A blue text on a black background

Description automatically generated

**Imperial College Business School**

**MSc Financial Engineering and Risk Management**

**Quantitative Analysis**

**Forecasting Equity Index Drift with Macro and Market Signals: A Risk-Aware Monthly Allocation Framework for SPY and QQQ**

**Applied Project  
2024-2025**

**Abstract**

This research develops a machine learning framework to predict monthly returns of SPY and QQQ indices through ensemble models combined with dynamic position sizing. The results are quite interesting, despite achieving weak predictive correlations, the strategy still produced respectable Sharpe ratios of 0.579 and 0.404. This finding suggests that how positions are sized can be more important than getting high accuracy prediction. The research uses 376 monthly observations from April 1994 to July 2025. Starting with 91 features extracted from market data, economic surprises, and technical indicators, it reduced to 35 features using correlation-based pruning to avoid overfitting. The ensemble approach weights Ridge regression at 8%, ElasticNet at 7%, Random Forest at 25%, XGBoost at 35%, and LightGBM at 25%. Then, it applied Kalman filtering to smooth out the noisy predictions. The key contribution is demonstrating that position sizing through tanh activation function matters more than prediction accuracy for generating returns. Monte Carlo simulations show only a 13.1% probability of loss over 12 months, supporting the effectiveness of this risk management approach. These findings challenge the traditional view that profitable trading requires highly accurate predictions.

Table of Contents

Abstract 2

1. Introduction and Literature Review 4

2. Methodology 4

**2.1 Data Collection and Feature Engineering** 4

**2.2 Feature Selection and Dimensionality Reduction** 6

**2.3 Train-Test Split and Data Cleaning** 7

**2.4 Ensemble Model Architecture** 8

**2.5 Gaussian Mixture Model for Regime Detection** 9

**2.6 Meta-Model for Blending** 9

**2.7 Kalman Filtering and Position Sizing Innovation** 9

3. Empirical Results and Analysis 10

**3.1 Predictive Performance Metrics** 10

**3.2 Portfolio Performance** 11

**3.3 Statistical Significance Testing** 12

**3.4 Cross-Validation and Robustness Testing** 12

**3.5 Monte Carlo Risk Assessment** 13

**3.6 Position Sizing Attribution** 15

4. Conclusion 16

5. References 17

6. Appendices 18

7. Source Code 22

**1. Introduction and Literature Review**

In general, stock market is considered as one of the most challenging areas for forecasting due to its noisy nature and constantly evolving patterns. According to efficient market hypothesis, it says that prices are already incorporated all available information, which only leave a small window for predictability. Welch and Goyal (2008) discovered that traditional predictive models fail out of sample, with poor performance over 30 years and if an investor use such models would not outperform a simple buy-and-hold strategy. This was further confirmed by Goyal, Welch, and Zafirov (2024), they found that more than one-third of 29 newly proposed variables lost their empirical significance when tested with extended data.

However, Campbell and Thompson (2008) demonstrated that low monthly R-squared values can still transform into meaningful economic gains. They showed that with suitable economic constraints, some predictors achieved monthly R-squared values between 0.19% and 0.57%. This suggests that statistical significance and economic significance are completely different things.

Importantly, the thing that makes this research paper unique is it focuses on **position sizing** rather than prediction accuracy. Since López de Prado (2018), argues that position sizing might be more important than the signals themselves: "Your machine learning algorithm can achieve high accuracy, but if you don’t size your bets properly, your investment strategy will inevitably lose money." This insight led to the central research question: Can weak predictive signals actually generate profit through appropriate sizing?

Also, Gu et al. (2020) found out that using machine learning models achieve only 0.16%-0.40% R-squared for monthly returns across 30,000 stocks (1957–2016), but with small improvements result in Sharpe ratios of 0.77 compared to 0.51 for buy-and-hold strategies. So, it should be expected that the during application, it could show weak predictive signal, but it can still be valuable when combined with proper position sizing.

Additionally, Harvey et al. (2016) and Bailey and López de Prado (2014) emphasize that even low R-squared values can offer economic value if adjusted to prevent overfitting. Harvey et al. (2018) also showed that with volatility scaling, a form of dynamic position sizing, it could improve risk-adjusted returns, emphazing the importance of position sizing over prediction accuracy. Based on all these insights, this research’s goal is to confirm how strategic position sizing can transform weak signals into profitable strategies, achieving respectable Sharpe ratios with different target.

**2. Methodology**

**2.1 Data Collection and Feature Engineering**

The dataset has 376 monthly observations from April 1994 to July 2025, covering multiple major market events like financial crisis, COVID, inflation, etc. The target prediction is logarithmic return since normal prices exhibit non-stationarity:

To avoid look-ahead bias, each period's target represents the next period's return:

Respectively, SPY and QQQ were selected to represent broad market and technology sector exposure. SPY provides stability. While QQQ offers higher volatility and potentially strong signal, allowing the methodology to be generalized across different market segments.

**Market Microstructure Features**

Momentum exhibits price movements across 1-, 3-, 6- and 12-month periods, with all indicators using t-1 data to avoid leakage with as the price and k as the different periods:

Momentum acceleration measures trend strength:

Realized volatility uses standard deviation of log returns across 3, 6, and 12 month periods. With additional features include volatility percentage changes, skewness, and kurtosis to capture tail risks:

Technical indicators like RSI and MACD provide information about market behavior and momentum across different market conditions, which can be very useful to see the reflection of calm and volatile periods.

**Macroeconomic Surprise Features**

Macroeconomic surprise features compare actual data releases with Bloomberg survey medians for indicators like CPI, PMI, and unemployment:

with = historical surprise volatility.

Rolling 12-month z-scores account for changing market sensitivity:

**Cross Market Relationships**

Term spread (10-year minus 2-year Treasury yields) indicates growth expectations and recession probability:

VIX change and z-scores capture risk sentiment shifts:

Rolling SPY-QQQ correlations differentiate between systematic and sector-specific movements:

High correlation suggests systematic factors dominate, while low correlation indicates sector dominate.

**2.2 Feature Selection and Dimensionality Reduction**

Initially, there were 91 features in the strategy. Therefore, to prevent multicollinearity and overfitting, it needed a systematic reduction. There are three stages that are intended to eliminate the redundant features but kept the informative ones.

**Stage 1: Removing Useless Features**

The first stage removes features with excessive missing data, specifically those with over 40% missing values, and eliminates features with variance below 2nd percentile and removes any features with standard deviation less than 1e-10.

**Stage 2: Pre-selection**

This is done before the main pruning, if the remaining set still contains more than 2 x K (K =35), an early scoring filter would be applied to reduce features:

which high Variance means better signal, while Diversity accounts for average correlation between features. Features with higher scores are kept, ensuring the remains are both informative and diverse.

**Stage 3: Correlation-Based Pruning**

In the final stage, the strict\_prune function is implemented as the key process to reduce the features. It iteratively removes highly correlated features using this process:

1. Calculate correlation matrix
2. Find the highest correlation
3. If correlation above 0.95, drop the feature with lower variance
4. Stop when no correlations exceed 0.95 or hit 100 iterations

The idea is that if two features are 95% correlated, it only keep the one with higher variance because it has more information. However, if it still has more than 35 features after removing correlations, it does a final scoring:

which is the same as before but adding uniqueness helps it keep only the feature with the most diverse data in its dataset. This ensures the model would get the best possible set of features. The heatmaps are shown in Appendix 1.

**2.3 Train-Test Split and Data Cleaning**

For train and test split, it used 75% for training (about 282 months) and 25% for testing (94 months). Before training, both the training and testing sets are checked for missing values in any feature or in the target. Hence, any rows containing NaN were removed to ensure the model receives only complete data.

|  |
| --- |
| **Train – Test Split**  Months  Test (94 months)  Train (282 months) |

**Figure 1: Train/Test Split**

Next, to make sure the features and targets are all correctly aligned, the create\_aligned\_targets function is responsible for shifting the return column by -1, so the models predict the next monthly return.

**2.4 Ensemble Model Architecture**

The ensemble combines five models to balance stability and detect difficult patterns. Linear models capture trends but miss nonlinear relationships, while tree models detect nonlinearity but easily overfit. The ensemble combines each model's strengths to offset its weaknesses.

**Linear Models (15%):**

**Ridge Regression** (8%) uses L2 penalty with penalty=0.5 to handle correlated predictors and reduce overfitting.

**ElasticNet** (7%) combines L1 and L2 penalties (alpha=0.3, l1\_ratio=0.7) to act as feature selection and stability.

Both models use RobustScaler preprocessing, which uses median and interquartile range to make the scalling less sensitive which to handle outliers.

**Tree-Based Models (85%)**

**Random Forest** (25%) uses 120 trees with max\_depth=7, min\_samples\_split=8, min\_samples\_leaf=3, and max\_features="sqrt" to prevent overfitting through bootstrap sampling where each tree is trained and shuffle separately which adds randomness and reducing correlation.

**XGBoost** (35%), the highest-weighted model, employs gradient boosting which builds each new trees to learns mistakes from the previous one. XGBoost uses 250 trees, learning\_rate=0.04, subsample=0.75, colsample\_bytree=0.75, reg\_alpha=0.05, reg\_lambda=0.3, and gamma=0.005. This configuration ensures slow, steady learning rate which suitable for noisy data like monthly log return.

**LightGBM** (25%) is also boosting method like XGBoost and also uses 250 trees with learning\_rate=0.04, max\_depth=6, and num\_leaves=45. It grows tree leaf-wise and histogram-based splitting which enable faster training while still capturing complex patterns.

The final model is a weighted average of the five model predictions:

A signal enhancement step amplifies predictions when XGBoost and LightGBM agree, this gives the forecast a small boost. It takes the average prediction, compare it to its usual level:

where is the average of tree model predictions. This 0.12 boost strengthens the signals without adding excessive noise.

In short, the ensemble uses linear models for stability and tree models for pattern discovery. The weights reflect the practical role of each model: XGBoost and LightGBM do most of the heavy lifting, Random Forest adds non-linear structure, and the two linear models keep the whole system grounded.

**2.5 Gaussian Mixture Model for Regime Detection**

The Gaussian Mixture Model (GMM) divides the market into three regimes: bullish, bearish, and choppy, using recent returns, 20 day volatility, and 20 day average return. It assumes data comes from a mixed of three normal distributions and calculates regime probabilities of as:

where is weight of regime and is the Gaussian likelihood. This gives a soft assignment to regimes. Later, it uses these probabilities to size trades more aggressively when the regime is clear and reduce exposure when it is uncertain.

**2.6 Meta-Model for Blending**

Pure model predictions can be extreme, so model predictions are combined with the historical average return to prevent extreme forecasts. A blending weight is chosen by time-series cross-validation from a range of values 0.8 to 1.0. The historical average is computed only on the training part of each fold to avoid leakage. The blended forecast is calculated as:

In most cases, falls around 0.85 and 0.95, giving the model most of the weight while a small historical component (5–15%) helps stabilize predictions during unusual market regime. This stops the model from making crazy predictions during unstable regime.

**2.7 Kalman Filtering and Position Sizing Innovation**

The ensemble model is usually subject to high monthly volatility, which causes frequent position changes and higher trading costs. Hence, the Kalman filter smooths the ensemble's volatile predictions but still preserving signals and it treats outputs as noisy observations of true underlying trends. Process noise at 0.02 allows gradual signal changes while measurement noise (0.05) measures prediction uncertainty. These configurations help reducing volatility in the predictions. Now, the output is smoother and less noisy.

Pre-smoothing adjustments enhance signal clarity by reducing predictions below 40th percentile by 0.008, those above 60th percentile increased by 0.008, and middle values halved. This creates more decisive signals.

Post-smoothing, predictions convert to portfolio weights through z-score normalization using training standard deviation. Extreme predictions like below 30th or above 70th percentile are multiplied by 1.3 to emphasize strong signals.

These z-scores are then transformed into portfolio weights using the formula:

Finally, tanh activation function smoothly turns signals into weights, capping positions at 40% while ensuring gradually stable transitions. GMM regime uncertainty scales weights down proportionally. Strong signals maintain minimum 8% positions to ensure meaningful trades. In summary, these steps are the central to the entire research mechanism that connects the prediction strength and market confidence to position sizing, while embedding risk control.

