════════════
# DATA FETCHING (parallel, returns raw bars only)
# ═══════════════════════════════════════════════════════════════

def fetch_ticker(sym, max_retries=3):
    """
    Fetch daily + 15-min bars with retry + exponential back-off.
    Returns (sym, daily_df, fifteen_df) or (sym, None, None) on failure.
    Back-off: 2s -> 4s -> 8s between attempts to survive Yahoo rate limits.
    """
    for attempt in range(max_retries):
        try:
            t      = yf.Ticker(sym)
            raw_d  = t.history(start=D_START, end=today.strftime("%Y-%m-%d"),
interval="1d")
            raw_15 = t.history(period="60d",  interval="15m")

            dfs = []
            for df in [raw_d, raw_15]:
                if isinstance(df.columns, pd.MultiIndex):
                    df.columns = df.columns.get_level_values(0)
                if df.index.tz is not None:
                    df.index = df.index.tz_localize(None)
                df.index = pd.to_datetime(df.index)
                dfs.append(df[['Open','High','Low','Close','Volume']].dropna())

            d, f = dfs
            if len(d) < 20 or len(f) < 50:
                return sym, None, None
            f['ATR'] = calc_atr(f, ATR_PERIOD)
            return sym, d, f

        except Exception:
            if attempt < max_retries - 1:
                time.sleep(2 ** (attempt + 1))   # 2s, 4s, 8s
    return sym, None, None


# ═══════════════════════════════════════════════════════════════
```

```python
# SIGNAL GENERATOR  — no state, just returns candidate signals for each bar
# ═══════════════════════════════════════════════════════════════════════

def generate_signals(sym, daily_df, fifteen_df):
    """
    Returns a list of dicts sorted by timestamp — one per potential entry.
    Each dict: {ts, sym, type, signal, entry_px, sl, tp, atr}
    No equity / position-size computed here — that happens in the portfolio sim.
    """
    pivots = compute_daily_pivots(daily_df)
    f      = fifteen_df
    d_ts   = daily_df.index.values
    signals = []

    for i in range(2, len(f)):
        bar = f.iloc[i]
        ts  = f.index[i]
        px  = bar['Close']
        hi1 = f['High'].iat[i-1]
        lo1 = f['Low'].iat[i-1]
        px1 = f['Close'].iat[i-1]
        atr = bar['ATR']
        if pd.isna(atr) or atr == 0:
            continue

        ts_day = pd.Timestamp(ts.year, ts.month, ts.day)
        di_ = int(np.searchsorted(d_ts, ts_day.to_datetime64(), side='right')) - 1
        if di_ < 1 or di_ >= len(pivots):
            continue

        pP  = pivots['P'].iat[di_]
        pR1 = pivots['R1'].iat[di_]
        pS1 = pivots['S1'].iat[di_]
        if any(pd.isna(v) for v in [pP, pR1, pS1]):
            continue

        # LONG: S1 Bounce
        if lo1 <= pS1 and px > pS1:
            sl = px - atr * STOP_ATR_MULT
            tp = px + atr * STOP_ATR_MULT * RISK_REWARD
            if px - sl > 0.01:
                signals.append({'ts':ts, 'sym':sym, 'type':'LONG',
                                'signal':'BounceS1', 'entry_px':px,
                                'sl':sl, 'tp':tp, 'atr':atr})

        # LONG: Pivot Breakout
        elif px1 < pP and px > pP:
            sl = px - atr * STOP_ATR_MULT
            tp = px + atr * STOP_ATR_MULT * RISK_REWARD
            if px - sl > 0.01:
                signals.append({'ts':ts, 'sym':sym, 'type':'LONG',
```

```
                                'signal':'BreakP_Long', 'entry_px':px,
                                'sl':sl, 'tp':tp, 'atr':atr})

        # SHORT: R1 Rejection
        elif hi1 >= pR1 and px < pR1:
            sl = px + atr * STOP_ATR_MULT
            tp = px - atr * STOP_ATR_MULT * RISK_REWARD
            if sl - px > 0.01:
                signals.append({'ts':ts, 'sym':sym, 'type':'SHORT',
                                'signal':'RejectR1', 'entry_px':px,
                                'sl':sl, 'tp':tp, 'atr':atr})

        # SHORT: Pivot Breakdown
        elif px1 > pP and px < pP:
            sl = px + atr * STOP_ATR_MULT
            tp = px - atr * STOP_ATR_MULT * RISK_REWARD
            if sl - px > 0.01:
                signals.append({'ts':ts, 'sym':sym, 'type':'SHORT',
                                'signal':'BreakP_Short', 'entry_px':px,
                                'sl':sl, 'tp':tp, 'atr':atr})

    return signals


# ═══════════════════════════════════════════════════════════════
# PORTFOLIO SIMULATOR
# ═══════════════════════════════════════════════════════════════

def run_portfolio_simulation(ticker_data):
    """
    ticker_data: dict {sym: (daily_df, fifteen_df)}

    Simulates a SINGLE shared $100k portfolio across all tickers.
    Rules:
      - At most MAX_CONCURRENT trades open at any given timestamp.
      - Position size = (equity * BASE_RISK_PCT) / stop_distance at entry.
      - Equity updates in real-time: every exit updates the shared equity.
      - One trade per ticker at a time (no re-entry while position open).
      - Signals are processed in chronological order; earliest timestamp wins
        when multiple signals compete for the last slot.
    """
    # Step 1: collect all candidate signals from all tickers
    print("[SIM] Generating signals for all tickers...")
    all_signals = []
    for sym, (daily_df, fifteen_df) in ticker_data.items():
        sigs = generate_signals(sym, daily_df, fifteen_df)
        all_signals.extend(sigs)

    # Shuffle first so same-timestamp signals are in RANDOM order.
    # Without this, A/B/C tickers always grab all 10 slots first.
    random.shuffle(all_signals)
```

```
    all_signals.sort(key=lambda x: x['ts'])   # stable sort preserves shuffle
within same ts
    print(f"  ✓ {len(all_signals):,} raw candidate signals")

    # Step 2: build a unified 15-min price index for managing open trades
    # We need to check each open trade at every future bar
    # Build per-ticker bar lookup: {sym: fifteen_df}
    price_map = {sym: dfs[1] for sym, dfs in ticker_data.items()}

    # Step 3: portfolio-level simulation
    equity        = INITIAL_EQUITY
    eq_curve      = [(pd.Timestamp.min, equity)]   # (timestamp, equity)
    open_trades   = {}     # sym → trade dict  (max 1 per ticker)
    trade_log     = []
    open_count    = 0      # live counter of concurrent open trades
    tickers_open  = set()  # set of syms currently in a trade

    # Build a time-sorted event queue:
    # Events = signal arrivals. We'll also need to check exit conditions.
    # Strategy: process signals in order; at each new signal timestamp,
    # first check if any open trades have hit SL/TP at this timestamp.

    def check_exits_up_to(ts, equity_ref):
        """Check all open trades for SL/TP hits on bars <= ts. Returns updated
equity."""
        nonlocal open_count
        eq = equity_ref
        to_remove = []
        for sym, tr in list(open_trades.items()):
            f = price_map[sym]
            # Find bars strictly after entry and up to ts
            entry_ts = tr['EntryTime']
            mask = (f.index > entry_ts) & (f.index <= ts)
            bars_since = f[mask]
            EOD_TIME = dtime(15, 45)   # last 15-min bar of NYSE session
            for bar_ts, bar in bars_since.iterrows():
                hi = bar['High']; lo = bar['Low']
                hit = False
                if tr['type'] == 'LONG':
                    if hi >= tr['tp']:
                        pnl = (tr['tp'] - tr['entry_px']) * tr['shares']
                        eq += pnl
                        _close_trade(tr, tr['tp'], bar_ts, pnl, 'TakeProfit', eq)
                        hit = True
                    elif lo <= tr['sl']:
                        pnl = (tr['sl'] - tr['entry_px']) * tr['shares']
                        eq += pnl
                        _close_trade(tr, tr['sl'], bar_ts, pnl, 'StopLoss', eq)
                        hit = True
                else:
                    if lo <= tr['tp']:
```

```
                        pnl = (tr['entry_px'] - tr['tp']) * tr['shares']
                        eq += pnl
                        _close_trade(tr, tr['tp'], bar_ts, pnl, 'TakeProfit', eq)
                        hit = True
                    elif hi >= tr['sl']:
                        pnl = (tr['entry_px'] - tr['sl']) * tr['shares']
                        eq += pnl
                        _close_trade(tr, tr['sl'], bar_ts, pnl, 'StopLoss', eq)
                        hit = True
                # ── EOD intraday close: force-exit at 15:45 (end of NYSE session)
──
                if not hit and bar_ts.time() >= EOD_TIME:
                    close_px = float(bar['Close'])
                    pnl = ((close_px - tr['entry_px']) if tr['type'] == 'LONG'
                           else (tr['entry_px'] - close_px)) * tr['shares']
                    eq += pnl
                    _close_trade(tr, close_px, bar_ts, pnl, 'EOD', eq)
                    hit = True
                if hit:
                    eq_curve.append((bar_ts, eq))
                    trade_log.append(tr)
                    to_remove.append(sym)
                    open_count -= 1
                    break  # stop iterating bars for this trade; it's closed
        for sym in to_remove:
            del open_trades[sym]
            tickers_open.discard(sym)
        return eq

    def _close_trade(tr, exit_px, exit_ts, pnl, reason, eq_after):
        tr['ExitPrice']   = round(float(exit_px), 4)
        tr['ExitTime']    = exit_ts
        tr['TotalPnL']    = round(pnl, 2)
        tr['RMultiple']   = round(pnl / tr['risk_amt'], 4) if tr['risk_amt'] else 0
        tr['EquityAfter'] = round(eq_after, 2)
        tr['ExitReason']  = reason

    # Process signals in chronological order
    last_checked_ts = pd.Timestamp.min
    prev_sig_ts     = None

    for sig in all_signals:
        ts  = sig['ts']
        sym = sig['sym']

        # Check exits for all bars up to this signal's timestamp
        if ts != last_checked_ts:
            equity = check_exits_up_to(ts, equity)
            last_checked_ts = ts

        # Skip if ticker already in a trade
```

```
        if sym in tickers_open:
            continue

        # Skip if portfolio is at max concurrent trades
        if open_count >= MAX_CONCURRENT:
            continue

        # Open trade
        entry_px = sig['entry_px']
        sl       = sig['sl']
        tp       = sig['tp']
        stop_dist = abs(entry_px - sl)
        risk_amt  = equity * BASE_RISK_PCT
        shares    = risk_amt / stop_dist if stop_dist > 0 else 0
        if shares <= 0:
            continue

        tr = {
            'Ticker'      : sym,
            'Type'        : sig['type'],
            'Signal'      : sig['signal'],
            'EntryTime'   : ts,
            'EntryPrice'  : round(entry_px, 4),
            'StopLoss'    : round(sl, 4),
            'Target'      : round(tp, 4),
            'shares'      : shares,
            'Shares'      : round(shares, 2),
            'risk_amt'    : risk_amt,
            'RiskAmt'     : round(risk_amt, 2),
            'type'        : sig['type'],
            'entry_px'    : entry_px,
            'sl'          : sl,
            'tp'          : tp,
            'ExitPrice'   : None,
            'ExitTime'    : None,
            'TotalPnL'    : None,
            'RMultiple'   : None,
            'EquityBefore': round(equity, 2),
            'EquityAfter' : round(equity, 2),
            'ExitReason'  : 'Open',
        }
        open_trades[sym] = tr
        tickers_open.add(sym)
        open_count += 1

    # Force-close all remaining open trades at last bar
    last_ts = pd.Timestamp.max
    for sym, tr in list(open_trades.items()):
        f = price_map[sym]
        lp = float(f['Close'].iat[-1])
        last_bar_ts = f.index[-1]
```

```
        pnl = ((lp - tr['entry_px']) if tr['type'] == 'LONG'
               else (tr['entry_px'] - lp)) * tr['shares']
        equity += pnl
        eq_curve.append((last_bar_ts, equity))
        _close_trade(tr, lp, last_bar_ts, pnl, 'EOD', equity)
        trade_log.append(tr)

    print(f"  ✓ {len(trade_log):,} trades executed | Final equity: ${equity:,.2f}")
    trades_df = pd.DataFrame(trade_log)
    return trades_df, eq_curve, equity


# ═══════════════════════════════════════════════════════════════════════
# METRICS
# ═══════════════════════════════════════════════════════════════════════

def compute_max_drawdown(eq_curve):
    """Returns (max_drawdown_$, max_drawdown_pct) from equity curve list of (ts,
eq)."""
    equities = np.array([e for _, e in eq_curve])
    peak = equities[0]; max_dd = 0.0
    for eq in equities:
        if eq > peak:
            peak = eq
        dd = peak - eq
        if dd > max_dd:
            max_dd = dd
    return max_dd


def portfolio_metrics(df, eq_curve, initial_equity=INITIAL_EQUITY):
    wins = df[df['TotalPnL'] > 0]
    loss = df[df['TotalPnL'] <= 0]
    r    = df['RMultiple'].values
    gp   = wins['TotalPnL'].sum()
    gl   = abs(loss['TotalPnL'].sum())
    pf   = gp / gl if gl else np.inf
    pnl  = df['TotalPnL'].values
    pnl_mean = pnl.mean(); pnl_std = pnl.std(ddof=1) if len(pnl) > 1 else 1e-9
    sh   = (pnl_mean / (pnl_std + 1e-9)) * np.sqrt(252) if len(pnl) > 1 else 0
    net_pnl = df['TotalPnL'].sum()
    max_dd  = compute_max_drawdown(eq_curve)
    avg_r   = r.mean() if len(r) else 0
    avg_risk = df['RiskAmt'].mean()  # average risk $ per trade
    max_dd_r = max_dd / avg_risk if avg_risk else 0
    net_pnl_r = net_pnl / avg_risk if avg_risk else 0

    return {
        'Total Trades'          : len(df),
        'Unique Tickers'        : int(df['Ticker'].nunique()),
        'LONG Trades'           : int((df['Type'] == 'LONG').sum()),
```

```
        'SHORT Trades'          : int((df['Type'] == 'SHORT').sum()),
        'Winning Trades'        : len(wins),
        'Losing Trades'         : len(loss),
        'Win Rate %'            : round(len(wins) / len(df) * 100, 2),
        'Expectancy (R)'        : round(float(avg_r), 4),
        'Total R'               : round(float(r.sum()), 4),
        'Std Dev R'             : round(float(r.std(ddof=1)), 4) if len(r) > 1 else
0,
        'Best Trade R'          : round(float(r.max()), 4),
        'Worst Trade R'         : round(float(r.min()), 4),
        'Gross Profit $'        : round(gp, 2),
        'Gross Loss $'          : round(-gl, 2),
        'Net P&L $'             : round(net_pnl, 2),
        'Profit Factor'         : round(pf, 4),
        'Avg Winner $'          : round(wins['TotalPnL'].mean(), 2) if len(wins)
else 0,
        'Avg Loser $'           : round(loss['TotalPnL'].mean(), 2) if len(loss)
else 0,
        'Sharpe Ratio'          : round(float(sh), 4),
        'Max Drawdown $'        : round(max_dd, 2),
        'Max Drawdown (R)'      : round(max_dd_r, 4),
        'Net P&L (R)'           : round(net_pnl_r, 4),
    }


def per_ticker_metrics(df):
    rows = []
    for sym, grp in df.groupby('Ticker'):
        w = grp[grp['TotalPnL'] > 0]; l = grp[grp['TotalPnL'] <= 0]
        r = grp['RMultiple'].values
        gp = w['TotalPnL'].sum(); gl = abs(l['TotalPnL'].sum())
        rows.append({'Ticker':sym, 'Trades':len(grp),
                     'Win Rate %':round(len(w)/len(grp)*100, 1),
                     'Expectancy R':round(r.mean(), 4), 'Total R':round(r.sum(),
4),
                     'Net P&L $':round(grp['TotalPnL'].sum(), 2),
                     'Profit Factor':round(gp/gl, 3) if gl else 999,
                     'Best Trade R':round(r.max(), 4)})
    return pd.DataFrame(rows).sort_values('Expectancy R', ascending=False)


def print_metrics(m):
    print(f"\n{'='*58}")
    print(f"  S&P 500 — TRADITIONAL PIVOT POINTS — PORTFOLIO SIM")
    print(f"  Shared $100k | Max {MAX_CONCURRENT} concurrent trades")
    print(f"{'='*58}")
    for k, v in m.items():
        print(f"  {k:<26}: {v}")
    print(f"{'='*58}\n")


# ═══════════════════════════════════════════════════════════════
```

```
# CHARTS (7 panels)
# ═══════════════════════════════════════════════════════════════

def plot_all(master_df, metrics, ticker_df, eq_curve):
    fig = plt.figure(figsize=(20, 16), facecolor='#0d1117')
    fig.suptitle(
        f'S&P 500  |  Traditional Pivot Points  |  15-Min  |  '
        f'Shared $100k Portfolio  |  Max {MAX_CONCURRENT} Concurrent Trades',
        color='white', fontsize=12, fontweight='bold', y=0.99)

    gs = gridspec.GridSpec(4, 3, figure=fig,
                           hspace=0.58, wspace=0.38,
                           left=0.06, right=0.97, top=0.95, bottom=0.05)

    def style(ax, title):
        ax.set_facecolor('#161b22')
        ax.set_title(title, color='white', fontsize=10, pad=6)
        ax.tick_params(colors='#8b949e')
        for sp in ax.spines.values(): sp.set_color('#30363d')
        ax.grid(alpha=0.12, color='#8b949e')