## **3. Empirical Results and Analysis**

### **3.1 Predictive Performance Metrics**

A table with numbers and text

Description automatically generated

**Table 1: Predictive Performance**

Table 1 shows the model’s poor performance. From April 2020 to June 2025, correlation between predicted and actual monthly returns were 0.086 for SPY and 0.028 for QQQ, with negative R² values (–0.051 and –0.048) indicates forecasts underperformed a simple historical mean.

In addition, directional accuracy was modest at 50.8% for SPY and 57.1% for QQQ. ROC-AUC scores (SPY = 0.455 vs QQQ = 0.578) suggest stronger predictive structure in the technology sector, while information coefficients (0.115 and 0.076) show minimal positive rank correlation. Mean squared errors of 0.0022 (SPY) and 0.0036 (QQQ), alongside mean absolute errors of 0.0399 and 0.0475 confirm weak predictive power. However, it needs further configuration to see if such weak forecasts can still turn profit when combined with a robust position sizing and risk-management framework.

**3.2 Portfolio Performance**

A table of performance metrics

Description automatically generated

**Table 2: Portfolio Performance Metrics**

In Table 2, SPY showed Sharpe ratios of 0.579 and 0.519 after transaction costs, while QQQ achieved 0.404 and 0.368, suggesting that position sizing could transform weak signals into viable strategies.

Also, in the Appendix 2, Figures 1 and 2 shows that, the drawdowns remained contained at -3.6% for SPY and -11.8% for QQQ with mostly positive rolling 12 month Sharpe ratios across periods, but cumulative returns still underperformed buy-and-hold. Since the strategies avoided large losses during downturns but missed significant upside, resulting in flatter growth curves.

Monthly returns showed small frequent losses offset by occasional larger gains. Average position sizes of 20% for both indices demonstrate the tanh-based sizing's effectiveness in balancing exposure and risk. Overall, position sizing enhanced stability at the expense of absolute returns but it confirms that position sizing worked.

### **3.3 Statistical Significance Testing**

|  |  |  |
| --- | --- | --- |
| Metric | SPY | QQQ |
| T-Statistic | -1.8499 | -2.0418 |
| P-value | 0.0643 | 0.0412 |
| 95% CI Lower | -25.27% | -30.79% |
| 95% CI Upper | 1.28% | -0.25% |
| DM Test Value | 0.0665 | 0.0428 |

**Table 3:Statistical Significance Tests**

Table X shows QQQ's results are statistically significant (t-statistic: −2.04, p-value: 0.041) with 95% confidence interval from −30.79% to −0.25%, confirming consistent underperformance. SPY's results (p-value: 0.064, CI: −25.27% to 1.28%) fell short of significance. Diebold-Mariano tests confirmed these findings. **Despite underperforming buy-and-hold, both strategies achieved its primary goal, by generating positive absolute returns with minimal drawdowns from nearly random predictions.**

**3.4 Cross-Validation and Robustness Testing**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Fold** | **Test Period** | **SPY Sharpe** | **SPY Return** | **QQQ Sharpe** | **QQQ Return** |
| 1 | 2015-02 to 2017-02 | 1.898 | 7.0% | 0.887 | 6.1% |
| 2 | 2017-03 to 2019-03 | -0.269 | -1.5% | 0.241 | 1.6% |
| 3 | 2019-04 to 2021-04 | 0.896 | 6.5% | 0.389 | 4.2% |
| 4 | 2021-05 to 2023-05 | 0.337 | 2.4% | -0.064 | -1.2% |
| 5 | 2023-06 to 2025-06 | 0.320 | 1.6% | 0.445 | 3.2% |
| **Mean** |  | **0.636** | **3.2%** | **0.380** | **2.8%** |

**Table 4: Cross-Validation Results**

Using 5-fold expanding-window validation, SPY achieved the mean Sharpe ratio of 0.636 (σ=0.817), ranging from 1.898 to −0.269. QQQ showed more stability with mean Sharpe of 0.380 (σ=0.346), ranging from 0.887 to −0.064. Both strategies maintained 80% success rate and hit rates above random (SPY: 60.8%, QQQ: 59.2%). Despite the average of ROC-AUC over the periods was at 0.51 for SPY and 0.53 for QQQ, results show stable risk-controlled returns, with only one losing period each, SPY during the late bull market (2017-2019) and QQQ during Fed tightening (2021). These results suggest the position sizing mechanism adapts reasonably well to different market regimes, though performance naturally varies with market conditions. More plots are shown in the Appendix 3, Figure 1 and 2.

### **3.5 Monte Carlo Risk Assessment**

A comparison of graphs and diagrams

Description automatically generated with medium confidence

**Figure 2: Monte Carlo Simulation**

The simulations use 1,000 paths with Student-t distributions (df=5) and 2% jump probability, parameters estimated at the latest date at which μ = 6-month average of Kalman-smoothed drift, σ = VIX-adjusted recent volatility. In figure 3, Monte Carlo simulated with median outcome of $113 and 13.1% probability of loss, revealing positive expected returns but significant tail risk.

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Expected Return | 15.9% |
| Volatility | 14.4% |
| Sharpe Ratio | 1.10 |
| 95% Value-at-Risk | 6.0% |
| 95% Expected Shortfall | 10.7% |
| Probability of Loss | 13.1% |
| Median Outcome | ~$113 |
| 95th Percentile | ~$140 |

**Table 5:Forward-Looking Monte Carlo Simulation (12-Month Horizon)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **Combined Portfolio** | **SPY Individual** | **QQQ Individual** |
| Total Return | 70.9% | ~12.3%\* | ~11.1%\* |
| Annualized Return | 10.7% | ~2.3%\* | ~2.1%\* |
| Volatility (Annual) | 14.6% | 3.99% | 5.36% |
| Sharpe Ratio | 0.43 | 0.579 | 0.404 |
| Sortino Ratio | 0.56 | 0.915 | 0.462 |
| Maximum Drawdown | -28.3% | -3.6% | -10.6% |

**Table 6:Portfolio Backtest Performance (April 2020 - June 2025)**

The portfolio got higher absolute returns (70.9%) but worse risk-adjusted performance. In addition, the Sharpe ratio collapsed from 0.579 and 0.404 individually to 0.43 combined, while maximum drawdown exploded to -28.3%, which can also be seen in figure 3.

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **Average** | **Min** | **Max** |
| SPY Weight | 29.5% | 15% | 70% |
| QQQ Weight | 40.0% | 0% | 70% |
| Cash Weight | 30.5% | 0% | 60% |
| Correlation (SPY/QQQ) | 0.65 | 0.45 | 0.91 |

**Table 7:Portfolio Allocation Over Time**

The portfolio used Markowitz optimization to choose weights based on forecast returns, volatility, and the SPY vs QQQ correlation. Then, a Monte Carlo risk test would reduce positions if the 95% Expected Shortfall was above the limit. The reason why combined portfolio failed was because SPY and QQQ often moved together since its correlation = 91, while the portfolio maintained high equity exposure SPY = 29.5% and QQQ = 40%, which ultimately eliminated most diversification benefits.

|  |  |  |  |
| --- | --- | --- | --- |
| **Scenario** | **Expected Return** | **Volatility** | **5% VaR** |
| Normal Market | 11.1% | 16.3% | 13.7% |
| High Volatility | 6.2% | 28.3% | 34.0% |
| Bear Market | -10.5% | 17.7% | 36.6% |
| Market Crash | -24.2% | 29.4% | 62.9% |
| Bull Market | 24.4% | 14.6% | 2.0% |

**Table 8:Stress Test Scenarios (12-Month Forward)**

The stress tests reveal portfolio vulnerability to market shocks, with bear markets and crash scenarios producing losses up to −24.2%. This risk is increased by current aggressive positioning (June 2025) of 15% in SPY and 70% in QQQ, which concentrates exposure in two highly correlated index. Even though, the portfolio value grew from $100 to $170.87 it would still subject to high risk since in figure 3, it shows that even frequent cash allocations above 40% were not enough to prevent severe drawdowns. **These results demonstrate** that in multi-asset portfolios, position sizing methods that work for single assets can fail when assets are highly correlation which makes them behave like one position and increases risk.

A close-up of a graph

Description automatically generated

Figure 3: Risk Management Dashboard

### **3.6 Position Sizing Attribution**

A group of graphs with different colored lines

Description automatically generated with medium confidence

**Figure 4: Position Sizing Analysis**

The figure above highlights how the tanh position sizing works in practice. The top row shows the weight distributions, with SPY averaging about 6.5% and QQQ about 11.1% which cover both long and short positions. The bottom row shows S-shaped curves which is the tanh activation function, that link predictions to position sizes. Small predictions near zero lead to small weights, while stronger prediction signals move toward the forty percent limit. The category breakdown shows that about a quarter of positions are small, under ten percent weight, suggesting low conviction, while more than thirty percent are above thirty percent weight, showing high conviction. Even with very low forecast correlations, at 0.086 for SPY and 0.028 for QQQ, the tanh approach can separate weak from strong signals and size positions to get more value from noisy forecasts. This demonstrates that effective position sizing can enhance the utility of weak forecasts by allocating capital in line with signal strength while controlling risk.

## **4. Conclusion**

This research confirms that position sizing has the potential to be more important than prediction accuracy. Despite of low correlations for SPY and QQQ, the tanh position sizing still achieved Sharpe ratios of 0.579 and 0.404. This ultimately challenges the conventional belief that strong signals are necessary for profitable trading and researcher should pay more attention to it.

The ensemble models which using Ridge, ElasticNet, Random Forest, XGBoost, and LightGBM, along with Kalman filtering, meta model blending and GMM regime detection, successfully turned weak signals into viable strategies. Although, it still underperformed buy and hold, the framework maintained stable risk-adjusted returns with minimal drawdowns 3.6% for SPY and 11.8% for QQQ. As a result, it’s creating a useful approach for risk-averse investors seeking low risk returns.

There was a critical issue that was seen in mixed asset portfolios, where performance dropped significantly. The Sharpe ratio dropped to 0.43 and the maximum drawdown peak to -28.3%, due to high correlations between SPY and QQQ. This reveals that position sizing method is effective for single assets can fail when assets are highly correlated, essentially behaving as one large position.

Cross-validation results were stable, with 80% of periods having positive Sharpe ratios which was good, and it varies across different regimes. Overall, the strategy was good at avoiding large losses during downturns, but it still sacrificed its upside potential making it still not perfect.

In the future, research could focus on addressing correlation dynamics in multi-asset frameworks, potentially by incorporating dynamic correlation adjustments or testing with negatively correlated assets. Additionally, exploring different activation functions for position sizing could provide valuable insights.

In conclusion, this research emphasizes the importance of risk management through position sizing can be hugely impactful, and it encourages the exploration of unconventional approaches to generating profits. By leveraging an innovative strategy that hasn’t widely known, there is potential to gain an advantage before the market fully adapts to such methods.

**5. References**

Bailey, D.H. and López de Prado, M., (2014) ‘The deflated Sharpe ratio: Correcting for selection bias, backtest overfitting, and non-normality.’, Journal of Portfolio Management, 40(5), pp.94-107. Available at: <https://doi.org/10.3905/jpm.2014.40.5.094> [Accessed: 1 July 2025].

Campbell, J.Y. and Thompson, S.B., (2008) ‘*Predicting excess stock returns out of sample: Can anything beat the historical average?*’, The Review of Financial Studies, 21(4), pp.1509-1531. Available at: <https://doi.org/10.1093/rfs/hhm055> [Accessed: 20 June 2025].

Goyal, A., Welch, I. and Zafirov, A. (2024) ‘*A Comprehensive 2022 Look at the Empirical Performance of Equity Premium Prediction’*, Review of Financial Studies, 37(11), pp. 3490–3557. Available at: [https://doi:10.1093/rfs/hhae044](https://doi:10.1093/rfs/hhae044%20) [Accessed: 1 June 2025].

Gu, S., Kelly, B. and Xiu, D. (2020) ‘*Empirical Asset Pricing via Machine Learning’*, Review of Financial Studies, 33(5), pp. 2223–2273. Available at: <https://doi:10.1093/rfs/hhaa009> [Accessed: 20 June 2025].

Harvey, C.R., Hoyle, E., Korgaonkar, R., Rattray, S., Sargaison, M. and Van Hemert, O., (2018). *‘The impact of volatility targeting’*. Journal of Portfolio Management, 45(1), pp.14-33. Available at: <https://doi.org/10.3905/jpm.2018.45.1.014> [Accessed: 5 July 2025].