    # 1. Portfolio equity curve (full width)
    ax0 = fig.add_subplot(gs[0, :])
    style(ax0, 'Portfolio Equity Curve — Shared $100k | Max 10 Concurrent Trades')
    eq_ts  = [ts for ts, _ in eq_curve if ts != pd.Timestamp.min]
    eq_val = [eq for ts, eq in eq_curve if ts != pd.Timestamp.min]
    if eq_ts:
        ax0.plot(eq_ts, eq_val, color='#58a6ff', lw=1.2, alpha=0.9)
        ax0.fill_between(eq_ts, 100_000, eq_val,
                         where=[v >= 100_000 for v in eq_val],
                         color='#3fb950', alpha=0.15)
        ax0.fill_between(eq_ts, 100_000, eq_val,
                         where=[v < 100_000 for v in eq_val],
                         color='#da3633', alpha=0.15)
        ax0.axhline(100_000, color='white', lw=0.8, ls='--', alpha=0.4,
label='Start')
        final = eq_val[-1] if eq_val else 100_000
        ax0.axhline(final, color='#f0e040', lw=0.8, ls=':', alpha=0.6,
                    label=f'Final ${final:,.0f}')
    ax0.set_ylabel('Equity $', color='#8b949e')
    ax0.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, _: f'${x:,.0f}'))
    ax0.legend(facecolor='#161b22', labelcolor='white', fontsize=8)

    # 2. R-Multiple distribution
    ax1 = fig.add_subplot(gs[1, :])
    style(ax1, f'R-Multiple Distribution  |  {len(master_df):,} Trades  |  '
               f'Expectancy: {metrics["Expectancy (R)"]:.4f}R  |  '
               f'Win Rate: {metrics["Win Rate %"]}%  |  '
               f'Profit Factor: {metrics["Profit Factor"]}')
    rv   = master_df['RMultiple'].values
    bins = np.linspace(-1.2, 2.2, 50)
```

```
    ax1.hist(rv[rv > 0],  bins=bins, color='#3fb950', alpha=0.75,
             label=f'Winners ({(rv>0).sum():,})')
    ax1.hist(rv[rv <= 0], bins=bins, color='#da3633', alpha=0.75,
             label=f'Losers ({(rv<=0).sum():,})')
    ax1.axvline(0,          color='white',   lw=1,   alpha=0.5)
    ax1.axvline(rv.mean(),  color='#f0e040', lw=2,   ls='--',
                label=f'Mean={rv.mean():.4f}R')
    ax1.set_xlabel('R-Multiple', color='#8b949e')
    ax1.set_ylabel('Frequency',  color='#8b949e')
    ax1.legend(facecolor='#161b22', labelcolor='white', fontsize=9)

    # 3. Signal breakdown
    ax2 = fig.add_subplot(gs[2, 0])
    style(ax2, 'Signal Breakdown')
    sc = master_df['Signal'].value_counts()
    ax2.barh(sc.index, sc.values,
             color=['#58a6ff','#3fb950','#f0883e','#da3633'][:len(sc)], alpha=0.85)
    ax2.set_xlabel('# Trades', color='#8b949e')
    for i, v in enumerate(sc.values):
        ax2.text(v + 5, i, f'{v:,}', va='center', color='white', fontsize=8)

    # 4. Exit reasons
    ax3 = fig.add_subplot(gs[2, 1])
    style(ax3, 'Exit Reasons')
    ec   = master_df['ExitReason'].value_counts()
    ec_c = {'StopLoss':'#da3633', 'TakeProfit':'#3fb950', 'EOD':'#8b949e'}
    ax3.bar(ec.index, ec.values,
            color=[ec_c.get(x, '#58a6ff') for x in ec.index], alpha=0.85)
    ax3.set_ylabel('# Trades', color='#8b949e')
    for i, (x, v) in enumerate(zip(ec.index, ec.values)):
        ax3.text(i, v + 5, f'{v:,}', ha='center', color='white', fontsize=9)

    # 5. Top 20 tickers by expectancy
    ax4 = fig.add_subplot(gs[2, 2])
    style(ax4, 'Top 20 Tickers (Expectancy R)')
    top20 = ticker_df.head(20)
    c4 = ['#3fb950' if e > 0 else '#da3633' for e in top20['Expectancy R']]
    ax4.barh(top20['Ticker'][::-1], top20['Expectancy R'][::-1],
             color=c4[::-1], alpha=0.85)
    ax4.axvline(0, color='white', lw=0.8, alpha=0.5)
    ax4.set_xlabel('Expectancy (R)', color='#8b949e')

    # 6. Long vs Short R-distribution
    ax5 = fig.add_subplot(gs[3, 0])
    style(ax5, 'Long vs Short R-Distribution')
    ln = master_df[master_df['Type'] == 'LONG']['RMultiple'].values
    sh = master_df[master_df['Type'] == 'SHORT']['RMultiple'].values
    b2 = np.linspace(-1.2, 2.2, 40)
    if len(ln): ax5.hist(ln, bins=b2, color='#3fb950', alpha=0.6,
                         label=f'LONG ({len(ln):,})  E={ln.mean():.3f}R')
    if len(sh): ax5.hist(sh, bins=b2, color='#da3633', alpha=0.6,
```

```
                        label=f'SHORT ({len(sh):,})  E={sh.mean():.3f}R')
    ax5.set_xlabel('R-Multiple', color='#8b949e')
    ax5.legend(facecolor='#161b22', labelcolor='white', fontsize=8)

    # 7. Top 30 tickers net P&L
    ax6 = fig.add_subplot(gs[3, 1])
    style(ax6, 'Net P&L — Top 30 Tickers')
    t30 = ticker_df.nlargest(30, 'Net P&L $')
    c6  = ['#3fb950' if v > 0 else '#da3633' for v in t30['Net P&L $']]
    ax6.bar(range(len(t30)), t30['Net P&L $'].values, color=c6, alpha=0.85)
    ax6.set_xticks(range(len(t30)))
    ax6.set_xticklabels(t30['Ticker'].values, rotation=90, fontsize=6,
color='#8b949e')
    ax6.set_ylabel('Net P&L $', color='#8b949e')
    ax6.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, _: f'${x:,.0f}'))

    # 8. Win rate distribution
    ax7 = fig.add_subplot(gs[3, 2])
    style(ax7, 'Win Rate Distribution (per Ticker)')
    wr = ticker_df['Win Rate %'].values
    ax7.hist(wr, bins=30, color='#58a6ff', alpha=0.85, edgecolor='#30363d')
    ax7.axvline(wr.mean(), color='#f0e040', lw=1.5, ls='--',
                label=f'Mean={wr.mean():.1f}%')
    ax7.axvline(50, color='white', lw=1, ls=':', alpha=0.5, label='50% line')
    ax7.set_xlabel('Win Rate %', color='#8b949e')
    ax7.set_ylabel('# Tickers',  color='#8b949e')
    ax7.legend(facecolor='#161b22', labelcolor='white', fontsize=8)

    fig.savefig('outputppportfolio7/strategy_performance.png', dpi=150,
                bbox_inches='tight', facecolor='#0d1117')
    plt.close(fig)
    print("  ✓ Chart → outputppportfolio7/strategy_performance.png")

# ═══════════════════════════════════════════════════════════════════
# EXCEL EXPORT
# ═══════════════════════════════════════════════════════════════════

def export_excel(master_df, metrics, ticker_df):
    path = "outputppportfolio7/pivot_portfolio_journal.xlsx"
    wb   = openpyxl.Workbook()

    # Style constants
    HDR_FILL  = PatternFill("solid", fgColor="1F4E79")
    HDR_FONT  = Font(bold=True, color="FFFFFF", size=11, name="Calibri")
    WIN_FILL  = PatternFill("solid", fgColor="E2EFDA")
    LOSS_FILL = PatternFill("solid", fgColor="FADADD")
    SEC_FILL  = PatternFill("solid", fgColor="2F5496")
    SEC_FONT  = Font(bold=True, size=11, color="FFFFFF", name="Calibri")
    LBL_FONT  = Font(bold=True, name="Calibri", size=11)
    VAL_FONT  = Font(name="Calibri", size=11)
    CENTER    = Alignment(horizontal='center', vertical='center')
```

```
LEFT      = Alignment(horizontal='left',   vertical='center', indent=1)
BORDER    = Border(left=Side(style='thin'),  right=Side(style='thin'),
                   top=Side(style='thin'),   bottom=Side(style='thin'))

def banner(ws, text, end_col="O"):
    ws.merge_cells(f"A1:{end_col}1")
    c = ws["A1"]
    c.value = text
    c.font  = Font(bold=True, size=13, color="FFFFFF", name="Calibri")
    c.fill  = HDR_FILL
    c.alignment = CENTER
    ws.row_dimensions[1].height = 28
    ws.sheet_view.showGridLines = False

def add_headers(ws, hdrs, row=2):
    for j, h in enumerate(hdrs, 1):
        c = ws.cell(row, j, h)
        c.fill = HDR_FILL; c.font = HDR_FONT
        c.alignment = CENTER; c.border = BORDER
    ws.row_dimensions[row].height = 20

# ── Sheet 1: Trade Journal ──────────────────────────────────────
ws1 = wb.active; ws1.title = "Trade Journal"
banner(ws1, f"S&P 500 — Pivot Points Portfolio  |  Shared $100k  |  "
            f"Max {MAX_CONCURRENT} Concurrent  |  {len(master_df):,} Trades")
hdrs = ['#','Ticker','Type','Signal','Entry Time','Entry $','Stop $',
        'Target $','Shares','Risk $','Exit $','Exit Time',
        'P&L $','R-Multiple','Equity Before','Equity After','Exit Reason']
add_headers(ws1, hdrs)
col_map = ['Ticker','Type','Signal','EntryTime','EntryPrice','StopLoss',
           'Target','Shares','RiskAmt','ExitPrice','ExitTime',
           'TotalPnL','RMultiple','EquityBefore','EquityAfter','ExitReason']
num_fmts = {'EntryPrice':'$#,##0.00', 'StopLoss':'$#,##0.00',
            'Target':'$#,##0.00', 'ExitPrice':'$#,##0.00',
            'Shares':'#,##0.00', 'RiskAmt':'$#,##0.00',
            'TotalPnL':'$#,##0.00', 'RMultiple':'0.0000',
            'EquityBefore':'$#,##0.00', 'EquityAfter':'$#,##0.00'}

for rn, (_, row) in enumerate(master_df.iterrows(), 3):
    pnl_val = row.get('TotalPnL', 0) or 0
    fill = WIN_FILL if pnl_val > 0 else LOSS_FILL
    ws1.cell(rn, 1, rn - 2).fill = fill
    ws1.cell(rn, 1).alignment = CENTER
    ws1.cell(rn, 1).border = BORDER
    for j, col in enumerate(col_map, 2):
        val = row.get(col, '')
        if isinstance(val, float) and pd.isna(val): val = ''
        c = ws1.cell(rn, j, value=val)
        c.fill = fill; c.alignment = CENTER; c.border = BORDER
        c.font = VAL_FONT
        if col in num_fmts:
```

```
                c.number_format = num_fmts[col]

    for i, w in enumerate([5,8,7,14,20,20,12,12,10,10,12,20,12,12,14,14,14], 1):
        ws1.column_dimensions[get_column_letter(i)].width = w
    ws1.freeze_panes = 'A3'

    # ── Sheet 2: Portfolio Metrics ──────────────────────────────
    ws2 = wb.create_sheet("Portfolio Metrics")
    banner(ws2, "Portfolio Metrics — Shared $100k | Van K. Tharp Framework", "D")
    ws2.sheet_view.showGridLines = False

    # Section groupings for metrics
    sections = {
        "TRADE COUNT": ["Total Trades","Unique Tickers","LONG Trades","SHORT
Trades",
                        "Winning Trades","Losing Trades"],
        "R-MULTIPLE ANALYSIS": ["Win Rate %","Expectancy (R)","Total R","Std Dev
R",
                                "Best Trade R","Worst Trade R","Net P&L (R)","Max
Drawdown (R)"],
        "P&L ANALYSIS": ["Gross Profit $","Gross Loss $","Net P&L $",
                         "Avg Winner $","Avg Loser $","Profit Factor"],
        "RISK METRICS": ["Sharpe Ratio","Max Drawdown $"],
    }

    row = 2
    for sec_name, keys in sections.items():
        # Section header
        ws2.merge_cells(f"B{row}:D{row}")
        c = ws2.cell(row, 2, sec_name)
        c.fill = SEC_FILL; c.font = SEC_FONT; c.alignment = CENTER
        ws2.row_dimensions[row].height = 20
        row += 1
        for k in keys:
            v = metrics.get(k, 'N/A')
            ck = ws2.cell(row, 2, k)
            ck.font = LBL_FONT; ck.border = BORDER; ck.alignment = LEFT
            cv = ws2.cell(row, 3, v)
            cv.font = VAL_FONT; cv.border = BORDER; cv.alignment = CENTER
            if isinstance(v, (int, float)) and not isinstance(v, bool):
                if '$' in k:
                    cv.number_format = '$#,##0.00'
                elif '%' in k:
                    cv.number_format = '0.00"%"' if isinstance(v, float) else '0'
                elif 'R)' in k or '(R' in k or k in ('Total R','Std Dev R','Best
Trade R',
                                                     'Worst Trade
R','Expectancy (R)'):
                    cv.number_format = '0.0000'
                else:
                    cv.number_format = '#,##0.00'
```

```
            if k in ('Win Rate %','Expectancy (R)','Profit Factor','Net P&L $',
                     'Sharpe Ratio','Net P&L (R)'):
                cv.fill = WIN_FILL if (isinstance(v, (int,float)) and v >= 0) else
LOSS_FILL
            if k in ('Max Drawdown $','Max Drawdown (R)','Gross Loss $','Worst
Trade R'):
                cv.fill = LOSS_FILL
            row += 1
        row += 1  # spacing between sections

    ws2.column_dimensions['A'].width = 3
    ws2.column_dimensions['B'].width = 28
    ws2.column_dimensions['C'].width = 20
    ws2.column_dimensions['D'].width = 5

    # ── Sheet 3: Per-Ticker Leaderboard ──────────────────────────────────
    ws3 = wb.create_sheet("Per-Ticker Leaderboard")
    banner(ws3, "Per-Ticker Leaderboard — Sorted by Expectancy R", "I")
    t_hdrs = ['Ticker','Trades','Win Rate %','Expectancy R',
              'Total R','Net P&L $','Profit Factor','Best Trade R']
    add_headers(ws3, t_hdrs)
    for rn, (_, row) in enumerate(ticker_df.iterrows(), 3):
        fill = WIN_FILL if row['Expectancy R'] > 0 else LOSS_FILL
        for j, col in enumerate(t_hdrs, 1):
            val = row.get(col, '')
            c = ws3.cell(rn, j, val)
            c.fill = fill; c.alignment = CENTER; c.border = BORDER; c.font =
VAL_FONT
            if col == 'Net P&L $': c.number_format = '$#,##0.00'
            elif col != 'Ticker': c.number_format = '#,##0.00'
    for i, w in enumerate([10,9,13,15,13,15,15,15], 1):
        ws3.column_dimensions[get_column_letter(i)].width = w
    ws3.freeze_panes = 'A3'

    # ── Sheet 4: Strategy Checklist ──────────────────────────────────────
    ws4 = wb.create_sheet("Strategy Checklist")
    banner(ws4, "TRADITIONAL PIVOT POINTS — Strategy Checklist v6.0", "E")
    ws4.sheet_view.showGridLines = False

    checklist = [
        ("PORTFOLIO ARCHITECTURE", [
            f"  Portfolio Capital     : $100,000 (shared across all trades)",
            f"  Max Concurrent Trades : {MAX_CONCURRENT} (enforced globally)",
            f"  Risk per Trade        : {BASE_RISK_PCT*100:.1f}% of current
portfolio equity",
            "  Position Size         : Risk $ ÷ Stop Distance (shares)",
            "  One trade per ticker at any given time",
        ]),
        ("PIVOT LEVEL FORMULAS (Traditional, Daily resolution)", [
            "  P  = (prev_High + prev_Low + prev_Close) / 3",
            "  R1 = P×2 − prev_Low              S1 = P×2 − prev_High",
```

```
            "  R2 = P + (prev_High − prev_Low)  S2 = P − (prev_High − prev_Low)",
            "  R3 = P×2 + (prev_High − 2×prev_Low)",
        ]),
        ("LONG ENTRY SIGNALS (15-min bar)", [
            "  □ BounceS1    : prior 15-min bar Low ≤ S1  AND  current Close > S1",
            "  □ BreakP_Long : prior 15-min Close < P     AND  current Close > P",
        ]),
        ("SHORT ENTRY SIGNALS (15-min bar)", [
            "  □ RejectR1     : prior 15-min bar High ≥ R1  AND  current Close <
R1",
            "  □ BreakP_Short : prior 15-min Close > P      AND  current Close <
P",
        ]),
        ("EXIT RULES", [
            f"  □ Stop Loss   = Entry ± {STOP_ATR_MULT}×ATR(14)",
            f"  □ Take Profit = Entry ± {STOP_ATR_MULT * RISK_REWARD}×ATR(14)  (R:R
= 1:{RISK_REWARD})",
            "  □ EOD force-close at 15:45 (end of NYSE session) — no overnight
holds",
        ]),
    ]

    row = 3
    for sec, items in checklist:
        ws4.merge_cells(f"B{row}:E{row}")
        ws4[f"B{row}"] = sec
        ws4[f"B{row}"].font = SEC_FONT
        ws4[f"B{row}"].fill = SEC_FILL
        ws4[f"B{row}"].alignment = LEFT
        ws4.row_dimensions[row].height = 20
        row += 1
        for item in items:
            ws4.merge_cells(f"B{row}:E{row}")
            ws4[f"B{row}"] = item
            ws4[f"B{row}"].font = Font(size=10, name="Calibri")
            ws4[f"B{row}"].alignment = LEFT
            ws4.row_dimensions[row].height = 16
            row += 1
        row += 1

    ws4.column_dimensions['A'].width = 3
    ws4.column_dimensions['B'].width = 55
    ws4.column_dimensions['C'].width = 5

    # ── Sheet 5: Drawdown Risk Scenarios ──────────────────────────────
    ws5 = wb.create_sheet("Drawdown Scenarios")
    ws5.sheet_view.showGridLines = False
    banner(ws5, "Drawdown Risk Scenario Analysis — $100,000 Portfolio", "N")

    SCENARIO_RISKS  = [0.005, 0.01, 0.02, 0.03, 0.05, 0.10, 0.25, 0.50, 1.00]
    SCENARIO_LABELS = ["0.5%", "1%", "2%", "3%", "5%", "10%", "25%", "50%", "100%"]
```

```
    SCEN_COLORS     = ["2ECC71","27AE60","F1C40F","E67E22","E74C3C",
                       "C0392B","8E44AD","6C3483","9B59B6"]

    r_vals = master_df['RMultiple'].dropna().values

    def _sim_scenario(risk_pct):
        eq, peak = INITIAL_EQUITY, INITIAL_EQUITY
        eq_curve_s, dd_curve_s, pnl_s = [eq], [0.0], []
        for r in r_vals:
            ra   = eq * risk_pct
            pnl  = ra * r
            eq  += pnl
            pnl_s.append(pnl)
            if eq > peak: peak = eq
            dd_curve_s.append((peak - eq) / peak * 100)
            eq_curve_s.append(eq)
        pnl_arr  = np.array(pnl_s)
        avg_risk = np.mean([INITIAL_EQUITY * risk_pct] * len(r_vals))
        max_dd_p = max(dd_curve_s)
        max_dd_d = max_dd_p / 100 * np.array(eq_curve_s).max()
        net_pnl  = eq - INITIAL_EQUITY
        sharpe   = (pnl_arr.mean()/(pnl_arr.std(ddof=1)+1e-9))*np.sqrt(252) if
len(pnl_arr)>1 else 0
        return {
            'eq': eq_curve_s, 'dd': dd_curve_s,
            'final': eq, 'net_pnl': net_pnl,
            'net_pct': net_pnl/INITIAL_EQUITY*100,
            'max_dd_pct': max_dd_p, 'max_dd_d': max_dd_d,
            'max_dd_r': max_dd_d / avg_risk if avg_risk else 0,
            'net_r': net_pnl / avg_risk if avg_risk else 0,
            'sharpe': sharpe,
        }

    scen = {lbl: _sim_scenario(rp) for lbl, rp in zip(SCENARIO_LABELS,
SCENARIO_RISKS)}

    # ── Summary table ──────────────────────────────────────────────
    SUM_HDRS = ["Risk %","Final Equity","Net P&L $","Net P&L %",
                "Max DD $","Max DD % (peak)","Max DD (R)","Net P&L
(R)","Sharpe","Outcome"]
    row = 3
    ws5.merge_cells(f"B{row}:K{row}")
    c = ws5.cell(row, 2, "Scenario Summary"); c.fill = SEC_FILL; c.font = SEC_FONT
    c.alignment = CENTER; ws5.row_dimensions[row].height = 20; row += 1
    for j, h in enumerate(SUM_HDRS, 2):
        c = ws5.cell(row, j, h); c.fill = HDR_FILL; c.font = HDR_FONT
        c.alignment = CENTER; c.border = BORDER
    ws5.row_dimensions[row].height = 20; row += 1

    for lbl, col_hex in zip(SCENARIO_LABELS, SCEN_COLORS):
        s = scen[lbl]
```

```
        fin = s['final']
        outcome = "RUINED" if fin < 1000 else ("⚠️ EXTREME" if s['max_dd_pct']>90
else
                  ("⚠️ HIGH" if s['max_dd_pct']>60 else "✅ VIABLE"))
        rfill = LOSS_FILL if fin < 1000 else (PatternFill("solid",
fgColor="FFF2CC")
                if s['max_dd_pct'] > 60 else WIN_FILL)
        vals = [lbl, f"${fin:,.0f}", f"${s['net_pnl']:,.0f}",
f"{s['net_pct']:.1f}%",
                f"${s['max_dd_d']:,.0f}", f"{s['max_dd_pct']:.1f}%",
                f"{s['max_dd_r']:.1f}R", f"{s['net_r']:.1f}R",
                f"{s['sharpe']:.3f}", outcome]
        for j, val in enumerate(vals, 2):
            c = ws5.cell(row, j, val); c.fill = rfill
            c.font = Font(name="Calibri", size=10,
                          bold=(outcome in ["RUINED","✅ VIABLE","⚠️ EXTREME","⚠️
HIGH"]),
                          color="FFFFFF" if outcome == "RUINED" else "000000")
            c.alignment = CENTER; c.border = BORDER
        row += 1
    row += 2

    # ── Equity curve data table ──────────────────────────────────────
    ws5.merge_cells(f"B{row}:K{row}")
    c = ws5.cell(row, 2, "Equity Curve by Risk Level"); c.fill = SEC_FILL
    c.font = SEC_FONT; c.alignment = CENTER; ws5.row_dimensions[row].height = 20;
row += 1
    ws5.cell(row, 2, "Trade #").fill = HDR_FILL; ws5.cell(row, 2).font = HDR_FONT
    ws5.cell(row, 2).alignment = CENTER
    for j, (lbl, col_hex) in enumerate(zip(SCENARIO_LABELS, SCEN_COLORS), 3):
        c = ws5.cell(row, j, lbl)
        c.fill = PatternFill("solid", fgColor=col_hex)
        c.font = Font(bold=True, color="FFFFFF" if j>4 else "000000", size=10,
name="Calibri")
        c.alignment = CENTER; c.border = BORDER
    ws5.row_dimensions[row].height = 20
    eq_hdr_row = row; row += 1
    n_pts  = len(r_vals)
    step   = max(1, n_pts // 500)
    eq_rows = 0
    for i in range(0, n_pts+1, step):
        ws5.cell(row, 2, i).number_format = "#,##0"; ws5.cell(row,2).alignment =
CENTER
        for j, lbl in enumerate(SCENARIO_LABELS, 3):
            eq_list = scen[lbl]['eq']
            val = eq_list[i] if i < len(eq_list) else eq_list[-1]
            c = ws5.cell(row, j, round(val, 2)); c.number_format = '$#,##0'
            c.alignment = CENTER
        row += 1; eq_rows += 1
    row += 2
```

```
    # ── Drawdown data table ──────────────────────────────────────────
    ws5.merge_cells(f"B{row}:K{row}")
    c = ws5.cell(row, 2, "Drawdown from Peak (%) by Risk Level"); c.fill = SEC_FILL
    c.font = SEC_FONT; c.alignment = CENTER; ws5.row_dimensions[row].height = 20;
row += 1
    ws5.cell(row, 2, "Trade #").fill = HDR_FILL; ws5.cell(row, 2).font = HDR_FONT
    ws5.cell(row, 2).alignment = CENTER
    for j, (lbl, col_hex) in enumerate(zip(SCENARIO_LABELS, SCEN_COLORS), 3):
        c = ws5.cell(row, j, lbl)
        c.fill = PatternFill("solid", fgColor=col_hex)
        c.font = Font(bold=True, color="FFFFFF" if j>4 else "000000", size=10,
name="Calibri")
        c.alignment = CENTER; c.border = BORDER
    ws5.row_dimensions[row].height = 20
    dd_hdr_row = row; row += 1
    dd_rows = 0
    for i in range(0, n_pts+1, step):
        ws5.cell(row, 2, i).number_format = "#,##0"; ws5.cell(row,2).alignment =
CENTER
        for j, lbl in enumerate(SCENARIO_LABELS, 3):
            dd_list = scen[lbl]['dd']
            val = dd_list[i] if i < len(dd_list) else dd_list[-1]
            c = ws5.cell(row, j, round(-val, 4)); c.number_format = '0.00"%"'
            c.alignment = CENTER
            if val > 50: c.fill = PatternFill("solid", fgColor="FADADD")
        row += 1; dd_rows += 1

    # ── Embed charts ─────────────────────────────────────────────────
    from openpyxl.chart import LineChart, Reference as ChartRef
    chart_eq2 = LineChart()
    chart_eq2.title = "Portfolio Equity Curve — Risk Scenarios"
    chart_eq2.style = 10; chart_eq2.height = 14; chart_eq2.width = 28
    chart_eq2.y_axis.title = "Equity ($)"; chart_eq2.x_axis.title = "Trade #"
    chart_eq2.add_data(
        ChartRef(ws5, min_col=3, max_col=11,
                 min_row=eq_hdr_row, max_row=eq_hdr_row+eq_rows),
        titles_from_data=True)
    chart_dd2 = LineChart()
    chart_dd2.title = "Drawdown from Peak (%) — Risk Scenarios"
    chart_dd2.style = 10; chart_dd2.height = 14; chart_dd2.width = 28
    chart_dd2.y_axis.title = "Drawdown (%)"; chart_dd2.x_axis.title = "Trade #"
    chart_dd2.add_data(
        ChartRef(ws5, min_col=3, max_col=11,
                 min_row=dd_hdr_row, max_row=dd_hdr_row+dd_rows),
        titles_from_data=True)
    ws5.add_chart(chart_eq2, "M3")
    ws5.add_chart(chart_dd2, "M45")

    for i, w in enumerate([3,10,13,13,13,13,13,13,13,13,13], 1):
        ws5.column_dimensions[get_column_letter(i)].width = w
```

```
    wb.save(path)
    print(f"  ✓ Excel → {path}")


# ═══════════════════════════════════════════════════════════════════════
# MAIN
# ═══════════════════════════════════════════════════════════════════════

if __name__ == "__main__":
    print("=" * 68)
    print("  S&P 500  |  TRADITIONAL PIVOT POINTS  |  PORTFOLIO SIM  |  v5.0")
    print(f"  Portfolio: ${INITIAL_EQUITY:,}  |  Risk/Trade:
{BASE_RISK_PCT*100:.1f}%  |  "
          f"Max Concurrent: {MAX_CONCURRENT}")
    print("=" * 68)

    # 1. Get tickers
    sp500 = get_sp500_tickers()

    # 2. Fetch all raw data in parallel
    print(f"\n[FETCH] {len(sp500)} tickers | {MAX_WORKERS} threads...")
    ticker_data = {}; failed = []; t0 = time.time()

    # Shuffle so rate-limit failures are random, not confined to letters D-Z
    sp500_shuffled = sp500.copy()
    random.shuffle(sp500_shuffled)

    BATCH_SIZE  = 50     # process 50 tickers per batch
    BATCH_PAUSE = 2.0    # seconds between batches

    done = 0
    for batch_start in range(0, len(sp500_shuffled), BATCH_SIZE):
        batch = sp500_shuffled[batch_start : batch_start + BATCH_SIZE]
        with ThreadPoolExecutor(max_workers=MAX_WORKERS) as exe:
            futs = {exe.submit(fetch_ticker, sym): sym for sym in batch}
            for fut in as_completed(futs):
                sym, daily_df, fifteen_df = fut.result(); done += 1
                if daily_df is not None:
                    ticker_data[sym] = (daily_df, fifteen_df)
                else:
                    failed.append(sym)
        print(f"  [{done:>3}/{len(sp500_shuffled)}] loaded: {len(ticker_data):>3} |
"
              f"failed: {len(failed):>3} | elapsed: {time.time()-t0:.0f}s")
        if batch_start + BATCH_SIZE < len(sp500_shuffled):
            time.sleep(BATCH_PAUSE)   # rate-limit pause between batches

    # Second-pass retry for any failed tickers
    if failed:
        print(f"\n[RETRY] {len(failed)} failed tickers — pausing 10s then
retrying...")
```

```
        time.sleep(10)
        retry_still_failed = []
        with ThreadPoolExecutor(max_workers=2) as exe:
            futs = {exe.submit(fetch_ticker, sym): sym for sym in failed}
            for fut in as_completed(futs):
                sym, daily_df, fifteen_df = fut.result()
                if daily_df is not None:
                    ticker_data[sym] = (daily_df, fifteen_df)
                    print(f"    ✓ Recovered {sym}")
                else:
                    retry_still_failed.append(sym)
        failed = retry_still_failed
        print(f"  Retry complete — loaded: {len(ticker_data)} | "
              f"permanently failed: {len(failed)}")

    print(f"\n✓ Data fetched: {len(ticker_data)} tickers in {time.time()-t0:.0f}s")

    # 3. Run portfolio-level simulation
    master, eq_curve, final_equity = run_portfolio_simulation(ticker_data)
    master = master.sort_values('EntryTime').reset_index(drop=True)
    print(f"  Master journal: {len(master):,} trades | "
          f"{master['Ticker'].nunique()} tickers | Final equity:
${final_equity:,.2f}")

    # 4. Metrics
    metrics   = portfolio_metrics(master, eq_curve)
    ticker_df = per_ticker_metrics(master)
    print_metrics(metrics)

    # 5. Export
    print("[EXPORT] Saving outputs...")
    export_excel(master, metrics, ticker_df)
    plot_all(master, metrics, ticker_df, eq_curve)

    print("\n✅ ALL DONE — outputs in outputppportfolio7/")
    print("   pivot_portfolio_journal.xlsx")
    print("   strategy_performance.png")
    print("=" * 68)
```

C.4: Pivot Portfolio V8

```
"""
================================================================================
S&P 500 — TRADITIONAL PIVOT POINTS STRATEGY — 15-MIN BACKTEST ENGINE  v8.0
================================================================================
KEY CHANGES IN v8.0 (from v7):
  • NEXT-BAR OPEN ENTRY  — entries fill at the open of the bar following the signal
  • TICK-BASED EXITS     — every market bar checked for SL/TP/EOD across all open
trades
  • SLIPPAGE MODEL       — 0.05% of fill price on entry and exit (both sides)
```

```
  • COMMISSION          — $1.00 flat per side ($2.00 round-trip per trade)
  • GAP-ADJUSTED STOPS  — if open gaps through SL or TP, fill at open price, not level
================================================================================
Universe  : All S&P 500 tickers (~503) via Wikipedia live scrape
Entry TF  : 15-minute bars (Yahoo Finance, 60-day history)
Pivot TF  : Daily (computed from prior day OHLC — no lookahead)
Framework : Van K. Tharp R-Multiple / CPR Position Sizing
================================================================================
"""

import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import openpyxl
from openpyxl.styles import PatternFill, Font, Alignment, Border, Side
from openpyxl.utils import get_column_letter
from concurrent.futures import ThreadPoolExecutor, as_completed
from datetime import datetime, timedelta, time as dtime
from io import StringIO
import requests, warnings, os, time, heapq, random
warnings.filterwarnings('ignore')
os.makedirs("outputppportfolio8", exist_ok=True)


# ═══════════════════════════════════════════════════════════════════════════
# CONFIGURATION
# ═══════════════════════════════════════════════════════════════════════════
INITIAL_EQUITY  = 100_000      # SINGLE shared portfolio
BASE_RISK_PCT   = 0.01         # 1% of PORTFOLIO equity per trade
ATR_PERIOD      = 14
STOP_ATR_MULT   = 1.0
RISK_REWARD     = 3.0
MAX_CONCURRENT  = 10           # max simultaneous open trades across all tickers
MAX_WORKERS     = 4
SLIPPAGE_PCT    = 0.0005       # 0.05% of fill price (each side)
COMMISSION      = 1.00         # $1.00 flat per side ($2 round-trip)

today   = datetime.now()
D_START = (today - timedelta(days=1100)).strftime("%Y-%m-%d")


# ═══════════════════════════════════════════════════════════════════════════
# S&P 500 TICKER LIST
# ═══════════════════════════════════════════════════════════════════════════

def get_sp500_tickers():
```

```
    print("[TICKERS] Fetching S&P 500 list from Wikipedia...")
    r = requests.get(
        "https://en.wikipedia.org/wiki/List_of_S%26P_500_companies",
        headers={'User-Agent': 'Mozilla/5.0'}
    )
    tickers = pd.read_html(StringIO(r.text))[0]['Symbol'].tolist()
    tickers = [t.replace('.', '-') for t in tickers]
    print(f"  ✓ {len(tickers)} tickers loaded")
    return tickers


# ═══════════════════════════════════════════════════════════════
# INDICATORS
# ═══════════════════════════════════════════════════════════════

def calc_atr(df, p=14):
    tr = pd.concat([
        df['High'] - df['Low'],
        (df['High'] - df['Close'].shift()).abs(),
        (df['Low']  - df['Close'].shift()).abs()
    ], axis=1).max(axis=1)
    return tr.rolling(p).mean()


def compute_daily_pivots(daily_df):
    H = daily_df['High'].shift(1)
    L = daily_df['Low'].shift(1)
    C = daily_df['Close'].shift(1)
    P = (H + L + C) / 3
    return pd.DataFrame({
        'P'  : P,
        'R1' : P * 2 - L,
        'S1' : P * 2 - H,
        'R2' : P + (H - L),
        'S2' : P - (H - L),
        'R3' : P * 2 + (H - 2 * L),
        'S3' : P * 2 - (2 * H - L),
    }, index=daily_df.index)