Harvey, C.R., Liu, Y. and Zhu, H., (2016). *‘…and the cross-section of expected returns’*. The Review of Financial Studies, 29(1), pp.5-68. Available at: <https://doi.org/10.1093/rfs/hhv059> [Accessed: 3 August 2025].

López de Prado, M. (2018). ‘*Advances in Financial Machine Learning (Chapter 1)*’. Advances in Financial Machine Learning, 1st ed. Wiley. ISBN: 978-1-119-48208-6. Available at: <https://ssrn.com/abstract=3104847> [Accessed: 5 August 2025].

## 

## **6. Appendices**

**Appendix 1: Heatmap before and after pruning**

A screenshot of a graph

Description automatically generated

**Appendix 1. Figure 1: SPY Heatmap before pruning**

A screenshot of a graph

Description automatically generated

**Appendix 2. Figure 2: QQQ Heatmap before pruning**

A screenshot of a graph

Description automatically generated

**Appendix 3. Figure 3: SPY Heatmap after pruning**

A screenshot of a graph

Description automatically generated

**Appendix 4. Figure 4: QQQ Heatmap after pruning**

**Appendix 2: Model Performance Evaluation**

A group of graphs and diagrams

Description automatically generated

**Appendix 2. Figure 1: SPY Performance**

A group of graphs and diagrams

Description automatically generated

**Appendix 2. Figure 2: QQQ Performance**

**Appendix 3: Cross-Validation Analysis**

A screenshot of a graph

Description automatically generated

**Appendix 3. Figure 1: SPY Cross-Validation**

A group of graphs with different colored lines

Description automatically generated with medium confidence

**Appendix 3. Figure 2: QQQ Cross-Validatio**

## **7. Source Code**

**Listing 1: Pipeline for Feature Building, Model Training, and Evaluation**

import numpy as np,pandas as pd

from sklearn.model\_selection import TimeSeriesSplit

from sklearn.metrics import \*

from sklearn.linear\_model import Ridge,ElasticNet

from sklearn.preprocessing import RobustScaler

from sklearn.ensemble import RandomForestRegressor

from sklearn.mixture import GaussianMixture

from xgboost import XGBRegressor

from lightgbm import LGBMRegressor

from filterpy.kalman import KalmanFilter

from filterpy.common import Q\_discrete\_white\_noise

from scipy import stats

EXCEL="/Users/bo/Desktop/Applied Project /The Final Applied Proj/Applied Project ICL .xlsx"

np.random.seed(42)

TOP\_K=35;MAX\_CORR=0.95;ALPHA\_TANH=1.2;MAX\_WEIGHT=0.4;TRANSACTION\_COST\_BPS=10

EXPORTS={"loaded":"01\_loaded\_full.csv","clean":"02\_clean\_preprune.csv",

"spy":"03\_spy\_pruned.csv","qqq":"04\_qqq\_pruned.csv","metrics":"05\_metrics\_reg.csv"}

def safe\_divide(num,den,default=0.0,eps=1e-10):

if isinstance(den,(pd.Series,pd.DataFrame)):

safe\_den=den.where(den.abs()>eps,np.inf)

result=num/safe\_den

return result.replace([np.inf,-np.inf],default).fillna(default)

return np.where(np.abs(den)>eps,num/den,default)

def calc\_rsi(prices,n=14):

delta=prices.diff()

up=delta.clip(lower=0).ewm(alpha=1/n,adjust=False).mean()

dn=-delta.clip(upper=0).ewm(alpha=1/n,adjust=False).mean()

return 100-100/(1+up/dn)

def get\_macd(prices):

ema12=prices.ewm(span=12,adjust=False).mean();ema26=prices.ewm(span=26,adjust=False).mean()

macd=ema12-ema26;signal=macd.ewm(span=9,adjust=False).mean()

return macd,signal

def build\_feats(excel\_path):

xls=pd.ExcelFile(excel\_path);F=pd.DataFrame()

for ticker in ["SPY","QQQ","VIX"]:

df=pd.read\_excel(xls,ticker)[["Date","Last Price"]]

df["Date"]=pd.to\_datetime(df.Date)

df=df.set\_index("Date").resample("M").ffill();df.columns=[f"{ticker.lower()}\_price"]

F=F.join(df,how="outer") if len(F)>0 else df

for sym in ["spy","qqq"]:

p=F[f"{sym}\_price"];F[f"{sym}\_ret"]=np.log(p/p.shift(1));p\_lag=p.shift(1)

for w in [1,3,6,12]:

F[f"{sym}\_mom\_{w}m"]=p\_lag.pct\_change(w);F[f"{sym}\_mom\_acc\_{w}m"]=F[f"{sym}\_mom\_{w}m"].diff()

for w in [3,6,12]:

F[f"{sym}\_vol\_{w}m"]=F[f"{sym}\_ret"].shift(1).rolling(w).std()

F[f"{sym}\_vol\_change\_{w}m"]=F[f"{sym}\_vol\_{w}m"].pct\_change()

F[f"{sym}\_skew\_{w}m"]=F[f"{sym}\_ret"].shift(1).rolling(w).skew()

F[f"{sym}\_kurt\_{w}m"]=F[f"{sym}\_ret"].shift(1).rolling(w).kurt()

rsi=calc\_rsi(p\_lag);F[f"{sym}\_rsi\_norm"]=(rsi-50)/50

m,s=get\_macd(p\_lag)

F[f"{sym}\_macd\_norm"]=safe\_divide(m,F[f"{sym}\_vol\_3m"]\*np.sqrt(12),0)

F[f"{sym}\_macd\_signal"]=np.sign(m-s)

for w in [3,6,12]:

ma=p\_lag.rolling(w).mean();std=p\_lag.rolling(w).std()

F[f"{sym}\_zscore\_{w}m"]=safe\_divide(p\_lag-ma,std,0)

F[f"{sym}\_ma\_20"]=p\_lag/p\_lag.rolling(20).mean()-1

F[f"{sym}\_ma\_50"]=p\_lag/p\_lag.rolling(50).mean()-1

high=p\_lag.rolling(60).max();low=p\_lag.rolling(60).min()

F[f"{sym}\_high\_3m"]=p\_lag/high-1;F[f"{sym}\_low\_3m"]=p\_lag/low-1

F[f"{sym}\_sharpe\_3m"]=safe\_divide(F[f"{sym}\_ret"].shift(1).rolling(3).mean(),F[f"{sym}\_vol\_3m"],0)\*np.sqrt(12)

F["spread"]=F["spy\_ret"].shift(1)-F["qqq\_ret"].shift(1)

F["correlation\_3m"]=F["spy\_ret"].shift(1).rolling(3).corr(F["qqq\_ret"].shift(1))

F["correlation\_6m"]=F["spy\_ret"].shift(1).rolling(6).corr(F["qqq\_ret"].shift(1))

F["vix\_change"]=F["vix\_price"].shift(1).pct\_change()

F["vix\_zscore"]=(F["vix\_price"].shift(1)-F["vix\_price"].shift(1).rolling(20).mean())/F["vix\_price"].shift(1).rolling(20).std()

Y=pd.DataFrame()

for sh,nm in {"2Y":"y2","10Y":"y10","30Y":"y30"}.items():

y\_data=pd.read\_excel(xls,sh)[["Date","Last Price"]]

y\_data["Date"]=pd.to\_datetime(y\_data.Date);y\_data=y\_data.set\_index("Date").resample("M").ffill()

y\_data.columns=[nm];Y=Y.join(y\_data,how="outer") if len(Y)>0 else y\_data

F=F.join(Y,how="outer")

for col in Y.columns:F[f"{col}\_change"]=F[col].shift(1).pct\_change()

F["term\_10\_2\_change"]=(F["y10"].shift(1)-F["y2"].shift(1)).diff()

for sh,tag in {"CPI":"cpi","ISM PMI":"pmi","UNEMPLOYED":"unemp"}.items():

d=pd.read\_excel(xls,sh)[["Date","Actual Economic Release Values","BN Survey Median"]]

d["Date"]=pd.to\_datetime(d.Date);d=d.set\_index("Date")

s=(d.iloc[:,0]-d.iloc[:,1]).resample("M").ffill();s.name=f"{tag}\_surprise"

F=F.join(s,how="outer")

for lag in [1,3,6]:F[f"{tag}\_lag{lag}"]=s.shift(lag)

F[f"{tag}\_change"]=s.diff();F[f"{tag}\_zscore"]=(s-s.rolling(12).mean())/s.rolling(12).std()

price\_cols=[c for c in F.columns if '\_price' in c or c in ['y2','y10','y30']]

F=F.drop(columns=price\_cols,errors='ignore');F=F.ffill(limit=3)

for c in F.columns:

if any(p in c for p in ['\_surprise','\_zscore']):F[c]=F[c].shift(1)

return F

def load\_data():

F\_all=build\_feats(EXCEL)

F\_all.to\_csv("feature\_matrix\_full.csv")

F\_all.to\_csv(EXPORTS["loaded"])

clean=F\_all.loc[:,F\_all.isna().mean()<0.4].loc[:,F\_all.nunique()>1]

clean.to\_csv(EXPORTS["clean"])

return F\_all,clean

def strict\_prune(df,max\_corr=MAX\_CORR,k=TOP\_K):

w=df.copy()

if w.empty or w.shape[1]==0:return pd.DataFrame()

w=w.dropna(axis=1,how='all')

if w.empty:return pd.DataFrame()

w=w.loc[:,w.std()>1e-10]

variance\_threshold=w.var().quantile(0.02)

low\_var\_features=w.columns[w.var()<=variance\_threshold].tolist()

if low\_var\_features:w=w.drop(columns=low\_var\_features)

if w.empty:return pd.DataFrame()

iteration=0

while len(w.columns)>k and iteration<100:

if w.shape[1]<2:break

corr=w.corr().abs()

if corr.empty:break

np.fill\_diagonal(corr.values,0)

max\_corr\_value=corr.max().max()

if max\_corr\_value<=max\_corr:break

corr\_stack=corr.stack()

if corr\_stack.empty:

avg\_corrs=corr.mean()

if not avg\_corrs.empty:

drop=avg\_corrs.idxmax();w=w.drop(columns=drop)

else:

i,j=corr\_stack.idxmax()

drop=i if w[i].var()<w[j].var() else j

w=w.drop(columns=drop)

iteration+=1

if w.empty:return pd.DataFrame()

if len(w.columns)>k:

scores=pd.Series(index=w.columns)

for col in w.columns:

var\_score=w[col].var();corr\_score=1-w.corr()[col].abs().mean()

unique\_score=w[col].nunique()/len(w)

scores[col]=0.5\*var\_score+0.3\*corr\_score+0.2\*unique\_score

selected\_features=scores.nlargest(k).index.tolist()

w=w[selected\_features]

return w

def build\_ensemble\_model(X\_train,y\_train):

scaler=RobustScaler();X\_scaled=scaler.fit\_transform(X\_train)

models={'ridge':Ridge(alpha=0.5,random\_state=42),'elastic':ElasticNet(alpha=0.3,l1\_ratio=0.7,random\_state=42),

'rf':RandomForestRegressor(n\_estimators=120,max\_depth=7,min\_samples\_split=8,min\_samples\_leaf=3,max\_features='sqrt',random\_state=42,n\_jobs=-1),

'xgb':XGBRegressor(n\_estimators=250,max\_depth=6,learning\_rate=0.04,subsample=0.75,colsample\_bytree=0.75,reg\_alpha=0.05,reg\_lambda=0.3,gamma=0.005,min\_child\_weight=1,random\_state=42,verbosity=0),

'lgbm':LGBMRegressor(n\_estimators=250,max\_depth=6,learning\_rate=0.04,num\_leaves=45,subsample=0.75,colsample\_bytree=0.75,reg\_alpha=0.05,reg\_lambda=0.3,min\_child\_weight=1,random\_state=42,verbosity=-1)}

fitted\_models={};scalers={}

for name,model in models.items():

if name in ['ridge','elastic']:

fitted\_models[name]=model.fit(X\_scaled,y\_train);scalers[name]=scaler

else:

fitted\_models[name]=model.fit(X\_train,y\_train);scalers[name]=None

return fitted\_models,scalers

def ensemble\_predict(fitted\_models,scalers,X\_test):

predictions=[];weights={'ridge':0.08,'elastic':0.07,'rf':0.25,'xgb':0.35,'lgbm':0.25}

all\_preds={}

for name,model in fitted\_models.items():

if scalers.get(name) is not None:X\_use=scalers[name].transform(X\_test)

else:X\_use=X\_test

pred=model.predict(X\_use);all\_preds[name]=pred;predictions.append(pred\*weights.get(name,0.2))

if not predictions:return np.zeros(len(X\_test))

avg\_pred=np.sum(predictions,axis=0)

if 'xgb' in all\_preds and 'lgbm' in all\_preds:

tree\_signal=(all\_preds['xgb']+all\_preds['lgbm'])/2

tree\_dev=tree\_signal-np.mean(tree\_signal);avg\_pred=avg\_pred+0.12\*tree\_dev

return avg\_pred

def fit\_gmm\_regimes(returns,n\_components=3):

features=pd.DataFrame()

features['returns']=returns;features['volatility']=returns.rolling(20).std()

features['momentum']=returns.rolling(20).mean();features=features.dropna()

if len(features)<n\_components\*10:return None,None

gmm=GaussianMixture(n\_components=n\_components,covariance\_type='full',random\_state=42,n\_init=10)

gmm.fit(features);regimes=gmm.predict(features);probs=gmm.predict\_proba(features)

return regimes,probs

def fit\_meta\_model\_continuous(X\_train,y\_train,fitted\_models,scalers):

n\_splits=min(4,len(X\_train)//15)

if n\_splits<2:return None,1.0

tscv=TimeSeriesSplit(n\_splits=n\_splits);oof\_preds=[];oof\_actual=[]

for train\_idx,val\_idx in tscv.split(X\_train):

X\_tr,X\_val=X\_train.iloc[train\_idx],X\_train.iloc[val\_idx]

y\_tr,y\_val=y\_train.iloc[train\_idx],y\_train.iloc[val\_idx]

temp\_models,temp\_scalers=build\_ensemble\_model(X\_tr,y\_tr)

fold\_pred=ensemble\_predict(temp\_models,temp\_scalers,X\_val)

oof\_preds.extend(fold\_pred);oof\_actual.extend(y\_val.values)

if len(oof\_preds)<10:return None,1.0

oof\_preds=np.array(oof\_preds);oof\_actual=np.array(oof\_actual)

mean\_pred=np.mean(oof\_actual);best\_alpha=1.0;best\_mse=np.mean((oof\_preds-oof\_actual)\*\*2)

for alpha in np.linspace(0.8,1.0,5):

blended=alpha\*oof\_preds+(1-alpha)\*mean\_pred;mse=np.mean((blended-oof\_actual)\*\*2)

if mse<best\_mse:best\_mse=mse;best\_alpha=alpha

return mean\_pred,best\_alpha

def apply\_kalman\_smoothing(predictions):

kf=KalmanFilter(dim\_x=2,dim\_z=1)

kf.F=np.array([[1,1],[0,1]]);kf.H=np.array([[1,0]]);kf.R=0.05

kf.Q=Q\_discrete\_white\_noise(2,dt=1,var=0.02);kf.x=np.array([0,0]);kf.P\*=100

smoothed=np.zeros\_like(predictions)

for i,z in enumerate(predictions):kf.predict();kf.update(z);smoothed[i]=kf.x[0]

return smoothed

def calculate\_weights\_continuous(predictions,train\_sigma=None,gmm\_probs=None,alpha=ALPHA\_TANH,max\_weight=MAX\_WEIGHT):

if not np.all(np.isfinite(predictions)):predictions=np.nan\_to\_num(predictions,nan=0.0)

pct\_rank=pd.Series(predictions).rank(pct=True).values

if train\_sigma is not None and train\_sigma>1e-10:

z\_scores=predictions/train\_sigma

z\_scores=np.where(pct\_rank>0.7,z\_scores\*1.3,z\_scores)

z\_scores=np.where(pct\_rank<0.3,z\_scores\*1.3,z\_scores)

else:z\_scores=(pct\_rank-0.5)\*3

z\_weights=np.tanh(z\_scores\*alpha);weights=z\_weights\*max\_weight

if gmm\_probs is not None and len(gmm\_probs)==len(weights):

confidence=np.max(gmm\_probs,axis=1);weights=weights\*confidence

strong\_long=predictions>np.percentile(predictions,75)

strong\_short=predictions<np.percentile(predictions,25)

weights[strong\_long]=np.clip(weights[strong\_long],0.2\*max\_weight,max\_weight)

weights[strong\_short]=np.clip(weights[strong\_short],-max\_weight,-0.2\*max\_weight)

return np.clip(weights,-max\_weight,max\_weight)

def create\_aligned\_targets(df,ret\_col,forecast\_horizon=1):

df=df.copy().sort\_index()

if len(df)<forecast\_horizon+5:return pd.DataFrame()

df['y']=df[ret\_col].shift(-forecast\_horizon)

if forecast\_horizon>0:df=df.iloc[:-forecast\_horizon]

df=df.drop(columns=[ret\_col],errors="ignore");df\_clean=df.dropna(subset=['y'])

return df\_clean

def run\_ticker(tkr):

global F\_all,clean

ret\_col=f"{tkr.lower()}\_ret";other="qqq" if tkr=="SPY" else "spy";price\_col=f"{tkr.lower()}\_price"

if price\_col in F\_all.columns:

returns=F\_all[price\_col].pct\_change().dropna()

gmm\_regimes,gmm\_probs=fit\_gmm\_regimes(returns,n\_components=3)

else:gmm\_regimes,gmm\_probs=None,None

exclude\_patterns=[f"{other}\_price",f"{other}\_ret"]

base=clean.drop(columns=[c for c in clean.columns if any(pattern in c for pattern in exclude\_patterns)],errors="ignore")

if 'vix\_level' in base.columns and 'spy\_qqq\_ratio' in base.columns:base['vix\_ratio\_interaction']=base['vix\_level']\*base['spy\_qqq\_ratio']

if 'term\_10\_2' in base.columns and 'vix\_level' in base.columns:base['yield\_vix\_interaction']=base['term\_10\_2']\*base['vix\_level']

if len(base.columns)>TOP\_K\*2:

feature\_scores=pd.DataFrame(index=base.columns)

variance=base.var();feature\_scores['variance\_norm']=(variance-variance.min())/(variance.max()-variance.min()+1e-10)

corr\_matrix=base.corr().abs();np.fill\_diagonal(corr\_matrix.values,0)

feature\_scores['avg\_corr']=corr\_matrix.mean(axis=1);feature\_scores['diversity']=1-feature\_scores['avg\_corr']

feature\_scores['combined']=0.6\*feature\_scores['variance\_norm']+0.4\*feature\_scores['diversity']

selected\_features=feature\_scores['combined'].nlargest(min(int(TOP\_K\*1.5),len(base.columns))).index.tolist()

base=base[selected\_features]

if ret\_col not in base.columns:

if ret\_col in F\_all.columns:df=base.join(F\_all[[ret\_col]],how="inner").dropna()

else:return {}

else:df=base.dropna()

if len(df)<50:return {}

df\_with\_targets=create\_aligned\_targets(df,ret\_col,forecast\_horizon=1)

if len(df\_with\_targets)==0:return {}

if df\_with\_targets['y'].isna().any():df\_with\_targets=df\_with\_targets.dropna(subset=['y'])

if len(df\_with\_targets)<30:return {}

df=df\_with\_targets;cut=int(len(df)\*0.75);tr,te=df.iloc[:cut],df.iloc[cut:]

if len(tr)<20 or len(te)<5:return {}

sel\_cols=strict\_prune(tr.drop(columns="y"))

if len(sel\_cols.columns)==0:return {}

sel\_cols.to\_csv(EXPORTS[tkr.lower()])

Xtr,ytr=tr[sel\_cols.columns],tr["y"];Xte,yte=te[sel\_cols.columns],te["y"]

if Xtr.isna().any().any() or ytr.isna().any():

train\_clean\_mask=~(Xtr.isna().any(axis=1)|ytr.isna())

Xtr=Xtr[train\_clean\_mask];ytr=ytr[train\_clean\_mask]

if Xte.isna().any().any() or yte.isna().any():

test\_clean\_mask=~(Xte.isna().any(axis=1)|yte.isna())

Xte=Xte[test\_clean\_mask];yte=yte[test\_clean\_mask]

if len(Xtr)<10 or len(ytr)<10 or len(Xte)<3 or len(yte)<3:return {}

feature\_analysis=pd.DataFrame({'feature':sel\_cols.columns,'variance':sel\_cols.var().values,

'mean\_abs\_value':sel\_cols.abs().mean().values,

'correlation\_with\_target':[abs(tr[col].corr(tr["y"])) if col in tr.columns else 0 for col in sel\_cols.columns]}).sort\_values('correlation\_with\_target',ascending=False)

feature\_analysis.to\_csv(f"{tkr.lower()}\_selected\_features.csv",index=False)

fitted\_models,scalers=build\_ensemble\_model(Xtr,ytr)

mu\_pred=ensemble\_predict(fitted\_models,scalers,Xte)

if not np.all(np.isfinite(mu\_pred)):mu\_pred=np.nan\_to\_num(mu\_pred,nan=0.0)

percentile\_rank=pd.Series(mu\_pred).rank(pct=True).values

mu\_pred[percentile\_rank<0.4]=mu\_pred[percentile\_rank<0.4]-0.008

mu\_pred[percentile\_rank>0.6]=mu\_pred[percentile\_rank>0.6]+0.008

mu\_pred[(percentile\_rank>=0.4)&(percentile\_rank<=0.6)]=mu\_pred[(percentile\_rank>=0.4)&(percentile\_rank<=0.6)]\*0.5

mean\_target,blend\_alpha=fit\_meta\_model\_continuous(Xtr,ytr,fitted\_models,scalers)

if mean\_target is not None and blend\_alpha<1.0:mu\_pred=blend\_alpha\*mu\_pred+(1-blend\_alpha)\*mean\_target

mu\_smooth=apply\_kalman\_smoothing(mu\_pred)

train\_preds=ensemble\_predict(fitted\_models,scalers,Xtr);train\_sigma=np.std(train\_preds)+1e-12

test\_gmm\_probs=None

if gmm\_probs is not None:

gmm\_index=pd.Series(gmm\_probs).index

test\_indices=[i for i,idx in enumerate(gmm\_index) if idx in yte.index]

if len(test\_indices)>0:test\_gmm\_probs=gmm\_probs[test\_indices]

w=calculate\_weights\_continuous(mu\_smooth,train\_sigma=train\_sigma,gmm\_probs=test\_gmm\_probs)

out=pd.DataFrame({"date":yte.index,"mu\_pred":mu\_pred,"mu\_kalman":mu\_smooth,"mu":mu\_smooth,

"weight":w,"y\_true":yte.values,"sigma":train\_sigma},index=yte.index)

out.to\_csv(f"{tkr.lower()}\_forecast\_detail.csv")

monte\_carlo\_df=pd.DataFrame({"mu":mu\_smooth,"mu\_kalman":mu\_smooth,"sigma":np.ones(len(mu\_smooth))\*train\_sigma,"weight":w},index=yte.index)

monte\_carlo\_df.to\_csv(f"{tkr.lower()}\_monte\_carlo\_inputs.csv")

yte\_values=yte.values;strat\_ret=w\*yte\_values

mse=mean\_squared\_error(yte\_values,mu\_smooth);mae=mean\_absolute\_error(yte\_values,mu\_smooth)

ss\_res=np.sum((yte\_values-mu\_smooth)\*\*2);ss\_tot=np.sum((yte\_values-np.mean(yte\_values))\*\*2)

r2=1-(ss\_res/(ss\_tot+1e-10))

if len(yte\_values)>1:correlation=np.corrcoef(yte\_values,mu\_smooth)[0,1];ic=stats.spearmanr(mu\_smooth,yte\_values)[0]

else:correlation=0;ic=0

hit\_rate=(np.sign(mu\_smooth)==np.sign(yte\_values)).mean()

y\_binary=(yte\_values>0).astype(int)

if len(np.unique(y\_binary))>1:roc\_auc\_val=roc\_auc\_score(y\_binary,mu\_smooth)

else:roc\_auc\_val=0.5

if len(strat\_ret)>1 and strat\_ret.std()>1e-10:sharpe=(strat\_ret.mean()/strat\_ret.std())\*np.sqrt(12)

else:sharpe=0

weight\_series=pd.Series(w,index=yte.index)

tc=weight\_series.diff().abs().fillna(weight\_series.abs())\*(TRANSACTION\_COST\_BPS/10000)

strat\_ret\_net=pd.Series(strat\_ret,index=yte.index)-tc

sharpe\_net=(strat\_ret\_net.mean()/strat\_ret\_net.std())\*np.sqrt(12) if strat\_ret\_net.std()>0 else 0

cum\_returns=pd.Series((1+strat\_ret).cumprod())

max\_dd=(cum\_returns/cum\_returns.expanding().max()-1).min()

metrics={"MSE":mse,"MAE":mae,"R2":r2,"Correlation":correlation,"IC":ic,"Hit\_Rate":hit\_rate,