# ═══════════════════════════════════════════════════════════════
# DATA FETCHING (parallel, returns raw bars only)
# ═══════════════════════════════════════════════════════════════

def fetch_ticker(sym, max_retries=3):
    """
    Fetch daily + 15-min bars with retry + exponential back-off.
    Returns (sym, daily_df, fifteen_df) or (sym, None, None) on failure.
    Back-off: 2s -> 4s -> 8s between attempts to survive Yahoo rate limits.
    """
    for attempt in range(max_retries):
        try:
```

```
            t      = yf.Ticker(sym)
            raw_d  = t.history(start=D_START, end=today.strftime("%Y-%m-%d"),
interval="1d")
            raw_15 = t.history(period="60d",  interval="15m")

            dfs = []
            for df in [raw_d, raw_15]:
                if isinstance(df.columns, pd.MultiIndex):
                    df.columns = df.columns.get_level_values(0)
                if df.index.tz is not None:
                    df.index = df.index.tz_localize(None)
                df.index = pd.to_datetime(df.index)
                dfs.append(df[['Open','High','Low','Close','Volume']].dropna())

            d, f = dfs
            if len(d) < 20 or len(f) < 50:
                return sym, None, None
            f['ATR'] = calc_atr(f, ATR_PERIOD)
            return sym, d, f

        except Exception:
            if attempt < max_retries - 1:
                time.sleep(2 ** (attempt + 1))   # 2s, 4s, 8s
    return sym, None, None


# ═══════════════════════════════════════════════════════════════
# SIGNAL GENERATOR  — no state, just returns candidate signals for each bar
# ═══════════════════════════════════════════════════════════════

def generate_signals(sym, daily_df, fifteen_df):
    """
    Returns a list of dicts sorted by timestamp — one per potential entry.
    Each dict: {ts, sym, type, signal, entry_px, sl, tp, atr}
    No equity / position-size computed here — that happens in the portfolio sim.
    """
    pivots = compute_daily_pivots(daily_df)
    f      = fifteen_df
    d_ts   = daily_df.index.values
    signals = []

    for i in range(2, len(f)):
        bar = f.iloc[i]
        ts  = f.index[i]
        px  = bar['Close']
        hi1 = f['High'].iat[i-1]
        lo1 = f['Low'].iat[i-1]
        px1 = f['Close'].iat[i-1]
        atr = bar['ATR']
        if pd.isna(atr) or atr == 0:
            continue
```

```
        ts_day = pd.Timestamp(ts.year, ts.month, ts.day)
        di_ = int(np.searchsorted(d_ts, ts_day.to_datetime64(), side='right')) - 1
        if di_ < 1 or di_ >= len(pivots):
            continue

        pP  = pivots['P'].iat[di_]
        pR1 = pivots['R1'].iat[di_]
        pS1 = pivots['S1'].iat[di_]
        if any(pd.isna(v) for v in [pP, pR1, pS1]):
            continue

        # LONG: S1 Bounce
        if lo1 <= pS1 and px > pS1:
            sl = px - atr * STOP_ATR_MULT
            tp = px + atr * STOP_ATR_MULT * RISK_REWARD
            if px - sl > 0.01:
                signals.append({'ts':ts, 'sym':sym, 'type':'LONG',
                                'signal':'BounceS1', 'entry_px':px,
                                'sl':sl, 'tp':tp, 'atr':atr, 'bar_idx':i})

        # LONG: Pivot Breakout
        elif px1 < pP and px > pP:
            sl = px - atr * STOP_ATR_MULT
            tp = px + atr * STOP_ATR_MULT * RISK_REWARD
            if px - sl > 0.01:
                signals.append({'ts':ts, 'sym':sym, 'type':'LONG',
                                'signal':'BreakP_Long', 'entry_px':px,
                                'sl':sl, 'tp':tp, 'atr':atr, 'bar_idx':i})

        # SHORT: R1 Rejection
        elif hi1 >= pR1 and px < pR1:
            sl = px + atr * STOP_ATR_MULT
            tp = px - atr * STOP_ATR_MULT * RISK_REWARD
            if sl - px > 0.01:
                signals.append({'ts':ts, 'sym':sym, 'type':'SHORT',
                                'signal':'RejectR1', 'entry_px':px,
                                'sl':sl, 'tp':tp, 'atr':atr, 'bar_idx':i})

        # SHORT: Pivot Breakdown
        elif px1 > pP and px < pP:
            sl = px + atr * STOP_ATR_MULT
            tp = px - atr * STOP_ATR_MULT * RISK_REWARD
            if sl - px > 0.01:
                signals.append({'ts':ts, 'sym':sym, 'type':'SHORT',
                                'signal':'BreakP_Short', 'entry_px':px,
                                'sl':sl, 'tp':tp, 'atr':atr, 'bar_idx':i})

    return signals
```

```python
# ═══════════════════════════════════════════════════════════════════
# PORTFOLIO SIMULATOR
# ═══════════════════════════════════════════════════════════════════

def run_portfolio_simulation(ticker_data):
    """
    ticker_data: dict {sym: (daily_df, fifteen_df)}

    Simulates a SINGLE shared $100k portfolio across all tickers.
    Rules:
      - At most MAX_CONCURRENT trades open at any given timestamp.
      - Position size = (equity * BASE_RISK_PCT) / stop_distance at entry.
      - Equity updates in real-time: every exit updates the shared equity.
      - One trade per ticker at a time (no re-entry while position open).
      - Signals are processed in chronological order; earliest timestamp wins
        when multiple signals compete for the last slot.
    """
    EOD_TIME = dtime(15, 45)   # last 15-min bar of NYSE session

    # ── Step 1: generate signals ──────────────────────────────────
    print("[SIM] Generating signals for all tickers...")
    all_signals = []
    for sym, (daily_df, fifteen_df) in ticker_data.items():
        sigs = generate_signals(sym, daily_df, fifteen_df)
        all_signals.extend(sigs)

    random.shuffle(all_signals)
    all_signals.sort(key=lambda x: x['ts'])   # stable sort keeps random order
within ts
    print(f"  ✓ {len(all_signals):,} raw candidate signals")

    # ── Step 2: build price maps + global tick timeline ───────────
    price_map = {sym: dfs[1] for sym, dfs in ticker_data.items()}

    # Build a sorted global bar-timestamp list (TICK-BASED EXITS: every bar
checked)
    all_bar_ts_set = set()
    for f in price_map.values():
        all_bar_ts_set.update(f.index.tolist())
    global_timeline = sorted(all_bar_ts_set)

    # Index signal arrivals by their signal-bar timestamp
    # Entry will happen on the NEXT bar after the signal fires
    from collections import defaultdict
    signals_at = defaultdict(list)  # signal_ts -> [sig, ...]
    for sig in all_signals:
        signals_at[sig['ts']].append(sig)

    # Build per-ticker index maps for fast O(1) next-bar lookup
    ticker_bar_idx = {sym: {ts: i for i, ts in enumerate(f.index)}
                      for sym, f in price_map.items()}
```

```
    # Pending entries: bar_ts -> [sig, ...]  (queued to enter at that bar's open)
    pending_at = defaultdict(list)
    for sig in all_signals:
        sym = sig['sym']
        idx = sig.get('bar_idx')
        if idx is None:
            continue
        f = price_map[sym]
        if idx + 1 < len(f):
            next_bar_ts = f.index[idx + 1]
            pending_at[next_bar_ts].append(sig)
        # if no next bar (signal on very last bar) → skip

    # ── Step 3: portfolio state ──────────────────────────────────────
    equity       = INITIAL_EQUITY
    eq_curve     = [(pd.Timestamp.min, equity)]
    open_trades  = {}    # sym → trade dict
    trade_log    = []
    open_count   = 0
    tickers_open = set()

    # ── Helpers ──────────────────────────────────────────────────────
    def _apply_exit_slippage(px, trade_type, reason):
        """Add slippage on exit: adverse fill for SL/EOD, favorable capped for
TP."""
        if reason in ('StopLoss', 'StopLoss_Gap', 'EOD'):
            # Worse fill on SL/EOD exits
            return px * (1 + SLIPPAGE_PCT) if trade_type == 'LONG' else px * (1 -
SLIPPAGE_PCT)
        else:
            # TP: slippage reduces profit slightly
            return px * (1 - SLIPPAGE_PCT) if trade_type == 'LONG' else px * (1 +
SLIPPAGE_PCT)

    def _close_trade(tr, exit_px_raw, exit_ts, reason, eq_after):
        """Apply exit slippage + commission, then record trade."""
        exit_px = _apply_exit_slippage(exit_px_raw, tr['type'], reason)
        pnl_gross = ((exit_px - tr['entry_px']) if tr['type'] == 'LONG'
                     else (tr['entry_px'] - exit_px)) * tr['shares']
        commission_exit  = COMMISSION
        pnl_net  = pnl_gross - commission_exit
        eq_final = eq_after + pnl_net
        tr['ExitPrice']   = round(float(exit_px), 4)
        tr['ExitTime']    = exit_ts
        tr['Slippage$']   = round(
            abs(exit_px - exit_px_raw) * tr['shares'] +
            abs(tr['entry_px'] - tr['entry_px_raw']) * tr['shares'], 4)
        tr['Commission$'] = round(2 * COMMISSION, 4)   # round-trip stored on exit
        tr['TotalPnL']    = round(pnl_net, 2)
```

```
        tr['RMultiple']   = round(pnl_net / tr['risk_amt'], 4) if tr['risk_amt']
else 0
        tr['EquityAfter'] = round(eq_final, 2)
        tr['ExitReason']  = reason
        return eq_final

    # ── Step 4: main tick-by-tick simulation ──────────────────────────
    prev_bar_ts = None
    for bar_ts in global_timeline:

        # ── 4a: Check exits for ALL open trades at this bar ──────────
        to_remove = []
        for sym, tr in list(open_trades.items()):
            f = price_map[sym]
            if bar_ts not in ticker_bar_idx[sym]:
                continue                          # ticker has no bar here
            if bar_ts <= tr['EntryTime']:
                continue                          # entry bar itself, not eligible

            bar  = f.loc[bar_ts]
            op   = float(bar['Open'])
            hi   = float(bar['High'])
            lo   = float(bar['Low'])
            cl   = float(bar['Close'])
            hit  = False
            exit_px_raw = None
            reason      = None

            if tr['type'] == 'LONG':
                if op <= tr['sl']:                # GAP through stop on open
                    exit_px_raw, reason = op, 'StopLoss_Gap'
                elif op >= tr['tp']:              # GAP through target on open
                    exit_px_raw, reason = op, 'TakeProfit_Gap'
                elif lo <= tr['sl']:
                    exit_px_raw, reason = tr['sl'], 'StopLoss'
                elif hi >= tr['tp']:
                    exit_px_raw, reason = tr['tp'], 'TakeProfit'
                elif bar_ts.time() >= EOD_TIME:
                    exit_px_raw, reason = cl, 'EOD'
            else:  # SHORT
                if op >= tr['sl']:
                    exit_px_raw, reason = op, 'StopLoss_Gap'
                elif op <= tr['tp']:
                    exit_px_raw, reason = op, 'TakeProfit_Gap'
                elif hi >= tr['sl']:
                    exit_px_raw, reason = tr['sl'], 'StopLoss'
                elif lo <= tr['tp']:
                    exit_px_raw, reason = tr['tp'], 'TakeProfit'
                elif bar_ts.time() >= EOD_TIME:
                    exit_px_raw, reason = cl, 'EOD'
```

```
            if exit_px_raw is not None:
                eq_before_exit = equity
                equity = _close_trade(tr, exit_px_raw, bar_ts, reason, equity)
                eq_curve.append((bar_ts, equity))
                trade_log.append(tr)
                to_remove.append(sym)
                open_count -= 1
                hit = True

        for sym in to_remove:
            del open_trades[sym]; tickers_open.discard(sym)

        # ── 4b: Process pending entries at this bar's open ──────────────
        if bar_ts in pending_at:
            # Randomise order to avoid alphabetical bias for same-bar entries
            pending = pending_at[bar_ts][:]
            random.shuffle(pending)
            for sig in pending:
                sym = sig['sym']
                if sym in tickers_open:
                    continue
                if open_count >= MAX_CONCURRENT:
                    break                          # stop trying once slots full

                f = price_map[sym]
                if bar_ts not in ticker_bar_idx[sym]:
                    continue
                bar = f.loc[bar_ts]
                entry_px_raw = float(bar['Open'])

                # Apply entry slippage (adverse direction)
                entry_px = (entry_px_raw * (1 + SLIPPAGE_PCT) if sig['type'] ==
'LONG'
                            else entry_px_raw * (1 - SLIPPAGE_PCT))

                # Keep stop/target at same absolute PRICE LEVELS from signal,
                # recalculate stop_dist from actual slipped fill
                sl        = sig['sl']
                tp        = sig['tp']
                stop_dist = abs(entry_px - sl)
                if stop_dist < 0.01:
                    continue

                risk_amt = equity * BASE_RISK_PCT
                # Deduct entry commission from equity immediately
                equity  -= COMMISSION
                shares   = risk_amt / stop_dist

                if shares <= 0:
                    equity += COMMISSION   # refund if trade skipped
                    continue
```

```
                tr = {
                    'Ticker'        : sym,
                    'Type'          : sig['type'],
                    'Signal'        : sig['signal'],
                    'EntryTime'     : bar_ts,
                    'EntryPrice'    : round(entry_px, 4),
                    'StopLoss'      : round(sl, 4),
                    'Target'        : round(tp, 4),
                    'shares'        : shares,
                    'Shares'        : round(shares, 2),
                    'risk_amt'      : risk_amt,
                    'RiskAmt'       : round(risk_amt, 2),
                    'type'          : sig['type'],
                    'entry_px'      : entry_px,
                    'entry_px_raw'  : entry_px_raw,
                    'sl'            : sl,
                    'tp'            : tp,
                    'Slippage$'     : 0,
                    'Commission$'   : 0,
                    'ExitPrice'     : None,
                    'ExitTime'      : None,
                    'TotalPnL'      : None,
                    'RMultiple'     : None,
                    'EquityBefore'  : round(equity + COMMISSION, 2),  # before
entry commission
                    'EquityAfter'   : round(equity, 2),
                    'ExitReason'    : 'Open',
                }
                open_trades[sym] = tr
                tickers_open.add(sym)
                open_count += 1

        prev_bar_ts = bar_ts

    # ── Step 5: force-close any trades still open at final bar ──────────
    for sym, tr in list(open_trades.items()):
        f = price_map[sym]
        lp          = float(f['Close'].iat[-1])
        last_bar_ts = f.index[-1]
        equity = _close_trade(tr, lp, last_bar_ts, 'EOD', equity)
        eq_curve.append((last_bar_ts, equity))
        trade_log.append(tr)

    print(f"  ✓ {len(trade_log):,} trades executed | Final equity: ${equity:,.2f}")
    trades_df = pd.DataFrame(trade_log)
    return trades_df, eq_curve, equity


# ═══════════════════════════════════════════════════════════════════════
# METRICS
```

```
# ═══════════════════════════════════════════════════════════════

def compute_max_drawdown(eq_curve):
    """Returns (max_drawdown_$, max_drawdown_pct) from equity curve list of (ts,
eq)."""
    equities = np.array([e for _, e in eq_curve])
    peak = equities[0]; max_dd = 0.0
    for eq in equities:
        if eq > peak:
            peak = eq
        dd = peak - eq
        if dd > max_dd:
            max_dd = dd
    return max_dd


def portfolio_metrics(df, eq_curve, initial_equity=INITIAL_EQUITY):
    wins = df[df['TotalPnL'] > 0]
    loss = df[df['TotalPnL'] <= 0]
    r    = df['RMultiple'].values
    gp   = wins['TotalPnL'].sum()
    gl   = abs(loss['TotalPnL'].sum())
    pf   = gp / gl if gl else np.inf
    pnl  = df['TotalPnL'].values
    pnl_mean = pnl.mean(); pnl_std = pnl.std(ddof=1) if len(pnl) > 1 else 1e-9
    sh   = (pnl_mean / (pnl_std + 1e-9)) * np.sqrt(252) if len(pnl) > 1 else 0
    net_pnl = df['TotalPnL'].sum()
    max_dd  = compute_max_drawdown(eq_curve)
    avg_r   = r.mean() if len(r) else 0
    avg_risk = df['RiskAmt'].mean()  # average risk $ per trade
    max_dd_r = max_dd / avg_risk if avg_risk else 0
    net_pnl_r = net_pnl / avg_risk if avg_risk else 0

    return {
        'Total Trades'          : len(df),
        'Unique Tickers'        : int(df['Ticker'].nunique()),
        'LONG Trades'           : int((df['Type'] == 'LONG').sum()),
        'SHORT Trades'          : int((df['Type'] == 'SHORT').sum()),
        'Winning Trades'        : len(wins),
        'Losing Trades'         : len(loss),
        'Win Rate %'            : round(len(wins) / len(df) * 100, 2),
        'Expectancy (R)'        : round(float(avg_r), 4),
        'Total R'               : round(float(r.sum()), 4),
        'Std Dev R'             : round(float(r.std(ddof=1)), 4) if len(r) > 1 else
0,
        'Best Trade R'          : round(float(r.max()), 4),
        'Worst Trade R'         : round(float(r.min()), 4),
        'Gross Profit $'        : round(gp, 2),
        'Gross Loss $'          : round(-gl, 2),
        'Net P&L $'             : round(net_pnl, 2),
        'Profit Factor'         : round(pf, 4),
```

```
        'Avg Winner $'          : round(wins['TotalPnL'].mean(), 2) if len(wins)
else 0,
        'Avg Loser $'           : round(loss['TotalPnL'].mean(), 2) if len(loss)
else 0,
        'Sharpe Ratio'          : round(float(sh), 4),
        'Max Drawdown $'        : round(max_dd, 2),
        'Max Drawdown (R)'      : round(max_dd_r, 4),
        'Net P&L (R)'           : round(net_pnl_r, 4),
    }


def per_ticker_metrics(df):
    rows = []
    for sym, grp in df.groupby('Ticker'):
        w = grp[grp['TotalPnL'] > 0]; l = grp[grp['TotalPnL'] <= 0]
        r = grp['RMultiple'].values
        gp = w['TotalPnL'].sum(); gl = abs(l['TotalPnL'].sum())
        rows.append({'Ticker':sym, 'Trades':len(grp),
                     'Win Rate %':round(len(w)/len(grp)*100, 1),
                     'Expectancy R':round(r.mean(), 4), 'Total R':round(r.sum(),
4),
                     'Net P&L $':round(grp['TotalPnL'].sum(), 2),
                     'Profit Factor':round(gp/gl, 3) if gl else 999,
                     'Best Trade R':round(r.max(), 4)})
    return pd.DataFrame(rows).sort_values('Expectancy R', ascending=False)


def print_metrics(m):
    print(f"\n{'='*58}")
    print(f"  S&P 500 — TRADITIONAL PIVOT POINTS — PORTFOLIO SIM")
    print(f"  Shared $100k | Max {MAX_CONCURRENT} concurrent trades")
    print(f"{'='*58}")
    for k, v in m.items():
        print(f"  {k:<26}: {v}")
    print(f"{'='*58}\n")


# ═══════════════════════════════════════════════════════════════════
# CHARTS (7 panels)
# ═══════════════════════════════════════════════════════════════════

def plot_all(master_df, metrics, ticker_df, eq_curve):
    fig = plt.figure(figsize=(20, 16), facecolor='#0d1117')
    fig.suptitle(
        f'S&P 500  |  Traditional Pivot Points  |  15-Min  |  '
        f'Shared $100k Portfolio  |  Max {MAX_CONCURRENT} Concurrent Trades',
        color='white', fontsize=12, fontweight='bold', y=0.99)

    gs = gridspec.GridSpec(4, 3, figure=fig,
                           hspace=0.58, wspace=0.38,
                           left=0.06, right=0.97, top=0.95, bottom=0.05)
```

```
    def style(ax, title):
        ax.set_facecolor('#161b22')
        ax.set_title(title, color='white', fontsize=10, pad=6)
        ax.tick_params(colors='#8b949e')
        for sp in ax.spines.values(): sp.set_color('#30363d')
        ax.grid(alpha=0.12, color='#8b949e')

    # 1. Portfolio equity curve (full width)
    ax0 = fig.add_subplot(gs[0, :])
    style(ax0, 'Portfolio Equity Curve — Shared $100k | Max 10 Concurrent Trades')
    eq_ts  = [ts for ts, _ in eq_curve if ts != pd.Timestamp.min]
    eq_val = [eq for ts, eq in eq_curve if ts != pd.Timestamp.min]
    if eq_ts:
        ax0.plot(eq_ts, eq_val, color='#58a6ff', lw=1.2, alpha=0.9)
        ax0.fill_between(eq_ts, 100_000, eq_val,
                         where=[v >= 100_000 for v in eq_val],
                         color='#3fb950', alpha=0.15)
        ax0.fill_between(eq_ts, 100_000, eq_val,
                         where=[v < 100_000 for v in eq_val],
                         color='#da3633', alpha=0.15)
        ax0.axhline(100_000, color='white', lw=0.8, ls='--', alpha=0.4,
label='Start')
        final = eq_val[-1] if eq_val else 100_000
        ax0.axhline(final, color='#f0e040', lw=0.8, ls=':', alpha=0.6,
                    label=f'Final ${final:,.0f}')
    ax0.set_ylabel('Equity $', color='#8b949e')
    ax0.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, _: f'${x:,.0f}'))
    ax0.legend(facecolor='#161b22', labelcolor='white', fontsize=8)

    # 2. R-Multiple distribution
    ax1 = fig.add_subplot(gs[1, :])
    style(ax1, f'R-Multiple Distribution  |  {len(master_df):,} Trades  |  '
               f'Expectancy: {metrics["Expectancy (R)"]:.4f}R  |  '
               f'Win Rate: {metrics["Win Rate %"]}%  |  '
               f'Profit Factor: {metrics["Profit Factor"]}')
    rv   = master_df['RMultiple'].values
    bins = np.linspace(-1.2, 2.2, 50)
    ax1.hist(rv[rv > 0],  bins=bins, color='#3fb950', alpha=0.75,
             label=f'Winners ({(rv>0).sum():,})')
    ax1.hist(rv[rv <= 0], bins=bins, color='#da3633', alpha=0.75,
             label=f'Losers ({(rv<=0).sum():,})')
    ax1.axvline(0,          color='white',   lw=1,   alpha=0.5)
    ax1.axvline(rv.mean(),  color='#f0e040', lw=2,   ls='--',
                label=f'Mean={rv.mean():.4f}R')
    ax1.set_xlabel('R-Multiple', color='#8b949e')
    ax1.set_ylabel('Frequency',  color='#8b949e')
    ax1.legend(facecolor='#161b22', labelcolor='white', fontsize=9)

    # 3. Signal breakdown
    ax2 = fig.add_subplot(gs[2, 0])
    style(ax2, 'Signal Breakdown')
```

```
    sc = master_df['Signal'].value_counts()
    ax2.barh(sc.index, sc.values,
             color=['#58a6ff','#3fb950','#f0883e','#da3633'][:len(sc)], alpha=0.85)
    ax2.set_xlabel('# Trades', color='#8b949e')
    for i, v in enumerate(sc.values):
        ax2.text(v + 5, i, f'{v:,}', va='center', color='white', fontsize=8)

    # 4. Exit reasons
    ax3 = fig.add_subplot(gs[2, 1])
    style(ax3, 'Exit Reasons')
    ec   = master_df['ExitReason'].value_counts()
    ec_c = {'StopLoss':'#da3633', 'TakeProfit':'#3fb950', 'EOD':'#8b949e'}
    ax3.bar(ec.index, ec.values,
            color=[ec_c.get(x, '#58a6ff') for x in ec.index], alpha=0.85)
    ax3.set_ylabel('# Trades', color='#8b949e')
    for i, (x, v) in enumerate(zip(ec.index, ec.values)):
        ax3.text(i, v + 5, f'{v:,}', ha='center', color='white', fontsize=9)

    # 5. Top 20 tickers by expectancy
    ax4 = fig.add_subplot(gs[2, 2])
    style(ax4, 'Top 20 Tickers (Expectancy R)')
    top20 = ticker_df.head(20)
    c4 = ['#3fb950' if e > 0 else '#da3633' for e in top20['Expectancy R']]
    ax4.barh(top20['Ticker'][::-1], top20['Expectancy R'][::-1],
             color=c4[::-1], alpha=0.85)
    ax4.axvline(0, color='white', lw=0.8, alpha=0.5)
    ax4.set_xlabel('Expectancy (R)', color='#8b949e')

    # 6. Long vs Short R-distribution
    ax5 = fig.add_subplot(gs[3, 0])
    style(ax5, 'Long vs Short R-Distribution')
    ln = master_df[master_df['Type'] == 'LONG']['RMultiple'].values
    sh = master_df[master_df['Type'] == 'SHORT']['RMultiple'].values
    b2 = np.linspace(-1.2, 2.2, 40)
    if len(ln): ax5.hist(ln, bins=b2, color='#3fb950', alpha=0.6,
                         label=f'LONG ({len(ln):,})  E={ln.mean():.3f}R')
    if len(sh): ax5.hist(sh, bins=b2, color='#da3633', alpha=0.6,
                         label=f'SHORT ({len(sh):,})  E={sh.mean():.3f}R')
    ax5.set_xlabel('R-Multiple', color='#8b949e')
    ax5.legend(facecolor='#161b22', labelcolor='white', fontsize=8)

    # 7. Top 30 tickers net P&L
    ax6 = fig.add_subplot(gs[3, 1])
    style(ax6, 'Net P&L — Top 30 Tickers')
    t30 = ticker_df.nlargest(30, 'Net P&L $')
    c6  = ['#3fb950' if v > 0 else '#da3633' for v in t30['Net P&L $']]
    ax6.bar(range(len(t30)), t30['Net P&L $'].values, color=c6, alpha=0.85)
    ax6.set_xticks(range(len(t30)))
    ax6.set_xticklabels(t30['Ticker'].values, rotation=90, fontsize=6,
color='#8b949e')
    ax6.set_ylabel('Net P&L $', color='#8b949e')
```

```
    ax6.yaxis.set_major_formatter(plt.FuncFormatter(lambda x, _: f'${x:,.0f}'))

    # 8. Win rate distribution
    ax7 = fig.add_subplot(gs[3, 2])
    style(ax7, 'Win Rate Distribution (per Ticker)')
    wr = ticker_df['Win Rate %'].values
    ax7.hist(wr, bins=30, color='#58a6ff', alpha=0.85, edgecolor='#30363d')
    ax7.axvline(wr.mean(), color='#f0e040', lw=1.5, ls='--',
                label=f'Mean={wr.mean():.1f}%')
    ax7.axvline(50, color='white', lw=1, ls=':', alpha=0.5, label='50% line')
    ax7.set_xlabel('Win Rate %', color='#8b949e')
    ax7.set_ylabel('# Tickers',  color='#8b949e')
    ax7.legend(facecolor='#161b22', labelcolor='white', fontsize=8)

    fig.savefig('outputppportfolio8/strategy_performance.png', dpi=150,
                bbox_inches='tight', facecolor='#0d1117')
    plt.close(fig)
    print("  ✓ Chart → outputppportfolio8/strategy_performance.png")

# ═══════════════════════════════════════════════════════════════
# EXCEL EXPORT
# ═══════════════════════════════════════════════════════════════

def export_excel(master_df, metrics, ticker_df):
    path = "outputppportfolio8/pivot_portfolio_journal.xlsx"
    wb   = openpyxl.Workbook()

    # Style constants
    HDR_FILL  = PatternFill("solid", fgColor="1F4E79")
    HDR_FONT  = Font(bold=True, color="FFFFFF", size=11, name="Calibri")
    WIN_FILL  = PatternFill("solid", fgColor="E2EFDA")
    LOSS_FILL = PatternFill("solid", fgColor="FADADD")
    SEC_FILL  = PatternFill("solid", fgColor="2F5496")
    SEC_FONT  = Font(bold=True, size=11, color="FFFFFF", name="Calibri")
    LBL_FONT  = Font(bold=True, name="Calibri", size=11)
    VAL_FONT  = Font(name="Calibri", size=11)
    CENTER    = Alignment(horizontal='center', vertical='center')
    LEFT      = Alignment(horizontal='left',   vertical='center', indent=1)
    BORDER    = Border(left=Side(style='thin'),  right=Side(style='thin'),
                       top=Side(style='thin'),   bottom=Side(style='thin'))

    def banner(ws, text, end_col="O"):
        ws.merge_cells(f"A1:{end_col}1")
        c = ws["A1"]
        c.value = text
        c.font  = Font(bold=True, size=13, color="FFFFFF", name="Calibri")
        c.fill  = HDR_FILL
        c.alignment = CENTER
        ws.row_dimensions[1].height = 28
        ws.sheet_view.showGridLines = False
```

```
    def add_headers(ws, hdrs, row=2):
        for j, h in enumerate(hdrs, 1):
            c = ws.cell(row, j, h)
            c.fill = HDR_FILL; c.font = HDR_FONT
            c.alignment = CENTER; c.border = BORDER
        ws.row_dimensions[row].height = 20

    # ── Sheet 1: Trade Journal ──────────────────────────────────────
    ws1 = wb.active; ws1.title = "Trade Journal"
    banner(ws1, f"S&P 500 — Pivot Points Portfolio v8  |  Shared $100k  |  "
                f"Max {MAX_CONCURRENT} Concurrent  |  Slip={SLIPPAGE_PCT*100:.2f}%
|  Comm=${COMMISSION:.0f}/side  |  {len(master_df):,} Trades")
    hdrs = ['#','Ticker','Type','Signal','Entry Time','Entry $','Stop $',
            'Target $','Shares','Risk $','Slip $','Comm $','Exit $','Exit Time',
            'P&L $','R-Multiple','Equity Before','Equity After','Exit Reason']
    add_headers(ws1, hdrs)
    col_map = ['Ticker','Type','Signal','EntryTime','EntryPrice','StopLoss',
               'Target','Shares','RiskAmt','Slippage$','Commission$',
               'ExitPrice','ExitTime',
               'TotalPnL','RMultiple','EquityBefore','EquityAfter','ExitReason']
    num_fmts = {'EntryPrice':'$#,##0.00', 'StopLoss':'$#,##0.00',
                'Target':'$#,##0.00', 'ExitPrice':'$#,##0.00',
                'Shares':'#,##0.00', 'RiskAmt':'$#,##0.00',
                'Slippage$':'$#,##0.00', 'Commission$':'$#,##0.00',
                'TotalPnL':'$#,##0.00', 'RMultiple':'0.0000',
                'EquityBefore':'$#,##0.00', 'EquityAfter':'$#,##0.00'}

    for rn, (_, row) in enumerate(master_df.iterrows(), 3):
        pnl_val = row.get('TotalPnL', 0) or 0
        fill = WIN_FILL if pnl_val > 0 else LOSS_FILL
        ws1.cell(rn, 1, rn - 2).fill = fill
        ws1.cell(rn, 1).alignment = CENTER
        ws1.cell(rn, 1).border = BORDER
        for j, col in enumerate(col_map, 2):
            val = row.get(col, '')
            if isinstance(val, float) and pd.isna(val): val = ''
            c = ws1.cell(rn, j, value=val)
            c.fill = fill; c.alignment = CENTER; c.border = BORDER
            c.font = VAL_FONT
            if col in num_fmts:
                c.number_format = num_fmts[col]

    for i, w in enumerate([5,8,7,14,20,20,12,12,10,10,11,11,12,20,12,12,14,14,14],
1):
        ws1.column_dimensions[get_column_letter(i)].width = w
    ws1.freeze_panes = 'A3'

    # ── Sheet 2: Portfolio Metrics ──────────────────────────────────
    ws2 = wb.create_sheet("Portfolio Metrics")
    banner(ws2, "Portfolio Metrics — Shared $100k | Van K. Tharp Framework", "D")
    ws2.sheet_view.showGridLines = False
```

```
    # Section groupings for metrics
    sections = {
        "TRADE COUNT": ["Total Trades","Unique Tickers","LONG Trades","SHORT
Trades",
                        "Winning Trades","Losing Trades"],
        "R-MULTIPLE ANALYSIS": ["Win Rate %","Expectancy (R)","Total R","Std Dev
R",
                                "Best Trade R","Worst Trade R","Net P&L (R)","Max
Drawdown (R)"],
        "P&L ANALYSIS": ["Gross Profit $","Gross Loss $","Net P&L $",
                         "Avg Winner $","Avg Loser $","Profit Factor"],
        "RISK METRICS": ["Sharpe Ratio","Max Drawdown $"],
    }

    row = 2
    for sec_name, keys in sections.items():
        # Section header
        ws2.merge_cells(f"B{row}:D{row}")
        c = ws2.cell(row, 2, sec_name)
        c.fill = SEC_FILL; c.font = SEC_FONT; c.alignment = CENTER
        ws2.row_dimensions[row].height = 20
        row += 1
        for k in keys:
            v = metrics.get(k, 'N/A')
            ck = ws2.cell(row, 2, k)
            ck.font = LBL_FONT; ck.border = BORDER; ck.alignment = LEFT
            cv = ws2.cell(row, 3, v)
            cv.font = VAL_FONT; cv.border = BORDER; cv.alignment = CENTER
            if isinstance(v, (int, float)) and not isinstance(v, bool):
                if '$' in k:
                    cv.number_format = '$#,##0.00'
                elif '%' in k:
                    cv.number_format = '0.00"%"' if isinstance(v, float) else '0'
                elif 'R)' in k or '(R' in k or k in ('Total R','Std Dev R','Best
Trade R',
                                                     'Worst Trade
R','Expectancy (R)'):
                    cv.number_format = '0.0000'
                else:
                    cv.number_format = '#,##0.00'
            if k in ('Win Rate %','Expectancy (R)','Profit Factor','Net P&L $',
                     'Sharpe Ratio','Net P&L (R)'):
                cv.fill = WIN_FILL if (isinstance(v, (int,float)) and v >= 0) else
LOSS_FILL
            if k in ('Max Drawdown $','Max Drawdown (R)','Gross Loss $','Worst
Trade R'):
                cv.fill = LOSS_FILL
            row += 1
        row += 1  # spacing between sections
```

```
    ws2.column_dimensions['A'].width = 3
    ws2.column_dimensions['B'].width = 28
    ws2.column_dimensions['C'].width = 20
    ws2.column_dimensions['D'].width = 5

    # ── Sheet 3: Per-Ticker Leaderboard ──────────────────────────────
    ws3 = wb.create_sheet("Per-Ticker Leaderboard")
    banner(ws3, "Per-Ticker Leaderboard — Sorted by Expectancy R", "I")
    t_hdrs = ['Ticker','Trades','Win Rate %','Expectancy R',
              'Total R','Net P&L $','Profit Factor','Best Trade R']
    add_headers(ws3, t_hdrs)
    for rn, (_, row) in enumerate(ticker_df.iterrows(), 3):
        fill = WIN_FILL if row['Expectancy R'] > 0 else LOSS_FILL
        for j, col in enumerate(t_hdrs, 1):
            val = row.get(col, '')
            c = ws3.cell(rn, j, val)
            c.fill = fill; c.alignment = CENTER; c.border = BORDER; c.font =
VAL_FONT
            if col == 'Net P&L $': c.number_format = '$#,##0.00'
            elif col != 'Ticker': c.number_format = '#,##0.00'
    for i, w in enumerate([10,9,13,15,13,15,15,15], 1):
        ws3.column_dimensions[get_column_letter(i)].width = w
    ws3.freeze_panes = 'A3'