"ROC\_AUC":roc\_auc\_val,"Sharpe":sharpe,"Sharpe\_Net":sharpe\_net,"Max\_DD":max\_dd,"Avg\_Weight":np.abs(w).mean()}

return metrics

def walk\_forward\_validation(tkr,n\_splits=3):

global F\_all,clean

ret\_col=f"{tkr.lower()}\_ret";other="qqq" if tkr=="SPY" else "spy"

exclude\_patterns=[f"{other}\_price",f"{other}\_ret"]

base=clean.drop(columns=[c for c in clean.columns if any(pattern in c for pattern in exclude\_patterns)],errors="ignore")

if ret\_col not in base.columns:

if ret\_col in F\_all.columns:df=base.join(F\_all[[ret\_col]],how="inner").dropna()

else:return 0,0

else:df=base.dropna()

df=create\_aligned\_targets(df,ret\_col,forecast\_horizon=1)

if len(df)<n\_splits\*24:return 0,0

tscv=TimeSeriesSplit(n\_splits=n\_splits,test\_size=max(6,int(len(df)/(n\_splits\*2))))

cv\_scores=[]

for fold,(train\_idx,test\_idx) in enumerate(tscv.split(df)):

train\_data=df.iloc[train\_idx];test\_data=df.iloc[test\_idx]

if len(train\_data)<20 or len(test\_data)<3:continue

sel\_cols=strict\_prune(train\_data.drop(columns="y"))

if len(sel\_cols.columns)==0:continue

Xtr,ytr=train\_data[sel\_cols.columns],train\_data["y"];Xte,yte=test\_data[sel\_cols.columns],test\_data["y"]

if Xtr.isna().any().any() or ytr.isna().any():

mask=~(Xtr.isna().any(axis=1)|ytr.isna());Xtr,ytr=Xtr[mask],ytr[mask]

if Xte.isna().any().any() or yte.isna().any():

mask=~(Xte.isna().any(axis=1)|yte.isna());Xte,yte=Xte[mask],yte[mask]

if len(Xtr)<10 or len(Xte)<2:continue

fitted\_models,scalers=build\_ensemble\_model(Xtr,ytr)

mu\_pred=ensemble\_predict(fitted\_models,scalers,Xte)

train\_preds=ensemble\_predict(fitted\_models,scalers,Xtr);train\_sigma=np.std(train\_preds)+1e-12

if not np.all(np.isfinite(mu\_pred)):mu\_pred=np.nan\_to\_num(mu\_pred,nan=0.0)

mu\_smooth=apply\_kalman\_smoothing(mu\_pred);fold\_sharpe=0

if len(yte)>1:

weights=calculate\_weights\_continuous(mu\_smooth,train\_sigma=train\_sigma)

strat\_ret=weights\*yte.values

if strat\_ret.std()>1e-10:fold\_sharpe=(strat\_ret.mean()/strat\_ret.std())\*np.sqrt(12)

cv\_scores.append(fold\_sharpe)

if cv\_scores:return np.mean(cv\_scores),np.std(cv\_scores)

else:return 0,0

def run\_main\_pipeline():

global F\_all,clean

F\_all,clean=load\_data();metrics={};cv\_results={}

for t in ("SPY","QQQ"):

result=run\_ticker(t)

if result:

metrics[t]=result

cv\_mean,cv\_std=walk\_forward\_validation(t,n\_splits=3)

cv\_results[t]={"cv\_sharpe\_mean":cv\_mean,"cv\_sharpe\_std":cv\_std}

if metrics:

summary\_df=pd.DataFrame(metrics).T.round(4);summary\_df.to\_csv(EXPORTS["metrics"])

if cv\_results:cv\_df=pd.DataFrame(cv\_results).T.round(4);cv\_df.to\_csv("cross\_validation\_results.csv")

return metrics,cv\_results

if \_\_name\_\_ == "\_\_main\_\_":

results, cv\_results = run\_main\_pipeline()

# Print metrics

if results:

print("\n" + "="\*60)

print("MODEL RESULTS")

print("="\*60)

for ticker in results.keys():

print(f"\n{ticker} METRICS:")

print("-"\*30)

for metric, value in results[ticker].items():

if isinstance(value, float):

print(f"{metric:15s}: {value:8.4f}")

else:

print(f"{metric:15s}: {value}")

# Print summary DataFrame

print("\n" + "="\*60)

print("SUMMARY TABLE")

print("="\*60)

summary\_df = pd.DataFrame(results).T.round(4)

print(summary\_df)

# Print CV results

if cv\_results:

print("\n" + "="\*60)

print("CROSS-VALIDATION RESULTS")

print("="\*60)

cv\_df = pd.DataFrame(cv\_results).T.round(4)

print(cv\_df)

for ticker in cv\_results.keys():

mean = cv\_results[ticker]['cv\_sharpe\_mean']

std = cv\_results[ticker]['cv\_sharpe\_std']

print(f"\n{ticker}: CV Sharpe = {mean:.3f} ± {std:.3f}")

print("\n" + "="\*60)

print("FILES SAVED:")

print("="\*60)

print("- 05\_metrics\_reg.csv")

print("- cross\_validation\_results.csv")

print("- spy\_forecast\_detail.csv")

print("- qqq\_forecast\_detail.csv")

print("- spy\_monte\_carlo\_inputs.csv")

print("- qqq\_monte\_carlo\_inputs.csv")

**Listing 2: Monte Carlo Simulations for Portfolio Optimization and Stress Testing**

import matplotlib.pyplot as plt

from scipy.optimize import minimize\_scalar

import seaborn as sns

import warnings

warnings.filterwarnings('ignore')

np.random.seed(42)

N\_SIMS=2000;MAX\_LOSS\_PCT=0.12;MAX\_POSITION=0.7;MIN\_POSITION=0.15

TRANSACTION\_COST=0.001;VOL\_WINDOW=12;MIN\_VOL=0.08;MAX\_VOL=0.35

RISK\_FREE\_RATE=0.045;DEFAULT\_CORRELATION=0.65;REBALANCE\_THRESHOLD=0.08

MIN\_EQUITY\_ALLOCATION=0.40;KELLY\_FRACTION=0.35

def load\_forecast\_data():

data={}

for ticker in ['SPY','QQQ']:

df=pd.read\_csv(f"{ticker.lower()}\_forecast\_detail.csv",index\_col=0,parse\_dates=True)

data[ticker]=df.sort\_index()

vix=pd.Series()

features=pd.read\_csv("01\_loaded\_full.csv",index\_col=0,parse\_dates=True)

if 'vix\_price' in features.columns:

vix=features['vix\_price']/100;vix=vix.dropna()

return data,vix

def estimate\_volatility(returns,vix\_data,date):

estimated\_vol=None

if vix\_data is not None and len(vix\_data)>0:

vix\_vol=vix\_data.asof(date)

if not pd.isna(vix\_vol) and vix\_vol>0:estimated\_vol=vix\_vol\*0.75

if estimated\_vol is None:

historical=returns.loc[:date].tail(VOL\_WINDOW\*2)

if len(historical)>=6:

weights=np.exp(np.linspace(-1,0,len(historical)));weights/=weights.sum()

mean\_return=np.average(historical,weights=weights)

variance=np.average((historical-mean\_return)\*\*2,weights=weights)

hist\_vol=np.sqrt(variance)\*np.sqrt(12);estimated\_vol=hist\_vol

else:estimated\_vol=0.15

return np.clip(estimated\_vol,MIN\_VOL,MAX\_VOL)

def estimate\_correlation(data1,data2,date):

hist1=data1.loc[:date];hist2=data2.loc[:date]

lookback=min(36,len(hist1),len(hist2))

if lookback<12:return DEFAULT\_CORRELATION

recent1=hist1.tail(lookback).values;recent2=hist2.tail(lookback).values

min\_len=min(len(recent1),len(recent2))

recent1=recent1[-min\_len:];recent2=recent2[-min\_len:]

if len(recent1)<12:return DEFAULT\_CORRELATION

corr=np.corrcoef(recent1,recent2)[0,1]

return np.clip(float(corr),-0.95,0.95)

def monte\_carlo\_sim\_improved(params,n\_paths=N\_SIMS,n\_periods=1):

df=5;z=stats.t.rvs(df,size=(n\_paths,n\_periods));z=z/np.sqrt(df/(df-2))

drift=params.get('drift',0);vol=params.get('vol',0.15);weight=params.get('weight',0)

returns=np.zeros((n\_paths,n\_periods))

for t in range(n\_periods):

monthly\_vol=vol/np.sqrt(12)

jump\_prob=0.02;jump\_size=np.random.choice([-0.05,0.05],size=n\_paths)

jumps=np.random.binomial(1,jump\_prob,n\_paths)\*jump\_size

returns[:,t]=np.exp(drift-0.5\*monthly\_vol\*\*2+monthly\_vol\*z[:,t]+jumps)-1

if n\_periods>1:

portfolio\_value=np.cumprod(1+weight\*returns,axis=1);total\_return=portfolio\_value[:,-1]-1

else:total\_return=weight\*returns[:,0]

return total\_return

def calculate\_es95(returns):

if len(returns)<50:return np.nan

sorted\_returns=np.sort(returns)

cutoff\_idx=max(1,int(0.05\*len(sorted\_returns)))

cutoff\_idx=max(cutoff\_idx,max(5,int(0.01\*len(sorted\_returns))))

worst\_returns=sorted\_returns[:cutoff\_idx]

return -np.mean(worst\_returns)

def optimize\_weight\_kelly(drift,vol):

excess\_drift=drift-(RISK\_FREE\_RATE/12)

if vol<=0 or excess\_drift<=-0.01:return MIN\_POSITION

kelly\_full=excess\_drift/(vol\*\*2);kelly\_weight=kelly\_full\*KELLY\_FRACTION

if vol>0.25:kelly\_weight\*=0.7

return np.clip(kelly\_weight,MIN\_POSITION,MAX\_POSITION)

def optimize\_weight\_with\_es(drift,vol):

excess\_drift=drift-(RISK\_FREE\_RATE/12)

if excess\_drift<-0.02 or vol>0.40:return MIN\_POSITION

weights\_to\_test=np.linspace(MIN\_POSITION,MAX\_POSITION,15);valid\_weights=[]

for w in weights\_to\_test:

params={'drift':excess\_drift,'vol':vol,'weight':w}

sim\_returns=monte\_carlo\_sim\_improved(params,n\_paths=500,n\_periods=1)

es=calculate\_es95(sim\_returns)

if not np.isnan(es) and es<=MAX\_LOSS\_PCT:valid\_weights.append(w)

if not valid\_weights:return MIN\_POSITION

max\_safe\_weight=max(valid\_weights);kelly\_weight=optimize\_weight\_kelly(drift,vol)

return min(kelly\_weight,max\_safe\_weight)

def risk\_parity\_weights(vols):

vols=np.array(vols,dtype=float)

if len(vols)==0 or any(v<=0 for v in vols):

n=len(vols) if len(vols)>0 else 2

return np.ones(n)\*(MIN\_EQUITY\_ALLOCATION/n)

inv\_vols=1/vols;raw\_weights=inv\_vols/inv\_vols.sum()

target\_vol=0.12;portfolio\_vol=np.sqrt(np.sum((raw\_weights\*vols)\*\*2))

if portfolio\_vol>0:

scale=min(1.2,target\_vol/portfolio\_vol);raw\_weights\*=scale

weights=np.maximum(raw\_weights,MIN\_POSITION);weights=np.clip(weights,MIN\_POSITION,MAX\_POSITION)

if weights.sum()<MIN\_EQUITY\_ALLOCATION:weights=weights\*(MIN\_EQUITY\_ALLOCATION/weights.sum())