    # ── Sheet 4: Strategy Checklist ──────────────────────────────────
    ws4 = wb.create_sheet("Strategy Checklist")
    banner(ws4, "TRADITIONAL PIVOT POINTS — Strategy Checklist v8.0", "E")
    ws4.sheet_view.showGridLines = False

    checklist = [
        ("PORTFOLIO ARCHITECTURE", [
            f"  Portfolio Capital     : $100,000 (shared across all trades)",
            f"  Max Concurrent Trades : {MAX_CONCURRENT} (enforced globally)",
            f"  Risk per Trade        : {BASE_RISK_PCT*100:.1f}% of current
portfolio equity",
            "  Position Size         : Risk $ ÷ Stop Distance (shares)",
            "  One trade per ticker at any given time",
            f"  Slippage              : {SLIPPAGE_PCT*100:.2f}% of fill price (each
side)",
            f"  Commission            : ${COMMISSION:.2f} flat per side
(${2*COMMISSION:.2f} round-trip)",
        ]),
        ("PIVOT LEVEL FORMULAS (Traditional, Daily resolution)", [
            "  P  = (prev_High + prev_Low + prev_Close) / 3",
            "  R1 = P×2 − prev_Low              S1 = P×2 − prev_High",
            "  R2 = P + (prev_High − prev_Low)  S2 = P − (prev_High − prev_Low)",
            "  R3 = P×2 + (prev_High − 2×prev_Low)",
        ]),
        ("LONG ENTRY SIGNALS (15-min bar)", [
            "  □ BounceS1    : prior 15-min bar Low ≤ S1  AND  current Close > S1",
            "  □ BreakP_Long : prior 15-min Close < P     AND  current Close > P",
```

```
        ]),
        ("SHORT ENTRY SIGNALS (15-min bar)", [
            "  □ RejectR1     : prior 15-min bar High ≥ R1  AND  current Close <
R1",
            "  □ BreakP_Short : prior 15-min Close > P      AND  current Close <
P",
        ]),
        ("EXIT RULES", [
            f"  □ Entry         : Next bar OPEN after signal fires (+ slippage)",
            f"  □ Stop Loss     : Original signal ATR-level; gap-fills at bar
open",
            f"  □ Take Profit   : Original signal ATR-level; gap-fills at bar
open",
            f"  □ Tick Exits    : Every 15-min bar checked for SL/TP/EOD",
            "  □ EOD close     : 15:45 force-exit — no overnight holds",
            "  □ Exit reasons  : StopLoss | TakeProfit | StopLoss_Gap |
TakeProfit_Gap | EOD",
        ]),
    ]

    row = 3
    for sec, items in checklist:
        ws4.merge_cells(f"B{row}:E{row}")
        ws4[f"B{row}"] = sec
        ws4[f"B{row}"].font = SEC_FONT
        ws4[f"B{row}"].fill = SEC_FILL
        ws4[f"B{row}"].alignment = LEFT
        ws4.row_dimensions[row].height = 20
        row += 1
        for item in items:
            ws4.merge_cells(f"B{row}:E{row}")
            ws4[f"B{row}"] = item
            ws4[f"B{row}"].font = Font(size=10, name="Calibri")
            ws4[f"B{row}"].alignment = LEFT
            ws4.row_dimensions[row].height = 16
            row += 1
        row += 1

    ws4.column_dimensions['A'].width = 3
    ws4.column_dimensions['B'].width = 55
    ws4.column_dimensions['C'].width = 5

    # ── Sheet 5: Drawdown Risk Scenarios ──────────────────────────────
    ws5 = wb.create_sheet("Drawdown Scenarios")
    ws5.sheet_view.showGridLines = False
    banner(ws5, "Drawdown Risk Scenario Analysis — $100,000 Portfolio", "N")

    SCENARIO_RISKS  = [0.005, 0.01, 0.02, 0.03, 0.05, 0.10, 0.25, 0.50, 1.00]
    SCENARIO_LABELS = ["0.5%", "1%", "2%", "3%", "5%", "10%", "25%", "50%", "100%"]
    SCEN_COLORS     = ["2ECC71","27AE60","F1C40F","E67E22","E74C3C",
                       "C0392B","8E44AD","6C3483","9B59B6"]
```

```
    r_vals = master_df['RMultiple'].dropna().values

    def _sim_scenario(risk_pct):
        eq, peak = INITIAL_EQUITY, INITIAL_EQUITY
        eq_curve_s, dd_curve_s, pnl_s = [eq], [0.0], []
        for r in r_vals:
            ra   = eq * risk_pct
            pnl  = ra * r
            eq  += pnl
            pnl_s.append(pnl)
            if eq > peak: peak = eq
            dd_curve_s.append((peak - eq) / peak * 100)
            eq_curve_s.append(eq)
        pnl_arr  = np.array(pnl_s)
        avg_risk = np.mean([INITIAL_EQUITY * risk_pct] * len(r_vals))
        max_dd_p = max(dd_curve_s)
        max_dd_d = max_dd_p / 100 * np.array(eq_curve_s).max()
        net_pnl  = eq - INITIAL_EQUITY
        sharpe   = (pnl_arr.mean()/(pnl_arr.std(ddof=1)+1e-9))*np.sqrt(252) if
len(pnl_arr)>1 else 0
        return {
            'eq': eq_curve_s, 'dd': dd_curve_s,
            'final': eq, 'net_pnl': net_pnl,
            'net_pct': net_pnl/INITIAL_EQUITY*100,
            'max_dd_pct': max_dd_p, 'max_dd_d': max_dd_d,
            'max_dd_r': max_dd_d / avg_risk if avg_risk else 0,
            'net_r': net_pnl / avg_risk if avg_risk else 0,
            'sharpe': sharpe,
        }

    scen = {lbl: _sim_scenario(rp) for lbl, rp in zip(SCENARIO_LABELS,
SCENARIO_RISKS)}

    # ── Summary table ──────────────────────────────────────────────
    SUM_HDRS = ["Risk %","Final Equity","Net P&L $","Net P&L %",
                "Max DD $","Max DD % (peak)","Max DD (R)","Net P&L
(R)","Sharpe","Outcome"]
    row = 3
    ws5.merge_cells(f"B{row}:K{row}")
    c = ws5.cell(row, 2, "Scenario Summary"); c.fill = SEC_FILL; c.font = SEC_FONT
    c.alignment = CENTER; ws5.row_dimensions[row].height = 20; row += 1
    for j, h in enumerate(SUM_HDRS, 2):
        c = ws5.cell(row, j, h); c.fill = HDR_FILL; c.font = HDR_FONT
        c.alignment = CENTER; c.border = BORDER
    ws5.row_dimensions[row].height = 20; row += 1

    for lbl, col_hex in zip(SCENARIO_LABELS, SCEN_COLORS):
        s = scen[lbl]
        fin = s['final']
```

```
        outcome = "RUINED" if fin < 1000 else ("⚠️ EXTREME" if s['max_dd_pct']>90
else
                  ("⚠️ HIGH" if s['max_dd_pct']>60 else "✅ VIABLE"))
        rfill = LOSS_FILL if fin < 1000 else (PatternFill("solid",
fgColor="FFF2CC")
                  if s['max_dd_pct'] > 60 else WIN_FILL)
        vals = [lbl, f"${fin:,.0f}", f"${s['net_pnl']:,.0f}",
f"{s['net_pct']:.1f}%",
                  f"${s['max_dd_d']:,.0f}", f"{s['max_dd_pct']:.1f}%",
                  f"{s['max_dd_r']:.1f}R", f"{s['net_r']:.1f}R",
                  f"{s['sharpe']:.3f}", outcome]
        for j, val in enumerate(vals, 2):
            c = ws5.cell(row, j, val); c.fill = rfill
            c.font = Font(name="Calibri", size=10,
                          bold=(outcome in ["RUINED","✅ VIABLE","⚠️ EXTREME","⚠️
HIGH"]),
                          color="FFFFFF" if outcome == "RUINED" else "000000")
            c.alignment = CENTER; c.border = BORDER
        row += 1
    row += 2

    # ── Equity curve data table ──────────────────────────────────────
    ws5.merge_cells(f"B{row}:K{row}")
    c = ws5.cell(row, 2, "Equity Curve by Risk Level"); c.fill = SEC_FILL
    c.font = SEC_FONT; c.alignment = CENTER; ws5.row_dimensions[row].height = 20;
row += 1
    ws5.cell(row, 2, "Trade #").fill = HDR_FILL; ws5.cell(row, 2).font = HDR_FONT
    ws5.cell(row, 2).alignment = CENTER
    for j, (lbl, col_hex) in enumerate(zip(SCENARIO_LABELS, SCEN_COLORS), 3):
        c = ws5.cell(row, j, lbl)
        c.fill = PatternFill("solid", fgColor=col_hex)
        c.font = Font(bold=True, color="FFFFFF" if j>4 else "000000", size=10,
name="Calibri")
        c.alignment = CENTER; c.border = BORDER
    ws5.row_dimensions[row].height = 20
    eq_hdr_row = row; row += 1
    n_pts  = len(r_vals)
    step   = max(1, n_pts // 500)
    eq_rows = 0
    for i in range(0, n_pts+1, step):
        ws5.cell(row, 2, i).number_format = "#,##0"; ws5.cell(row,2).alignment =
CENTER
        for j, lbl in enumerate(SCENARIO_LABELS, 3):
            eq_list = scen[lbl]['eq']
            val = eq_list[i] if i < len(eq_list) else eq_list[-1]
            c = ws5.cell(row, j, round(val, 2)); c.number_format = '$#,##0'
            c.alignment = CENTER
        row += 1; eq_rows += 1
    row += 2

    # ── Drawdown data table ──────────────────────────────────────────
```

```
    ws5.merge_cells(f"B{row}:K{row}")
    c = ws5.cell(row, 2, "Drawdown from Peak (%) by Risk Level"); c.fill = SEC_FILL
    c.font = SEC_FONT; c.alignment = CENTER; ws5.row_dimensions[row].height = 20;
row += 1
    ws5.cell(row, 2, "Trade #").fill = HDR_FILL; ws5.cell(row, 2).font = HDR_FONT
    ws5.cell(row, 2).alignment = CENTER
    for j, (lbl, col_hex) in enumerate(zip(SCENARIO_LABELS, SCEN_COLORS), 3):
        c = ws5.cell(row, j, lbl)
        c.fill = PatternFill("solid", fgColor=col_hex)
        c.font = Font(bold=True, color="FFFFFF" if j>4 else "000000", size=10,
name="Calibri")
        c.alignment = CENTER; c.border = BORDER
    ws5.row_dimensions[row].height = 20
    dd_hdr_row = row; row += 1
    dd_rows = 0
    for i in range(0, n_pts+1, step):
        ws5.cell(row, 2, i).number_format = "#,##0"; ws5.cell(row,2).alignment =
CENTER
        for j, lbl in enumerate(SCENARIO_LABELS, 3):
            dd_list = scen[lbl]['dd']
            val = dd_list[i] if i < len(dd_list) else dd_list[-1]
            c = ws5.cell(row, j, round(-val, 4)); c.number_format = '0.00"%"'
            c.alignment = CENTER
            if val > 50: c.fill = PatternFill("solid", fgColor="FADADD")
        row += 1; dd_rows += 1

    # ── Embed charts ──────────────────────────────────────────────
    from openpyxl.chart import LineChart, Reference as ChartRef
    chart_eq2 = LineChart()
    chart_eq2.title = "Portfolio Equity Curve — Risk Scenarios"
    chart_eq2.style = 10; chart_eq2.height = 14; chart_eq2.width = 28
    chart_eq2.y_axis.title = "Equity ($)"; chart_eq2.x_axis.title = "Trade #"
    chart_eq2.add_data(
        ChartRef(ws5, min_col=3, max_col=11,
                 min_row=eq_hdr_row, max_row=eq_hdr_row+eq_rows),
        titles_from_data=True)
    chart_dd2 = LineChart()
    chart_dd2.title = "Drawdown from Peak (%) — Risk Scenarios"
    chart_dd2.style = 10; chart_dd2.height = 14; chart_dd2.width = 28
    chart_dd2.y_axis.title = "Drawdown (%)"; chart_dd2.x_axis.title = "Trade #"
    chart_dd2.add_data(
        ChartRef(ws5, min_col=3, max_col=11,
                 min_row=dd_hdr_row, max_row=dd_hdr_row+dd_rows),
        titles_from_data=True)
    ws5.add_chart(chart_eq2, "M3")
    ws5.add_chart(chart_dd2, "M45")

    for i, w in enumerate([3,10,13,13,13,13,13,13,13,13,13], 1):
        ws5.column_dimensions[get_column_letter(i)].width = w

    wb.save(path)
```

```
    print(f"  ✓ Excel → {path}")


# ═══════════════════════════════════════════════════════════════════════
# MAIN
# ═══════════════════════════════════════════════════════════════════════

if __name__ == "__main__":
    print("=" * 68)
    print("  S&P 500  |  TRADITIONAL PIVOT POINTS  |  PORTFOLIO SIM  |  v8.0")
    print(f"  Portfolio: ${INITIAL_EQUITY:,}  |  Risk/Trade:
{BASE_RISK_PCT*100:.1f}%  |  "
          f"Max Concurrent: {MAX_CONCURRENT}")
    print("=" * 68)

    # 1. Get tickers
    sp500 = get_sp500_tickers()

    # 2. Fetch all raw data in parallel
    print(f"\n[FETCH] {len(sp500)} tickers | {MAX_WORKERS} threads...")
    ticker_data = {}; failed = []; t0 = time.time()

    # Shuffle so rate-limit failures are random, not confined to letters D-Z
    sp500_shuffled = sp500.copy()
    random.shuffle(sp500_shuffled)

    BATCH_SIZE  = 50     # process 50 tickers per batch
    BATCH_PAUSE = 2.0    # seconds between batches

    done = 0
    for batch_start in range(0, len(sp500_shuffled), BATCH_SIZE):
        batch = sp500_shuffled[batch_start : batch_start + BATCH_SIZE]
        with ThreadPoolExecutor(max_workers=MAX_WORKERS) as exe:
            futs = {exe.submit(fetch_ticker, sym): sym for sym in batch}
            for fut in as_completed(futs):
                sym, daily_df, fifteen_df = fut.result(); done += 1
                if daily_df is not None:
                    ticker_data[sym] = (daily_df, fifteen_df)
                else:
                    failed.append(sym)
        print(f"  [{done:>3}/{len(sp500_shuffled)}] loaded: {len(ticker_data):>3} |
"
              f"failed: {len(failed):>3} | elapsed: {time.time()-t0:.0f}s")
        if batch_start + BATCH_SIZE < len(sp500_shuffled):
            time.sleep(BATCH_PAUSE)   # rate-limit pause between batches

    # Second-pass retry for any failed tickers
    if failed:
        print(f"\n[RETRY] {len(failed)} failed tickers — pausing 10s then
retrying...")
        time.sleep(10)
```

```
        retry_still_failed = []
        with ThreadPoolExecutor(max_workers=2) as exe:
            futs = {exe.submit(fetch_ticker, sym): sym for sym in failed}
            for fut in as_completed(futs):
                sym, daily_df, fifteen_df = fut.result()
                if daily_df is not None:
                    ticker_data[sym] = (daily_df, fifteen_df)
                    print(f"    ✓ Recovered {sym}")
                else:
                    retry_still_failed.append(sym)
        failed = retry_still_failed
        print(f"  Retry complete — loaded: {len(ticker_data)} | "
              f"permanently failed: {len(failed)}")

    print(f"\n✓ Data fetched: {len(ticker_data)} tickers in {time.time()-t0:.0f}s")

    # 3. Run portfolio-level simulation
    master, eq_curve, final_equity = run_portfolio_simulation(ticker_data)
    master = master.sort_values('EntryTime').reset_index(drop=True)
    print(f"  Master journal: {len(master):,} trades | "
          f"{master['Ticker'].nunique()} tickers | Final equity:
${final_equity:,.2f}")

    # 4. Metrics
    metrics   = portfolio_metrics(master, eq_curve)
    ticker_df = per_ticker_metrics(master)
    print_metrics(metrics)

    # 5. Export
    print("[EXPORT] Saving outputs...")
    export_excel(master, metrics, ticker_df)
    plot_all(master, metrics, ticker_df, eq_curve)

    print("\n✅ ALL DONE — outputs in outputppportfolio8/")
    print("   pivot_portfolio_journal.xlsx")
    print("   strategy_performance.png")
    print("=" * 68)
```

C.5: Sensitivity analysis code that is used for all four strategies

```
"""
╔══════════════════════════════════════════════════════════════════╗
║     SENSITIVITY MATRIX GENERATOR — Risk % × Slot Count           ║
║     Based on actual R-multiple trade journal                     ║
║                                                                  ║
║     WHAT THIS SCRIPT DOES:                                       ║
║     Takes a trade journal (Excel) and replays every trade's R-multiple  ║
║     under different risk % and slot count assumptions.           ║
║     Produces 4 matrices + equity curves + full annotated Excel workbook.  ║
║                                                                  ║
```

```
╔═══════════════════════════════════════════════════════════════════════╗
║     USAGE:                                                            ║
║         python sensitivity_matrix.py                                 ║
║     Then upload your trade journal when prompted, or set JOURNAL_PATH below. ║
╚═══════════════════════════════════════════════════════════════════════╝
"""

# ═══════════════════════════════════════════════════════════════════════
# 0. IMPORTS
# ═══════════════════════════════════════════════════════════════════════
import os, sys, math, warnings
import numpy as np
import pandas as pd
import openpyxl
from openpyxl.styles import (PatternFill, Font, Alignment, Border, Side,
                             GradientFill)
from openpyxl.formatting.rule import ColorScaleRule, DataBarRule
from openpyxl.utils import get_column_letter
warnings.filterwarnings('ignore')

# ═══════════════════════════════════════════════════════════════════════
# 1. CONFIGURATION — edit these to customise the analysis
# ═══════════════════════════════════════════════════════════════════════

# Path to your trade journal.
# Accepts .xlsx  The script auto-detects the header row and R-Multiple column.
JOURNAL_PATH = "outputppportfolio7/pivot_portfolio_journal.xlsx"

# Starting portfolio capital ($)
INITIAL_EQUITY = 100_000.0

# Risk % levels to test  (as decimals: 0.01 = 1%)
RISK_LEVELS = [0.001, 0.002, 0.003, 0.005, 0.0075,
               0.01, 0.015, 0.02, 0.025, 0.03, 0.04, 0.05,0.1, 0.15, 0.2, 0.25,
0.3, 0.4, 0.5]

# Max-concurrent-slot counts to test
SLOT_OPTIONS = [1, 2, 3, 5, 7, 10, 15, 20]

# Output file path
OUTPUT_PATH = "outputppportfolio7/pivot_portfolio_sensitivity_matrix.xlsx"

# ═══════════════════════════════════════════════════════════════════════
# 2. LOAD & PARSE THE TRADE JOURNAL
# ═══════════════════════════════════════════════════════════════════════

def load_journal(path: str) -> pd.DataFrame:
    """
    LOGIC:
    ──────
    We need just one column from the journal: the R-Multiple column.
    An R-Multiple is defined as:
```

```
        R = (Trade P&L $) / (Initial Risk $ for that trade)

    It is dimensionless — it tells you how many "units of risk" you won or lost.
      • R = +3.0  → won 3× your risk (hit the 1:3 target)
      • R = -1.0  → lost exactly your risk amount (hit stop loss)
      • R = -0.5  → exited early, lost half of risk

    Because R is dimensionless, we can replay the same sequence of R values
    under ANY starting equity or risk% and the relative outcomes are valid.
    This is the power of Van Tharp's R-multiple framework.