return weights

def markowitz\_optimization(drifts,vols,corr=DEFAULT\_CORRELATION):

n=len(drifts)

if n==1:return np.array([optimize\_weight\_with\_es(drifts[0],vols[0])])

drifts=np.array(drifts,dtype=float);vols=np.array(vols,dtype=float)

excess\_drifts=drifts-(RISK\_FREE\_RATE/12)

corr=float(corr) if not np.isnan(corr) else DEFAULT\_CORRELATION

if any(v<=0 for v in vols):return risk\_parity\_weights(vols)

mu=excess\_drifts

cov\_matrix=np.array([[vols[0]\*\*2,corr\*vols[0]\*vols[1]],[corr\*vols[0]\*vols[1],vols[1]\*\*2]],dtype=float)

min\_eigenvalue=np.min(np.linalg.eigvals(cov\_matrix))

if min\_eigenvalue<=0:cov\_matrix+=np.eye(n)\*abs(min\_eigenvalue)\*1.1

inv\_cov=np.linalg.inv(cov\_matrix)

risk\_aversion=1.5

numerator=inv\_cov@mu;denominator=risk\_aversion

raw\_weights=numerator/denominator

weights=np.maximum(raw\_weights,MIN\_POSITION);weights=np.minimum(weights,MAX\_POSITION)

if weights.sum()<MIN\_EQUITY\_ALLOCATION:

weights=weights\*(MIN\_EQUITY\_ALLOCATION/weights.sum());weights=np.minimum(weights,MAX\_POSITION)

if weights.sum()>1:weights=weights/weights.sum()

return weights

def run\_portfolio\_backtest(forecast\_data,vix\_data):

all\_dates=sorted(set().union(\*[data.index for data in forecast\_data.values()]))

print(f"\nBacktest: {all\_dates[0].strftime('%Y-%m')} to {all\_dates[-1].strftime('%Y-%m')}")

print(f"Total months: {len(all\_dates)}")

results=[];prev\_weights={};portfolio\_value=100

for i,date in enumerate(all\_dates):

tickers\_available=[];drifts=[];vols=[];actual\_returns={}

for ticker,data in forecast\_data.items():

if date in data.index:

drift=data.loc[date,'mu\_kalman'];actual\_return=data.loc[date,'y\_true']

returns=data['y\_true'];vol=estimate\_volatility(returns,vix\_data,date)

tickers\_available.append(ticker);drifts.append(drift);vols.append(vol)

actual\_returns[ticker]=actual\_return

if not tickers\_available:continue

corr=DEFAULT\_CORRELATION

if len(tickers\_available)>=2 and 'SPY' in tickers\_available and 'QQQ' in tickers\_available:

spy\_returns=forecast\_data['SPY']['y\_true'];qqq\_returns=forecast\_data['QQQ']['y\_true']

corr=estimate\_correlation(spy\_returns,qqq\_returns,date)

if len(tickers\_available)==1:weights=np.array([optimize\_weight\_with\_es(drifts[0],vols[0])])

else:

weights=markowitz\_optimization(drifts,vols,corr)

portfolio\_sim=np.zeros(1000)

for j,ticker in enumerate(tickers\_available):

params={'drift':drifts[j],'vol':vols[j],'weight':weights[j]}

portfolio\_sim+=monte\_carlo\_sim\_improved(params,n\_paths=1000)

es=calculate\_es95(portfolio\_sim)

if not np.isnan(es) and es>MAX\_LOSS\_PCT:

scale=MAX\_LOSS\_PCT/es\*0.9;weights\*=scale

target\_weights=dict(zip(tickers\_available,weights))

should\_rebalance=False

for ticker in target\_weights:

current=prev\_weights.get(ticker,0);target=target\_weights.get(ticker,0)

if abs(current-target)>REBALANCE\_THRESHOLD:should\_rebalance=True;break

if not should\_rebalance:current\_weights=prev\_weights.copy()

else:current\_weights=target\_weights

portfolio\_return=0;total\_equity\_weight=0

for ticker in tickers\_available:

weight=current\_weights.get(ticker,0)

portfolio\_return+=weight\*actual\_returns[ticker];total\_equity\_weight+=weight

cash\_weight=max(0,1-total\_equity\_weight)

if cash\_weight>0:portfolio\_return+=cash\_weight\*(RISK\_FREE\_RATE/12)

trans\_cost=0

for ticker in ['SPY','QQQ']:

curr\_w=current\_weights.get(ticker,0);prev\_w=prev\_weights.get(ticker,0)

trans\_cost+=abs(curr\_w-prev\_w)\*TRANSACTION\_COST

net\_return=portfolio\_return-trans\_cost;portfolio\_value\*=(1+net\_return)

portfolio\_sim=np.zeros(N\_SIMS)

for j,ticker in enumerate(tickers\_available):

if j<len(drifts):

params={'drift':drifts[j],'vol':vols[j],'weight':current\_weights.get(ticker,0)}

asset\_sim=monte\_carlo\_sim\_improved(params,N\_SIMS)

portfolio\_sim+=asset\_sim

es\_95=calculate\_es95(portfolio\_sim)

var\_95=-np.percentile(portfolio\_sim,5) if len(portfolio\_sim)>0 else np.nan

results.append({'Date':date,'Portfolio\_Value':portfolio\_value,'Net\_Return':net\_return,

'Gross\_Return':portfolio\_return,'Transaction\_Cost':trans\_cost,'ES\_95':es\_95,'VaR\_95':var\_95,

'SPY\_Weight':current\_weights.get('SPY',0),'QQQ\_Weight':current\_weights.get('QQQ',0),

'Cash\_Weight':max(0,1-sum(current\_weights.values())),'Correlation':corr,

'SPY\_Vol':vols[tickers\_available.index('SPY')] if 'SPY' in tickers\_available else np.nan,

'QQQ\_Vol':vols[tickers\_available.index('QQQ')] if 'QQQ' in tickers\_available else np.nan})

prev\_weights=current\_weights.copy()

results\_df=pd.DataFrame(results).set\_index('Date')

results\_df['Cumulative']=results\_df['Portfolio\_Value']/100

rolling\_max=results\_df['Cumulative'].expanding().max()

results\_df['Drawdown']=(results\_df['Cumulative']/rolling\_max-1)\*100

return results\_df

def monte\_carlo\_visualization(forecast\_data,vix\_data):

print("\nMonte Carlo Analysis...")

latest\_params={}

for ticker,data in forecast\_data.items():

recent\_drift=data['mu\_kalman'].tail(6).mean()

recent\_returns=data['y\_true'].tail(12)

recent\_vol=estimate\_volatility(recent\_returns,vix\_data,data.index[-1])

latest\_params[ticker]={'drift':recent\_drift,'vol':recent\_vol}

print(f"{ticker}: drift={recent\_drift:.3%}, vol={recent\_vol:.1%}")

n\_paths=1000;n\_months=12

if len(latest\_params)==2:

drifts=[float(p['drift']) for p in latest\_params.values()]

vols=[float(p['vol']) for p in latest\_params.values()]

spy\_returns=forecast\_data['SPY']['y\_true'];qqq\_returns=forecast\_data['QQQ']['y\_true']

common\_dates=spy\_returns.index.intersection(qqq\_returns.index)

if len(common\_dates)>0:

last\_common\_date=common\_dates[-1];corr=estimate\_correlation(spy\_returns,qqq\_returns,last\_common\_date)

else:corr=DEFAULT\_CORRELATION

corr=float(corr)

optimal\_weights=markowitz\_optimization(drifts,vols,corr)

weights={'SPY':optimal\_weights[0],'QQQ':optimal\_weights[1]}

print(f"Correlation: {corr:.3f}")

else:weights={'SPY':0.35,'QQQ':0.35}

print(f"Weights: SPY={weights['SPY']:.1%}, QQQ={weights.get('QQQ',0):.1%}")

paths=np.zeros((n\_paths,n\_months+1));paths[:,0]=100

np.random.seed(42)

for t in range(n\_months):

z1=np.random.randn(n\_paths);z2=corr\*z1+np.sqrt(1-corr\*\*2)\*np.random.randn(n\_paths)

monthly\_returns=np.zeros(n\_paths)

if 'SPY' in latest\_params:

params=latest\_params['SPY'];vol\_m=params['vol']/np.sqrt(12);drift\_m=params['drift']

ret=np.exp(drift\_m-0.5\*vol\_m\*\*2+vol\_m\*z1)-1

monthly\_returns+=weights.get('SPY',0)\*ret

if 'QQQ' in latest\_params:

params=latest\_params['QQQ'];vol\_m=params['vol']/np.sqrt(12);drift\_m=params['drift']

ret=np.exp(drift\_m-0.5\*vol\_m\*\*2+vol\_m\*z2)-1

monthly\_returns+=weights.get('QQQ',0)\*ret

paths[:,t+1]=paths[:,t]\*(1+monthly\_returns)

fig,(ax1,ax2)=plt.subplots(1,2,figsize=(16,6))

months=np.arange(n\_months+1);final\_values=paths[:,-1];sorted\_indices=np.argsort(final\_values)

for i in range(n\_paths):

idx=sorted\_indices[i]

if final\_values[idx]<92:color,alpha='darkred',0.3

elif final\_values[idx]<100:color,alpha='lightcoral',0.3

elif final\_values[idx]<108:color,alpha='lightblue',0.3

elif final\_values[idx]<115:color,alpha='lightgreen',0.3

else:color,alpha='darkgreen',0.3

ax1.plot(months,paths[idx],color=color,linewidth=0.3,alpha=alpha)

percentiles=[5,25,50,75,95];pct\_values=np.percentile(paths,percentiles,axis=0)

ax1.plot(months,pct\_values[0],'r--',linewidth=2,label='5th percentile')

ax1.plot(months,pct\_values[1],'orange',linewidth=2,label='25th percentile')

ax1.plot(months,pct\_values[2],'b-',linewidth=3,label='Median',zorder=5)

ax1.plot(months,pct\_values[3],'lightgreen',linewidth=2,label='75th percentile')

ax1.plot(months,pct\_values[4],'g--',linewidth=2,label='95th percentile')

ax1.axhline(100,color='black',linestyle=':',alpha=0.5,linewidth=1)

ax1.set\_xlabel('Months',fontsize=12);ax1.set\_ylabel('Portfolio Value ($)',fontsize=12)

ax1.set\_title('Monte Carlo Simulation (1000 paths, 12 months)',fontsize=14,fontweight='bold')

ax1.legend(loc='upper left',fontsize=10);ax1.grid(True,alpha=0.3);ax1.set\_xlim(0,12)

ax2.hist(final\_values,bins=50,alpha=0.7,color='skyblue',edgecolor='black',linewidth=0.5)

mean\_val=np.mean(final\_values);var\_5=np.percentile(final\_values,5);es\_5=np.mean(final\_values[final\_values<=var\_5])

ax2.axvline(mean\_val,color='blue',linestyle='--',linewidth=2,label=f'Mean: ${mean\_val:.1f}')

ax2.axvline(var\_5,color='orange',linestyle='--',linewidth=2,label=f'5% VaR: ${var\_5:.1f}')

ax2.axvline(es\_5,color='red',linestyle='--',linewidth=2,label=f'5% ES: ${es\_5:.1f}')

ax2.axvline(100,color='black',linestyle=':',linewidth=2,alpha=0.5,label='Initial: $100')

ax2.set\_xlabel('Final Portfolio Value ($)',fontsize=12);ax2.set\_ylabel('Frequency',fontsize=12)

ax2.set\_title('Distribution of Final Values',fontsize=14,fontweight='bold')

ax2.legend(loc='upper right',fontsize=10);ax2.grid(True,alpha=0.3)

plt.tight\_layout();plt.savefig('monte\_carlo\_analysis.png',dpi=150,bbox\_inches='tight');plt.show()

returns=(final\_values/100-1)\*100

print(f"\nMONTE CARLO RESULTS (12-month)")

print(f"Expected Return: {np.mean(returns):>8.1f}%")

print(f"Volatility: {np.std(returns):>8.1f}%")

print(f"Sharpe Ratio: {(np.mean(returns)-RISK\_FREE\_RATE)/np.std(returns) if np.std(returns)>0 else 0:>8.2f}")

print(f"95% VaR: {-np.percentile(returns,5):>8.1f}%")

print(f"95% CVaR (ES): {-np.mean(returns[returns<=np.percentile(returns,5)]):>8.1f}%")