    Steps:
      1. Read the workbook and auto-detect the real header row
         (some journals have a banner in row 1, real headers in row 2).
      2. Find the R-Multiple column by name.
      3. Parse numeric values, drop NaN rows.
      4. Optionally filter to a date range if EntryTime column is present.
    """
    print(f"\n[LOAD] Reading: {path}")

    # Read all sheets, find the one with trade data
    all_sheets = pd.read_excel(path, sheet_name=None, header=None)
    trade_sheet = None
    for sheet_name, raw in all_sheets.items():
        # Look for 'R-Multiple' anywhere in the first 5 rows
        for row_i in range(min(5, len(raw))):
            row_vals = [str(v).strip().lower() for v in raw.iloc[row_i].values
                        if pd.notna(v)]
            if any('r-multiple' in v or 'rmultiple' in v or 'r multiple' in v
                   for v in row_vals):
                trade_sheet = sheet_name
                header_row = row_i
                break
        if trade_sheet:
            break

    if trade_sheet is None:
        # Fallback: first sheet, row 1 as header
        trade_sheet = list(all_sheets.keys())[0]
        header_row = 1
        print(f"  ⚠ Could not auto-detect header. Using sheet='{trade_sheet}', row
1.")
    else:
        print(f"  ✓ Found trade data in sheet='{trade_sheet}', header at row
{header_row}")

    df = pd.read_excel(path, sheet_name=trade_sheet, header=header_row)
    df.columns = [str(c).strip() for c in df.columns]

    # Find R-Multiple column (case-insensitive)
```

```
    r_col = next((c for c in df.columns
                  if 'r-multiple' in c.lower() or 'rmultiple' in c.lower()
                  or 'r multiple' in c.lower()), None)
    if r_col is None:
        raise ValueError(
            "R-Multiple column not found. Expected a column named "
            "'R-Multiple', 'RMultiple', or 'R Multiple'.")

    # Find optional columns
    pnl_col   = next((c for c in df.columns if 'p&l' in c.lower() or 'pnl' in
c.lower()), None)
    risk_col  = next((c for c in df.columns if 'risk' in c.lower()), None)
    time_col  = next((c for c in df.columns
                      if 'entry' in c.lower() and 'time' in c.lower()), None)

    df[r_col] = pd.to_numeric(df[r_col], errors='coerce')
    df = df.dropna(subset=[r_col]).reset_index(drop=True)

    print(f"  ✓ {len(df):,} valid trades loaded")
    print(f"  ✓ R-Multiple column: '{r_col}'")
    if r_col:
        r = df[r_col].values
        wins  = (r > 0).sum()
        print(f"  ✓ R stats → mean={r.mean():.4f}  std={r.std():.4f}  "
              f"min={r.min():.2f}  max={r.max():.2f}")
        print(f"  ✓ Win rate: {wins/len(r)*100:.1f}%  "
              f"({wins} wins / {len(r)-wins} losses)")

    return df, r_col


# ═══════════════════════════════════════════════════════════════════════
# 3. THE CORE SIMULATION ENGINE
# ═══════════════════════════════════════════════════════════════════════

def simulate_path(r_series: np.ndarray,
                  risk_pct: float,
                  slots: int,
                  initial: float = 100_000.0):
    """
    WHAT IS BEING SIMULATED:
    ─────────────────────────
    We replay the trade journal's R-multiples in order, as if we were running
    this exact strategy with the given (risk_pct, slots) configuration.

    HOW POSITION SIZING WORKS (Van Tharp CPR — Constant Percentage Risk):
    ──────────────────────────────────────────────────────────────────────
    At the moment a trade opens, we decide position size so that if the trade
    hits its stop loss, we lose exactly (equity × risk_pct) dollars.

        slot_risk ($) = current_equity × risk_pct
```

```
    Example: equity=$100,000, risk=1%
        slot_risk = $100,000 × 0.01 = $1,000

The number of shares = slot_risk / stop_distance_in_dollars
(stop_distance = ATR × multiplier, from the strategy config)

WHY EQUITY-PERCENTAGE RISK:
──────────────────────────
• As equity grows, risk_$ grows → position sizes scale up → compounding.
• As equity falls, risk_$ falls → position sizes shrink → auto-protection.
• This is why equity curves at high risk% curve exponentially upward —
  winners compound on a larger base.

HOW R-MULTIPLE TRANSLATES TO P&L:
────────────────────────────────
P&L_i = slot_risk_i × R_i

Where R_i is from the journal.  Examples:
    slot_risk=$1,000, R=+3.0  → P&L = +$3,000 (hit 1:3 target)
    slot_risk=$1,000, R=-1.0  → P&L = -$1,000 (hit stop)
    slot_risk=$1,000, R=-0.5  → P&L = -$500   (partial stop)

WHAT "SLOTS" MODEL:
──────────────────
In the real strategy, up to `slots` trades can be open simultaneously.
When 10 trades are open and the market drops, all 10 stop-losses can fire
in the same session — so total intra-day exposure = slots × slot_risk.

We model this by grouping consecutive R values into batches of size=slots
(each batch = one "concurrent session"). All trades in a batch use the
SAME equity snapshot for sizing (taken at batch start), and their P&Ls
are summed before updating equity for the next batch.

Why this matters for drawdown:
    • 1 slot:  worst single day = -1.0 × slot_risk
    • 10 slots: worst day if ALL stop = -10 × slot_risk → much deeper DD

RETURNS:
────────
net_return_pct  : (final_equity - initial) / initial × 100
max_drawdown_pct: deepest peak-to-trough as % of peak equity
final_equity    : absolute ending equity $
equity_curve    : list of equity values after each batch (for plotting)

IMPORTANT — THIS IS A SEQUENTIAL REPLAY, NOT A MONTE CARLO:
──────────────────────────────────────────────────────────
We are NOT shuffling or randomising the R series.  We replay the exact
sequence that happened.  This means:
  • Results depend on the ORDER of trades (streak timing matters).
  • To stress-test, run Monte Carlo separately (shuffle r_series 1000×).
```

```
    """

    eq     = float(initial)
    peak   = float(initial)
    mdd    = 0.0
    eq_curve = [eq]

    i = 0
    while i < len(r_series):
        # --- grab a concurrent batch of up to `slots` trades ---
        batch = r_series[i : i + slots]
        # Each trade in the batch is sized on the CURRENT equity
        # (snapshot at batch start — all share the same equity base)
        slot_risk = eq * risk_pct

        # Sum of P&Ls across all concurrent trades in this batch
        batch_pnl = sum(slot_risk * r for r in batch)

        # Update equity after the batch closes
        eq += batch_pnl
        eq_curve.append(eq)

        # Track peak for drawdown calculation
        if eq > peak:
            peak = eq

        # Drawdown = (peak - current) / peak
        # This is the % you would have lost from peak if you tried to cash out now
        if peak > 0:
            dd = (peak - eq) / peak
            if dd > mdd:
                mdd = dd

        # Stop if portfolio is wiped out
        if eq <= 0:
            eq = 0.0
            break

        i += slots   # advance to next batch

    net_ret = (eq - initial) / initial * 100
    max_dd  = mdd * 100
    return net_ret, max_dd, eq, eq_curve


# ═══════════════════════════════════════════════════════════════════
# 4. BUILD ALL FOUR MATRICES
# ═══════════════════════════════════════════════════════════════════

def build_matrices(r_series: np.ndarray,
                   risk_levels=RISK_LEVELS,
```

```
                slot_options=SLOT_OPTIONS,
                initial=INITIAL_EQUITY):
"""
FOUR MATRICES EXPLAINED:
_____________________________

1. NET RETURN % MATRIX
   Value = (final_equity - initial_equity) / initial_equity × 100
   Answers: "How much did the portfolio grow over the 60-day window?"
   Green = good, Red = poor.
   Note: returns scale super-linearly (exponentially) with risk_pct because
   equity compounds — each winning trade adds to the base for the next trade.

2. MAX DRAWDOWN % MATRIX
   Value = deepest (peak_equity - trough_equity) / peak_equity × 100
   Answers: "What is the worst loss from the highest point I ever reached?"
   Lower = better.  Rule of thumb:
     <10%  → Very comfortable, barely noticeable
     10–20% → Normal, most systematic traders accept this
     20–40% → Challenging, tests discipline
     40–60% → Severe, majority of traders quit in this range
     >60%   → Portfolio destruction territory

3. RETURN ÷ DRAWDOWN RATIO (EFFICIENCY SCORE)
   Value = net_return_pct / max_drawdown_pct
   Answers: "How much return did I earn per 1% of drawdown I endured?"
   A ratio of 5.0 means: for every 1% drawdown, I earned 5% return.
   This is similar in spirit to a Calmar Ratio.
   Green = high efficiency, Red = low efficiency.
   Best configs appear top-left (low risk, few slots).

4. FINAL EQUITY $ MATRIX
   Value = ending portfolio value in dollars
   Answers: "What is my absolute ending wealth?"
   Useful for comparing raw dollar outcomes regardless of % framing.
"""
row_labels = [f"{r*100:.1f}%" for r in risk_levels]
col_labels = [f"{s} slots" for s in slot_options]

ret_m   = pd.DataFrame(index=row_labels, columns=col_labels, dtype=float)
mdd_m   = pd.DataFrame(index=row_labels, columns=col_labels, dtype=float)
ratio_m = pd.DataFrame(index=row_labels, columns=col_labels, dtype=float)
eq_m    = pd.DataFrame(index=row_labels, columns=col_labels, dtype=float)

# Store equity curves for each combination
eq_curves = {}

print("\n[SIMULATE] Building matrices...")
total = len(risk_levels) * len(slot_options)
done  = 0
```

```
    for risk in risk_levels:
        for slots in slot_options:
            net_ret, max_dd, final_eq, eq_curve = simulate_path(
                r_series, risk, slots, initial)

            rl = f"{risk*100:.1f}%"
            sl = f"{slots} slots"

            ret_m.loc[rl, sl]   = round(net_ret, 1)
            mdd_m.loc[rl, sl]   = round(max_dd, 1)
            eq_m.loc[rl, sl]    = round(final_eq, 0)
            # Efficiency = return per unit of drawdown; cap at 999 if DD ≈ 0
            ratio_m.loc[rl, sl] = round(net_ret / max_dd, 2) if max_dd > 0 else
999.0
            eq_curves[(risk, slots)] = eq_curve

            done += 1
            if done % 20 == 0 or done == total:
                print(f"  [{done:>3}/{total}] risk={risk*100:.1f}%  slots={slots}
"
                      f"→ return={net_ret:+.1f}%  mdd={max_dd:.1f}%")

    return ret_m, mdd_m, ratio_m, eq_m, eq_curves


# ═══════════════════════════════════════════════════════════════════════
# 5. DESCRIBE THE MATH — printed summary
# ═══════════════════════════════════════════════════════════════════════

def print_formula_explanation(r_series, risk_pct=0.01, slots=5, initial=100_000):
    """
    Walk through a concrete example trade-by-trade so the numbers are clear.
    """
    print("\n" + "="*72)
    print("  STEP-BY-STEP WORKED EXAMPLE")
    print(f"  Inputs: risk={risk_pct*100:.1f}%  slots={slots}
equity=${initial:,.0f}")
    print("="*72)

    eq    = float(initial)
    peak  = float(initial)
    mdd   = 0.0
    i_batch = 0
    i_trade = 0

    # Show first 5 batches
    idx = 0
    print(f"  {'Batch':>5}  {'Equity Start':>14}  {'slot_risk':>10}  "
          f"{'R values':>20}  {'Batch P&L':>11}  {'Equity End':>13}  {'MDD':>6}")
    print(f"  {'-'*5}  {'-'*14}  {'-'*10}  {'-'*20}  {'-'*11}  {'-'*13}  {'-'*6}")
```

```
    while idx < len(r_series) and i_batch < 5:
        batch     = r_series[idx : idx + slots]
        slot_risk = eq * risk_pct
        batch_pnl = sum(slot_risk * r for r in batch)
        eq_after  = eq + batch_pnl

        if eq_after > peak:
            peak = eq_after
        dd = (peak - eq_after) / peak * 100 if peak > 0 else 0
        if dd > mdd:
            mdd = dd

        r_str = "  ".join([f"{r:+.3f}" for r in batch[:3]])
        if len(batch) > 3:
            r_str += f" …+{len(batch)-3}"

        print(f"  {i_batch+1:>5}  ${eq:>13,.2f}  ${slot_risk:>9,.2f}  "
              f"{r_str:>20}  ${batch_pnl:>+10,.2f}  ${eq_after:>12,.2f}
{dd:>5.1f}%")

        eq   = max(eq_after, 0)
        idx += slots
        i_batch += 1

    print(f"\n  FORMULA RECAP:")
    print(f"    slot_risk    = current_equity × risk_pct")
    print(f"    trade_pnl    = slot_risk × R_i")
    print(f"    batch_pnl    = Σ(trade_pnl for all slots in batch)")
    print(f"    new_equity   = old_equity + batch_pnl")
    print(f"    drawdown     = (peak_equity - current_equity) / peak_equity × 100")
    print("="*72)


# ═══════════════════════════════════════════════════════════════════
# 6. EXCEL EXPORT — fully styled workbook
# ═══════════════════════════════════════════════════════════════════

def export_excel(df_trades, r_col, ret_m, mdd_m, ratio_m, eq_m,
                 eq_curves, r_series, initial=INITIAL_EQUITY):
    """
    WORKBOOK STRUCTURE:
    ─────────────────────
    Sheet 1 — Guide           : Explains every formula, term, and matrix
    Sheet 2 — Net Return %    : Green-red heatmap of portfolio returns
    Sheet 3 — Max Drawdown %  : Red-green heatmap (lower = better)
    Sheet 4 — Return÷MDD      : Efficiency score heatmap
    Sheet 5 — Final Equity $  : Absolute ending dollar value
    Sheet 6 — R-Series Detail : Every trade's R, cumulative R, running equity
    Sheet 7 — Equity Curves   : Selected equity paths for 4 risk configs
    """
    os.makedirs(os.path.dirname(OUTPUT_PATH), exist_ok=True)
```

```
    wb = openpyxl.Workbook()

    # ── Style constants ──────────────────────────────────────────────
    HDR_FILL = PatternFill('solid', fgColor='1F4E79')
    HDR_FONT = Font(bold=True, color='FFFFFF', size=11, name='Calibri')
    BAN_FILL = PatternFill('solid', fgColor='0D2137')
    BAN_FONT = Font(bold=True, color='FFFFFF', size=13, name='Calibri')
    SEC_FILL = PatternFill('solid', fgColor='2F5496')
    SEC_FONT = Font(bold=True, color='FFFFFF', size=11, name='Calibri')
    LBL_FILL = PatternFill('solid', fgColor='EEF2FF')
    LBL_FONT = Font(bold=True, size=10, name='Calibri', color='1F4E79')
    WIN_FILL = PatternFill('solid', fgColor='E2EFDA')
    LOS_FILL = PatternFill('solid', fgColor='FADADD')
    NEU_FILL = PatternFill('solid', fgColor='F5F5F5')
    BOD_FONT = Font(size=10, name='Calibri')
    CTR = Alignment(horizontal='center', vertical='center', wrap_text=True)
    LFT = Alignment(horizontal='left', vertical='center', indent=1)
    RT  = Alignment(horizontal='right', vertical='center')
    BRD = Border(left=Side(style='thin', color='C0C0C0'),
                 right=Side(style='thin', color='C0C0C0'),
                 top=Side(style='thin', color='C0C0C0'),
                 bottom=Side(style='thin', color='C0C0C0'))
    THK = Border(left=Side(style='medium', color='1F4E79'),
                 right=Side(style='medium', color='1F4E79'),
                 top=Side(style='medium', color='1F4E79'),
                 bottom=Side(style='medium', color='1F4E79'))

    # ── Helpers ──────────────────────────────────────────────────────
    def banner(ws, text, r, c1, c2, h=28):
        ws.merge_cells(start_row=r, start_column=c1, end_row=r, end_column=c2)
        c = ws.cell(r, c1, text)
        c.fill = BAN_FILL; c.font = BAN_FONT; c.alignment = CTR
        ws.row_dimensions[r].height = h

    def section(ws, text, r, c1, c2):
        ws.merge_cells(start_row=r, start_column=c1, end_row=r, end_column=c2)
        c = ws.cell(r, c1, text)
        c.fill = SEC_FILL; c.font = SEC_FONT; c.alignment = CTR; c.border = THK
        ws.row_dimensions[r].height = 20

    def hdr_row(ws, labels, r, c_start=1):
        for j, lbl in enumerate(labels, c_start):
            c = ws.cell(r, j, lbl)
            c.fill = HDR_FILL; c.font = HDR_FONT; c.alignment = CTR; c.border = BRD
        ws.row_dimensions[r].height = 18

    def write_matrix(wb, sheet_name, matrix, title, fmt, note,
                     banner_text, good_is_high, risk_levels, slot_options):
        """
        Writes one heatmap matrix sheet.
```

```
        HOW THE COLOR SCALE WORKS:
          ColorScaleRule applies a three-point gradient across all cells:
            • good_is_high=True  (return/ratio): low=red, mid=yellow, high=green
            • good_is_high=False (drawdown):     low=green, mid=yellow, high=red
          This lets you visually scan for the best/worst cells instantly.
        """
        ws = wb.create_sheet(sheet_name)
        ws.sheet_view.showGridLines = False
        row_labels = [f"{r*100:.1f}%" for r in risk_levels]
        col_labels = [str(s) for s in slot_options]
        nc = len(slot_options) + 2   # risk col + data cols + mean col

        # Banner and note
        banner(ws, banner_text, 1, 1, nc)
        ws.merge_cells(start_row=2, start_column=1, end_row=2, end_column=nc)
        c = ws.cell(2, 1, note)
        c.font = Font(italic=True, size=9, color='7A7974', name='Calibri')
        c.alignment = LFT

        # Sub-header: arrow label for slots
        ws.cell(3, 1, 'Risk % ↓ / Slots →').fill  = HDR_FILL
        ws.cell(3, 1).font      = HDR_FONT
        ws.cell(3, 1).alignment = CTR
        ws.cell(3, 1).border    = BRD
        for j, sl in enumerate(col_labels, 2):
            c = ws.cell(3, j, sl)
            c.fill = HDR_FILL; c.font = HDR_FONT; c.alignment = CTR; c.border = BRD
        c = ws.cell(3, nc, 'Row Avg')
        c.fill = HDR_FILL; c.font = HDR_FONT; c.alignment = CTR; c.border = BRD
        ws.row_dimensions[3].height = 18

        # Data
        data_start = 4
        for ri, rl in enumerate(row_labels, data_start):
            c = ws.cell(ri, 1, rl)
            c.fill = LBL_FILL; c.font = LBL_FONT; c.alignment = CTR; c.border = BRD
            row_vals = []
            for ci, sl in enumerate([f"{s} slots" for s in slot_options], 2):
                val = float(matrix.loc[rl, sl])
                row_vals.append(val)
                cell = ws.cell(ri, ci, round(val, 2))
                cell.number_format = fmt
                cell.alignment = CTR; cell.border = BRD; cell.font = BOD_FONT
            # Row average in last column
            avg = np.mean(row_vals)
            mc = ws.cell(ri, nc, round(avg, 2))
            mc.number_format = fmt; mc.alignment = CTR; mc.border = BRD
            mc.font = Font(bold=True, size=10, name='Calibri', color='2F5496')
            ws.row_dimensions[ri].height = 17

        # Color scale conditional formatting
```

```
        data_range = (f"B{data_start}:"
                      f"{get_column_letter(len(slot_options)+1)}"
                      f"{data_start + len(risk_levels) - 1}")
        if good_is_high:
            rule = ColorScaleRule(
                start_type='min',  start_color='F8696B',
                mid_type='percentile', mid_value=50, mid_color='FFEB84',
                end_type='max',    end_color='63BE7B')
        else:
            rule = ColorScaleRule(
                start_type='min',  start_color='63BE7B',
                mid_type='percentile', mid_value=50, mid_color='FFEB84',
                end_type='max',    end_color='F8696B')
        ws.conditional_formatting.add(data_range, rule)

        # Column widths
        ws.column_dimensions['A'].width = 13
        for j in range(2, nc + 1):
            ws.column_dimensions[get_column_letter(j)].width = 11
        ws.freeze_panes = 'B4'
        return ws

    # ═══════════════════════════════════════════════════════════════
    # SHEET 1: GUIDE
    # ═══════════════════════════════════════════════════════════════
    ws_guide = wb.active
    ws_guide.title = 'Guide'
    ws_guide.sheet_view.showGridLines = False
    banner(ws_guide,
           f'Sensitivity Matrix Guide — {len(r_series):,} Trades | '
           f'${initial:,.0f} Start | R-Multiple Replay Engine', 1, 1, 5)

    ws_guide.column_dimensions['A'].width = 3
    ws_guide.column_dimensions['B'].width = 30
    ws_guide.column_dimensions['C'].width = 65
    ws_guide.column_dimensions['D'].width = 22
    ws_guide.column_dimensions['E'].width = 22

    guide_content = [
        ('CORE CONCEPT', None),
        ('What is an R-Multiple?',
         'R = Trade P&L ÷ Initial Risk $. Dimensionless — independent of account
size.'),
        ('Why replay R-multiples?',
         'Because R encodes the outcome ratio, we can scale any sequence to any
account '
         'size or risk% and the relative result is correct.'),
        ('Van Tharp CPR sizing',
         'slot_risk = equity × risk_pct.  Shares = slot_risk ÷ stop_distance. '
         'Equity compounds after each trade.'),
```

```
        ('THE FORMULA', None),
        ('slot_risk ($)',
         'current_equity × risk_pct  →  e.g. $100,000 × 1% = $1,000'),
        ('trade_pnl ($)',
         'slot_risk × R  →  R=+3.0: +$3,000 win | R=-1.0: -$1,000 full stop'),
        ('batch_pnl ($)',
         'Σ(slot_risk × R) for all slots in concurrent batch  →  '
         'models simultaneous open positions'),
        ('new_equity ($)',
         'old_equity + batch_pnl  →  compounding: each win grows the base'),
        ('max_drawdown (%)',
         '(peak_equity - trough_equity) / peak_equity × 100  →  '
         'worst % loss from the highest point reached'),

        ('WHAT EACH MATRIX SHOWS', None),
        ('Net Return %',
         '(final_equity - initial) / initial × 100.  '
         'Green = good. Scales super-linearly with risk% due to compounding.'),
        ('Max Drawdown %',
         'Deepest peak-to-trough decline. Lower = better. '
         '>40% is very hard to recover from psychologically.'),
        ('Return ÷ MDD',
         'Efficiency score = how much return per 1% of drawdown. '
         'Similar to Calmar Ratio. Highest values = best risk-adjusted setup.'),
        ('Final Equity $',
         'Absolute ending portfolio value in dollars. '
         'Useful for comparing raw wealth outcomes.'),

        ('SLOTS — EXPLAINED', None),
        ('Does slot count change trade count?',
         'NO — all signals still fire. Slots only cap how many are open at once.'),
        ('What slots do change',
         'Concurrent exposure. 10 slots open = 10 trades can hit stops in same
session '
         '→ deeper intraday drawdown. Batching groups consecutive trades into '
         'concurrent windows of size=slots.'),
        ('Rule of thumb',
         '1–3 slots: lower drawdown, lower return. '
         '5–10 slots: balanced. >15 slots: high volatility in equity curve.'),

        ('SWEET SPOT GUIDANCE', None),
        ('Conservative',
         '0.5%–0.75% risk, 3–5 slots → ~60–100% return, <15% max drawdown'),
        ('Balanced',
         '1% risk, 5–7 slots → ~140–145% return, ~24% max drawdown, ~6×
efficiency'),
        ('Aggressive',
         '2% risk, 5–10 slots → ~370–400% return, ~43% max drawdown'),
        ('Danger zone',
         '≥3% risk, ≥10 slots → drawdown >60%, portfolio destruction risk'),
    ]
```

```
    rn = 3
    for label, desc in guide_content:
        if desc is None:
            section(ws_guide, label, rn, 2, 5)
        else:
            c1 = ws_guide.cell(rn, 2, label)
            c1.fill = LBL_FILL; c1.font = LBL_FONT; c1.border = BRD; c1.alignment =
LFT
            ws_guide.merge_cells(start_row=rn, start_column=3, end_row=rn,
end_column=5)
            c2 = ws_guide.cell(rn, 3, desc)
            c2.font = BOD_FONT; c2.border = BRD; c2.alignment = LFT
        ws_guide.row_dimensions[rn].height = 22
        rn += 1

    # ═══════════════════════════════════════════════════════════════
    # SHEETS 2–5: THE FOUR MATRICES
    # ═══════════════════════════════════════════════════════════════
    write_matrix(wb, 'Net Return %', ret_m,
        'Net Return %', '0.0"%"',
        'Each cell = total % portfolio growth over the trade window. Green = high
return.',
        f'Net Return % Matrix | {len(r_series):,} Trades | Green = Higher Return',
        good_is_high=True, risk_levels=RISK_LEVELS, slot_options=SLOT_OPTIONS)

    write_matrix(wb, 'Max Drawdown %', mdd_m,
        'Max Drawdown %', '0.0"%"',
        'Each cell = deepest equity decline from peak. Lower = better. Red =
dangerous.',
        f'Max Drawdown % Matrix | {len(r_series):,} Trades | Green = Lower
Drawdown',
        good_is_high=False, risk_levels=RISK_LEVELS, slot_options=SLOT_OPTIONS)

    write_matrix(wb, 'Return÷MDD Ratio', ratio_m,
        'Return ÷ MDD', '0.00"x"',
        'Each cell = Net Return % ÷ Max Drawdown %. Higher = more efficient use of
risk.',
        f'Return÷MDD Efficiency Score | Green = Best Risk-Adjusted Configs',
        good_is_high=True, risk_levels=RISK_LEVELS, slot_options=SLOT_OPTIONS)

    write_matrix(wb, 'Final Equity $', eq_m,
        'Final Equity $', '$#,##0',
        'Each cell = absolute ending portfolio value in $. Green = largest final
wealth.',
        f'Final Equity $ Matrix | Start: ${initial:,.0f}',
        good_is_high=True, risk_levels=RISK_LEVELS, slot_options=SLOT_OPTIONS)

    # ═══════════════════════════════════════════════════════════════
    # SHEET 6: R-SERIES DETAIL
    # ═══════════════════════════════════════════════════════════════
```

```
    ws_r = wb.create_sheet('R-Series Detail')
    ws_r.sheet_view.showGridLines = False
    banner(ws_r,
           f'R-Multiple Series — {len(r_series):,} Trades | '
           'Shows how each R feeds into running equity (1% risk baseline)', 1, 1,
7)

    hdr_row(ws_r,
        ['Trade #', 'R-Multiple', 'Win/Loss',
         'Cumulative R', 'Running Equity $', 'DD from Peak %', 'Notes'], 2)

    eq    = float(initial)
    peak  = float(initial)
    cum_r = 0.0
    for i, r_val in enumerate(r_series, 1):
        cum_r += r_val
        slot_risk = eq * 0.01        # use 1% baseline for this display
        eq = max(0.0, eq + slot_risk * r_val)
        if eq > peak: peak = eq
        dd = (peak - eq) / peak * 100 if peak > 0 else 0
        wl   = 'WIN' if r_val > 0 else 'LOSS'
        fill = WIN_FILL if r_val > 0 else LOS_FILL

        note = ''
        if r_val == r_series.min(): note = '◄ WORST TRADE'
        if r_val == r_series.max(): note = '◄ BEST TRADE'
        if dd >= 15: note = f'◄ DD {dd:.1f}%'

        vals = [i, round(r_val,4), wl, round(cum_r,4), round(eq,2), round(dd,2),
note]
        for j, v in enumerate(vals, 1):
            c = ws_r.cell(i+2, j, v)
            c.fill = fill; c.alignment = CTR; c.border = BRD
            c.font = BOD_FONT
            if j == 5: c.number_format = '$#,##0.00'
            if j == 6: c.number_format = '0.00"%"'
        ws_r.row_dimensions[i+2].height = 15

    # DataBar on R-Multiple column (visual magnitude)
    ws_r.conditional_formatting.add(f'B3:B{len(r_series)+2}',
        DataBarRule(start_type='min', end_type='max', color='4472C4'))
    for j, w in enumerate([9, 14, 10, 14, 18, 16, 18], 1):
        ws_r.column_dimensions[get_column_letter(j)].width = w
    ws_r.freeze_panes = 'A3'

    # ═══════════════════════════════════════════════════════════════
    # SHEET 7: EQUITY CURVES (table version)
    # ═══════════════════════════════════════════════════════════════
    ws_eq = wb.create_sheet('Equity Curves')
    ws_eq.sheet_view.showGridLines = False
    banner(ws_eq,
```

```
            'Equity Curves — 4 Risk Configs × 5 Slots | '
            'Shows how portfolio grows batch by batch', 1, 1, 5)

    configs = [(0.005, 5), (0.01, 5), (0.02, 5), (0.03, 5)]
    col_hdrs = ['Batch #'] + [f"{r*100:.1f}% risk / {s} slots"
                              for r, s in configs]
    hdr_row(ws_eq, col_hdrs, 2)

    max_len = max(len(eq_curves[(r, s)]) for r, s in configs)
    for bi in range(max_len):
        ri = bi + 3
        ws_eq.cell(ri, 1, bi).alignment = CTR
        ws_eq.cell(ri, 1).border = BRD
        ws_eq.cell(ri, 1).font = BOD_FONT
        for j, (r, s) in enumerate(configs, 2):
            curve = eq_curves[(r, s)]
            val = curve[bi] if bi < len(curve) else ''
            c = ws_eq.cell(ri, j, round(val, 2) if val != '' else '')
            c.alignment = CTR; c.border = BRD; c.font = BOD_FONT
            if val != '': c.number_format = '$#,##0'
        ws_eq.row_dimensions[ri].height = 15

    for j, w in enumerate([10, 22, 22, 22, 22], 1):
        ws_eq.column_dimensions[get_column_letter(j)].width = w
    ws_eq.freeze_panes = 'A3'

    # ── Save ──────────────────────────────────────────────────────────
    wb.save(OUTPUT_PATH)
    print(f"\n  ✓ Workbook saved → {OUTPUT_PATH}")


# ═══════════════════════════════════════════════════════════════════════
# 7. MAIN ENTRY POINT
# ═══════════════════════════════════════════════════════════════════════

def main():
    print("="*72)
    print("  SENSITIVITY MATRIX GENERATOR")
    print(f"  Journal   : {JOURNAL_PATH}")
    print(f"  Capital   : ${INITIAL_EQUITY:,.0f}")
    print(f"  Risk levels tested : {[f'{r*100:.1f}%' for r in RISK_LEVELS]}")
    print(f"  Slot options tested: {SLOT_OPTIONS}")
    print("="*72)

    # Load journal
    df_trades, r_col = load_journal(JOURNAL_PATH)
    r_series = df_trades[r_col].values

    # Worked example (educational printout)
    print_formula_explanation(r_series, risk_pct=0.01, slots=5)
```

```
    # Build matrices
    ret_m, mdd_m, ratio_m, eq_m, eq_curves = build_matrices(
        r_series, RISK_LEVELS, SLOT_OPTIONS, INITIAL_EQUITY)

    # Export
    print("\n[EXPORT] Writing Excel workbook...")
    export_excel(df_trades, r_col, ret_m, mdd_m, ratio_m, eq_m,
                 eq_curves, r_series, INITIAL_EQUITY)

    # Print key findings
    print("\n[RESULTS] Top configurations by Return÷MDD efficiency:")
    rows = []
    for risk in RISK_LEVELS:
        for slots in SLOT_OPTIONS:
            rl = f"{risk*100:.1f}%"
            sl = f"{slots} slots"
            rows.append({
                'Risk%': rl, 'Slots': slots,
                'Return%': ret_m.loc[rl, sl],
                'MaxDD%': mdd_m.loc[rl, sl],
                'Efficiency': ratio_m.loc[rl, sl],
            })
    top = sorted(rows, key=lambda x: x['Efficiency'], reverse=True)[:10]
    print(f"  {'Risk%':>7}  {'Slots':>6}  {'Return%':>9}  {'MaxDD%':>7}
{'Eff':>7}")
    print(f"  {'-'*7}  {'-'*6}  {'-'*9}  {'-'*7}  {'-'*7}")
    for r in top:
        print(f"  {r['Risk%']:>7}  {r['Slots']:>6}  "
              f"{r['Return%']:>8.1f}%  {r['MaxDD%']:>6.1f}%
{r['Efficiency']:>7.2f}x")

    print(f"\n✅ DONE. Open {OUTPUT_PATH}")
    print("="*72)


if __name__ == '__main__':
    main()
```