print(f"Probability of Loss: {(returns<0).mean():>8.1%}")

def risk\_dashboard(results):

fig=plt.figure(figsize=(16,10));gs=fig.add\_gridspec(3,3,hspace=0.3,wspace=0.3)

ax1=fig.add\_subplot(gs[0,:2])

ax1.plot(results['Cumulative'],linewidth=2,color='#2E86AB')

ax1.fill\_between(results.index,1,results['Cumulative'],alpha=0.3,color='#2E86AB')

ax1.axhline(1,color='black',linestyle=':',alpha=0.5)

ax1.set\_title('Cumulative Performance',fontsize=12,fontweight='bold')

ax1.set\_ylabel('Portfolio Value (Normalized)');ax1.grid(True,alpha=0.3)

ax2=fig.add\_subplot(gs[1,:2])

ax2.fill\_between(results.index,0,results['Drawdown'],where=(results['Drawdown']<0),alpha=0.5,color='red')

ax2.plot(results['Drawdown'],color='darkred',linewidth=1.5)

ax2.axhline(-5,color='orange',linestyle=':',alpha=0.5,label='5% Warning')

ax2.axhline(-10,color='red',linestyle=':',alpha=0.5,label='10% Limit')

ax2.set\_title(f'Drawdown (Max: {results["Drawdown"].min():.1f}%)',fontsize=12,fontweight='bold')

ax2.set\_ylabel('Drawdown (%)');ax2.legend(loc='lower right');ax2.grid(True,alpha=0.3)

ax3=fig.add\_subplot(gs[0,2])

returns=results['Net\_Return']\*100

ax3.hist(returns,bins=20,alpha=0.7,color='skyblue',edgecolor='black')

ax3.axvline(returns.mean(),color='red',linestyle='--',label=f'Mean: {returns.mean():.2f}%')

ax3.set\_title('Return Distribution',fontsize=12,fontweight='bold')

ax3.set\_xlabel('Monthly Return (%)');ax3.set\_ylabel('Frequency')

ax3.legend();ax3.grid(True,alpha=0.3)

ax4=fig.add\_subplot(gs[1,2])

ax4.plot(results['ES\_95']\*100,label='ES (95%)',color='red',linewidth=1.5)

ax4.plot(results['VaR\_95']\*100,label='VaR (95%)',color='orange',linewidth=1.5)

ax4.axhline(MAX\_LOSS\_PCT\*100,color='black',linestyle='--',label=f'Risk Limit ({MAX\_LOSS\_PCT\*100:.0f}%)')

ax4.set\_title('Risk Metrics',fontsize=12,fontweight='bold')

ax4.set\_ylabel('Expected Loss (%)');ax4.legend(loc='upper right');ax4.grid(True,alpha=0.3)

ax5=fig.add\_subplot(gs[2,:])

ax5.stackplot(results.index,results['SPY\_Weight']\*100,results['QQQ\_Weight']\*100,results['Cash\_Weight']\*100,

labels=['SPY','QQQ','Cash'],colors=['#4472C4','#ED7D31','#70AD47'],alpha=0.8)

ax5.set\_title('Portfolio Allocation Over Time',fontsize=12,fontweight='bold')

ax5.set\_ylabel('Weight (%)');ax5.set\_ylim(0,100);ax5.legend(loc='upper right');ax5.grid(True,alpha=0.3)

plt.suptitle('Risk Management Dashboard',fontsize=16,fontweight='bold')

plt.tight\_layout();plt.savefig('risk\_dashboard.png',dpi=120,bbox\_inches='tight');plt.show()

def performance\_summary(results):

ret=results['Net\_Return'];total\_ret=(results['Cumulative'].iloc[-1]-1)

n\_years=len(ret)/12;annual\_ret=(1+total\_ret)\*\*(1/n\_years)-1 if n\_years>0 else 0

vol=ret.std()\*np.sqrt(12);sharpe=(annual\_ret-RISK\_FREE\_RATE)/vol if vol>0 else 0

downside\_returns=ret[ret<0]

downside\_vol=downside\_returns.std()\*np.sqrt(12) if len(downside\_returns)>0 else vol

sortino=(annual\_ret-RISK\_FREE\_RATE)/downside\_vol if downside\_vol>0 else 0

max\_dd=results['Drawdown'].min();calmar=annual\_ret/abs(max\_dd) if max\_dd<0 else 0

print(f"\nPERFORMANCE SUMMARY")

print(f"Period: {results.index[0].strftime('%Y-%m')} to {results.index[-1].strftime('%Y-%m')}")

print(f"Total Return: {total\_ret:>8.1%}")

print(f"Annual Return: {annual\_ret:>8.1%}")

print(f"Volatility (Ann): {vol:>8.1%}")

print(f"Max Drawdown: {max\_dd:>8.1f}%")

print(f"Sharpe Ratio: {sharpe:>8.2f}")

print(f"Sortino Ratio: {sortino:>8.2f}")

print(f"Avg SPY Weight: {results['SPY\_Weight'].mean():>8.1%}")

print(f"Avg QQQ Weight: {results['QQQ\_Weight'].mean():>8.1%}")

print(f"Avg Cash Weight: {results['Cash\_Weight'].mean():>8.1%}")

def stress\_test(results):

print(f"\nSTRESS TEST")

mean\_ret=results['Net\_Return'].mean();vol=results['Net\_Return'].std()

scenarios={'Normal Market':{'drift':mean\_ret,'vol\_mult':1.0,'jump\_prob':0.02},

'High Volatility':{'drift':mean\_ret\*0.7,'vol\_mult':1.8,'jump\_prob':0.04},

'Bear Market':{'drift':-0.008,'vol\_mult':1.3,'jump\_prob':0.04},

'Market Crash':{'drift':-0.02,'vol\_mult':2.5,'jump\_prob':0.08},

'Bull Market':{'drift':mean\_ret\*2,'vol\_mult':0.8,'jump\_prob':0.01}}

print(f"{'Scenario':<18} {'1Y Return':<12} {'1Y Vol':<10} {'5% VaR':<10}")

for name,params in scenarios.items():

n\_sims=5000;annual\_returns=[]

for \_ in range(n\_sims):

monthly\_rets=[]

for \_ in range(12):

ret=np.random.normal(params['drift'],vol\*params['vol\_mult'])

if np.random.random()<params['jump\_prob']:

jump=np.random.choice([-0.08,-0.05,0.05],p=[0.2,0.6,0.2]);ret+=jump

monthly\_rets.append(ret)

annual\_ret=np.prod([1+r for r in monthly\_rets])-1

annual\_returns.append(annual\_ret)

annual\_returns=np.array(annual\_returns)

mean\_ret=annual\_returns.mean();annual\_vol=annual\_returns.std();var\_5=np.percentile(annual\_returns,5)

print(f"{name:<18} {mean\_ret:>10.1%} {annual\_vol:>8.1%} {-var\_5:>8.1%}")

# Main

if \_\_name\_\_ == "\_\_main\_\_":

print("\nMONTE CARLO PORTFOLIO RISK MANAGEMENT")

forecast\_data,vix\_data=load\_forecast\_data()

monte\_carlo\_visualization(forecast\_data,vix\_data)

results=run\_portfolio\_backtest(forecast\_data,vix\_data)

results.to\_csv('portfolio\_results.csv')

print("Results saved to portfolio\_results.csv")

performance\_summary(results)

stress\_test(results)

risk\_dashboard(results)

if len(results)>0:

latest=results.iloc[-1]

print(f"\nCURRENT SIGNALS")

print(f"Portfolio Val: ${latest['Portfolio\_Value']:.2f}")

print(f"SPY Weight: {latest['SPY\_Weight']:>8.1%}")

print(f"QQQ Weight: {latest['QQQ\_Weight']:>8.1%}")

print(f"Cash Weight: {latest['Cash\_Weight']:>8.1%}")

print(f"Risk (ES): {latest['ES\_95']:>8.2%}")

if latest['ES\_95']>MAX\_LOSS\_PCT:print("\nWARNING: Risk limit exceeded")

else:print("\nRisk within limits")

print("\nCOMPLETE")

**Listing 3: Walk-Forward Validation and Detailed Cross-Validation Analysis**

from datetime import datetime

def enhanced\_walk\_forward\_validation(tkr,F\_all,clean,n\_splits=5,verbose=True):

ret\_col=f"{tkr.lower()}\_ret";other="qqq" if tkr=="SPY" else "spy"

exclude\_patterns=[f"{other}\_price",f"{other}\_ret"]

base=clean.drop(columns=[c for c in clean.columns if any(pattern in c for pattern in exclude\_patterns)],errors="ignore")

if ret\_col not in base.columns:

if ret\_col in F\_all.columns:df=base.join(F\_all[[ret\_col]],how="inner").dropna()

else:return pd.DataFrame()

else:df=base.dropna()

df=create\_aligned\_targets(df,ret\_col,forecast\_horizon=1)

if len(df)<n\_splits\*24:return pd.DataFrame()

test\_size=max(12,int(len(df)/(n\_splits\*2)));tscv=TimeSeriesSplit(n\_splits=n\_splits,test\_size=test\_size)

fold\_results=[];all\_predictions=[]

for fold,(train\_idx,test\_idx) in enumerate(tscv.split(df),1):

train\_data=df.iloc[train\_idx];test\_data=df.iloc[test\_idx]

if len(train\_data)<20 or len(test\_data)<3:continue

train\_start=train\_data.index[0];train\_end=train\_data.index[-1]

test\_start=test\_data.index[0];test\_end=test\_data.index[-1]

sel\_cols=strict\_prune(train\_data.drop(columns="y"))

if len(sel\_cols.columns)==0:continue

Xtr,ytr=train\_data[sel\_cols.columns],train\_data["y"];Xte,yte=test\_data[sel\_cols.columns],test\_data["y"]

if Xtr.isna().any().any() or ytr.isna().any():

mask=~(Xtr.isna().any(axis=1)|ytr.isna());Xtr,ytr=Xtr[mask],ytr[mask]

if Xte.isna().any().any() or yte.isna().any():

mask=~(Xte.isna().any(axis=1)|yte.isna());Xte,yte=Xte[mask],yte[mask]

if len(Xtr)<10 or len(Xte)<2:continue

fitted\_models,scalers=build\_ensemble\_model(Xtr,ytr)

mu\_pred=ensemble\_predict(fitted\_models,scalers,Xte)

train\_preds=ensemble\_predict(fitted\_models,scalers,Xtr);train\_sigma=np.std(train\_preds)+1e-12

if not np.all(np.isfinite(mu\_pred)):mu\_pred=np.nan\_to\_num(mu\_pred,nan=0.0)

mu\_smooth=apply\_kalman\_smoothing(mu\_pred);weights=calculate\_weights\_continuous(mu\_smooth,train\_sigma=train\_sigma)

strat\_ret=weights\*yte.values;fold\_sharpe=0

if len(yte)>1 and strat\_ret.std()>1e-10:fold\_sharpe=(strat\_ret.mean()/strat\_ret.std())\*np.sqrt(12)

mse=np.mean((yte.values-mu\_smooth)\*\*2);mae=np.mean(np.abs(yte.values-mu\_smooth))

hit\_rate=(np.sign(mu\_smooth)==np.sign(yte.values)).mean()

y\_binary=(yte.values>0).astype(int)

if len(np.unique(y\_binary))>1:

from sklearn.metrics import roc\_auc\_score

roc\_auc=roc\_auc\_score(y\_binary,mu\_smooth)

else:roc\_auc=0.5

cum\_return=np.prod(1+strat\_ret)-1

fold\_result={'Fold':fold,'Train\_Start':train\_start.strftime('%Y-%m-%d'),'Train\_End':train\_end.strftime('%Y-%m-%d'),

'Test\_Start':test\_start.strftime('%Y-%m-%d'),'Test\_End':test\_end.strftime('%Y-%m-%d'),

'Train\_Months':len(train\_data),'Test\_Months':len(test\_data),'Sharpe\_Ratio':fold\_sharpe,

'Total\_Return':cum\_return\*100,'Annualized\_Return':((1+cum\_return)\*\*(12/len(test\_data))-1)\*100,

'Hit\_Rate':hit\_rate\*100,'ROC\_AUC':roc\_auc,'MSE':mse,'MAE':mae,'Avg\_Position':np.mean(np.abs(weights)),

'Max\_Position':np.max(np.abs(weights)),'N\_Trades':np.sum(np.abs(np.diff(weights))>0.01)}

fold\_results.append(fold\_result)

for i,idx in enumerate(yte.index):

all\_predictions.append({'Date':idx,'Fold':fold,'Actual':yte.iloc[i],'Predicted':mu\_smooth[i],

'Weight':weights[i],'Return':strat\_ret[i]})

if verbose:print(f"\nFold {fold}: {test\_start.strftime('%Y-%m')} to {test\_end.strftime('%Y-%m')}\n Sharpe: {fold\_sharpe:.3f}, Return: {cum\_return\*100:.1f}%, Hit Rate: {hit\_rate\*100:.1f}%")

fold\_df=pd.DataFrame(fold\_results);pred\_df=pd.DataFrame(all\_predictions)

return fold\_df,pred\_df

def generate\_cv\_report(tkr,fold\_df,pred\_df,save\_files=True):

print(f"\n{tkr} DETAILED CROSS-VALIDATION RESULTS")

print("\nFOLD-BY-FOLD PERFORMANCE:")

display\_cols=['Fold','Test\_Start','Test\_End','Test\_Months','Sharpe\_Ratio','Total\_Return','Hit\_Rate','ROC\_AUC']

print(fold\_df[display\_cols].to\_string(index=False))

print("\nSUMMARY STATISTICS:")

summary\_stats={'Mean\_Sharpe':fold\_df['Sharpe\_Ratio'].mean(),'Std\_Sharpe':fold\_df['Sharpe\_Ratio'].std(),

'Min\_Sharpe':fold\_df['Sharpe\_Ratio'].min(),'Max\_Sharpe':fold\_df['Sharpe\_Ratio'].max(),

'Mean\_Return\_%':fold\_df['Total\_Return'].mean(),'Mean\_Hit\_Rate\_%':fold\_df['Hit\_Rate'].mean(),

'Mean\_ROC\_AUC':fold\_df['ROC\_AUC'].mean(),'Positive\_Sharpe\_Folds':(fold\_df['Sharpe\_Ratio']>0).sum(),

'Total\_Folds':len(fold\_df)}

for key,value in summary\_stats.items():

if 'Sharpe' in key or 'ROC' in key:print(f"{key:20s}: {value:8.3f}")

else:print(f"{key:20s}: {value:8.1f}")

best\_fold=fold\_df.loc[fold\_df['Sharpe\_Ratio'].idxmax()];worst\_fold=fold\_df.loc[fold\_df['Sharpe\_Ratio'].idxmin()]

print("\nNOTABLE PERIODS:")

print(f"Best Period: Fold {int(best\_fold['Fold'])} ({best\_fold['Test\_Start']} to {best\_fold['Test\_End']})")

print(f" Sharpe: {best\_fold['Sharpe\_Ratio']:.3f}, Return: {best\_fold['Total\_Return']:.1f}%")

print(f"Worst Period: Fold {int(worst\_fold['Fold'])} ({worst\_fold['Test\_Start']} to {worst\_fold['Test\_End']})")

print(f" Sharpe: {worst\_fold['Sharpe\_Ratio']:.3f}, Return: {worst\_fold['Total\_Return']:.1f}%")

fig,axes=plt.subplots(2,3,figsize=(15,10));axes[1,2].remove()

fig.suptitle(f'{tkr} Cross-Validation Analysis',fontsize=16,fontweight='bold')

ax1=axes[0,0];colors=['green' if x>0 else 'red' for x in fold\_df['Sharpe\_Ratio']]

ax1.bar(fold\_df['Fold'],fold\_df['Sharpe\_Ratio'],color=colors,alpha=0.7)

ax1.axhline(y=0,color='black',linestyle='-',linewidth=0.5)

ax1.axhline(y=fold\_df['Sharpe\_Ratio'].mean(),color='blue',linestyle='--',label=f'Mean: {fold\_df["Sharpe\_Ratio"].mean():.3f}')

ax1.set\_xlabel('Fold');ax1.set\_ylabel('Sharpe Ratio');ax1.set\_title('Sharpe Ratio by Fold')

ax1.legend();ax1.grid(True,alpha=0.3)

ax2=axes[0,1];colors=['green' if x>0 else 'red' for x in fold\_df['Total\_Return']]

ax2.bar(fold\_df['Fold'],fold\_df['Total\_Return'],color=colors,alpha=0.7)

ax2.axhline(y=0,color='black',linestyle='-',linewidth=0.5)

ax2.set\_xlabel('Fold');ax2.set\_ylabel('Total Return (%)');ax2.set\_title('Total Return by Fold')

ax2.grid(True,alpha=0.3)

ax3=axes[0,2];ax3.bar(fold\_df['Fold'],fold\_df['Hit\_Rate'],color='blue',alpha=0.7)

ax3.axhline(y=50,color='red',linestyle='--',label='50% (Random)')

ax3.set\_xlabel('Fold');ax3.set\_ylabel('Hit Rate (%)');ax3.set\_title('Hit Rate by Fold')

ax3.legend();ax3.grid(True,alpha=0.3)

ax4=axes[1,0]

if not pred\_df.empty:

pred\_df\_sorted=pred\_df.sort\_values('Date')

cum\_returns=(1+pred\_df\_sorted.groupby('Date')['Return'].mean()).cumprod()

ax4.plot(cum\_returns.index,cum\_returns.values,linewidth=2)

ax4.set\_xlabel('Date');ax4.set\_ylabel('Cumulative Return')

ax4.set\_title('Cumulative Performance Across All Folds');ax4.grid(True,alpha=0.3)

plt.setp(ax4.xaxis.get\_majorticklabels(),rotation=45,ha='right')

ax5=axes[1,1];fold\_dates=pd.to\_datetime(fold\_df['Test\_Start'])

ax5.plot(fold\_dates,fold\_df['Sharpe\_Ratio'],'o-',linewidth=2,markersize=8)

ax5.axhline(y=0,color='red',linestyle='--',alpha=0.5)

ax5.set\_xlabel('Test Period Start');ax5.set\_ylabel('Sharpe Ratio')

ax5.set\_title('Sharpe Ratio Evolution Over Time');ax5.grid(True,alpha=0.3)

plt.setp(ax5.xaxis.get\_majorticklabels(),rotation=45,ha='right')

if save\_files:

fig\_filename=f'{tkr.lower()}\_cv\_detailed\_analysis.png'

plt.savefig(fig\_filename,dpi=150,bbox\_inches='tight')

print(f"\nVisualization saved to: {fig\_filename}")

fold\_filename=f'{tkr.lower()}\_cv\_fold\_results.csv'

fold\_df.to\_csv(fold\_filename,index=False)

print(f"Fold results saved to: {fold\_filename}")

pred\_filename=f'{tkr.lower()}\_cv\_all\_predictions.csv'

pred\_df.to\_csv(pred\_filename,index=False)

print(f"All predictions saved to: {pred\_filename}")

summary\_filename=f'{tkr.lower()}\_cv\_summary.txt'

with open(summary\_filename,'w') as f:

f.write(f"{tkr} Cross-Validation Summary\n\n")

f.write("Fold-by-Fold Results:\n");f.write(fold\_df.to\_string(index=False))

f.write("\n\nSummary Statistics:\n")

for key,value in summary\_stats.items():f.write(f"{key:30s}: {value:10.3f}\n")

print(f"Summary saved to: {summary\_filename}")

plt.show()

return summary\_stats

def run\_detailed\_cv\_analysis():

global F\_all,clean

detailed\_results={}

for ticker in ['SPY','QQQ']:

print(f"\nRunning Detailed CV Analysis for {ticker}")

fold\_df,pred\_df=enhanced\_walk\_forward\_validation(ticker,F\_all,clean,n\_splits=5,verbose=True)

if not fold\_df.empty:

summary=generate\_cv\_report(ticker,fold\_df,pred\_df,save\_files=True)

detailed\_results[ticker]={'fold\_results':fold\_df,'predictions':pred\_df,'summary':summary}

else:print(f"No valid results for {ticker}")

if len(detailed\_results)==2:

print("\nCOMPARATIVE ANALYSIS")

comparison\_df=pd.DataFrame({'Metric':['Mean Sharpe','Std Sharpe','Best Sharpe','Worst Sharpe','Success Rate'],

'SPY':[detailed\_results['SPY']['summary']['Mean\_Sharpe'],detailed\_results['SPY']['summary']['Std\_Sharpe'],

detailed\_results['SPY']['summary']['Max\_Sharpe'],detailed\_results['SPY']['summary']['Min\_Sharpe'],

detailed\_results['SPY']['summary']['Positive\_Sharpe\_Folds']/detailed\_results['SPY']['summary']['Total\_Folds']],

'QQQ':[detailed\_results['QQQ']['summary']['Mean\_Sharpe'],detailed\_results['QQQ']['summary']['Std\_Sharpe'],

detailed\_results['QQQ']['summary']['Max\_Sharpe'],detailed\_results['QQQ']['summary']['Min\_Sharpe'],

detailed\_results['QQQ']['summary']['Positive\_Sharpe\_Folds']/detailed\_results['QQQ']['summary']['Total\_Folds']]})

print(comparison\_df.to\_string(index=False))

comparison\_df.to\_csv('cv\_comparison\_spy\_qqq.csv',index=False)

print("\nComparison saved to: cv\_comparison\_spy\_qqq.csv")

return detailed\_results

if \_\_name\_\_=="\_\_main\_\_":

cv\_detailed\_results=run\_detailed\_cv\_analysis()

print("\nDETAILED CV ANALYSIS COMPLETE")

print("\nYou can now reference the following files in your paper:")

print(" - spy\_cv\_fold\_results.csv (specific period Sharpe ratios)")

print(" - qqq\_cv\_fold\_results.csv (specific period Sharpe ratios)")

print(" - spy\_cv\_detailed\_analysis.png (visualizations)")

print(" - qqq\_cv\_detailed\_analysis.png (visualizations)")

print(" - cv\_comparison\_spy\_qqq.csv (comparative metrics)")

**Listing 4: Position Sizing and Weight Distribution Visualization**

plt.style.use('seaborn-v0\_8-whitegrid')

sns.set\_palette("husl")

plt.rcParams['figure.dpi']=100;plt.rcParams['savefig.dpi']=300

plt.rcParams['font.size']=10;plt.rcParams['font.family']='serif'

plt.rcParams['font.serif']=['Times New Roman']

def load\_actual\_results():

spy\_forecast=pd.read\_csv('spy\_forecast\_detail.csv',index\_col=0,parse\_dates=True)

qqq\_forecast=pd.read\_csv('qqq\_forecast\_detail.csv',index\_col=0,parse\_dates=True)

metrics=pd.read\_csv('05\_metrics\_reg.csv',index\_col=0)

portfolio\_results=pd.read\_csv('portfolio\_results.csv',index\_col=0,parse\_dates=True)

spy\_features=pd.read\_csv('spy\_selected\_features.csv');qqq\_features=pd.read\_csv('qqq\_selected\_features.csv')

return {'spy\_forecast':spy\_forecast,'qqq\_forecast':qqq\_forecast,'metrics':metrics,

'portfolio':portfolio\_results,'spy\_features':spy\_features,'qqq\_features':qqq\_features}

def create\_figure\_3\_weight\_distribution(data):

spy\_data=data['spy\_forecast'];qqq\_data=data['qqq\_forecast']

fig,axes=plt.subplots(2,3,figsize=(16,10));axes[1,2].axis("off")

ax=axes[0,0];weights=spy\_data['weight'].values;weights\_nonzero=weights[weights!=0]

ax.hist(weights\_nonzero,bins=30,edgecolor='black',alpha=0.7,color='#2E86AB')

ax.axvline(np.mean(weights\_nonzero),color='red',linestyle='--',lw=2,label=f'Mean: {np.mean(weights\_nonzero):.3f}')

ax.axvline(0,color='black',linestyle='-',alpha=0.5)

ax.set\_xlabel('Portfolio Weight',fontsize=11);ax.set\_ylabel('Frequency',fontsize=11)

ax.set\_title('SPY: Weight Distribution',fontsize=12,fontweight='bold');ax.legend();ax.grid(True,alpha=0.3)

ax=axes[0,1];weights=qqq\_data['weight'].values;weights\_nonzero=weights[weights!=0]

ax.hist(weights\_nonzero,bins=30,edgecolor='black',alpha=0.7,color='#A23B72')

ax.axvline(np.mean(weights\_nonzero),color='red',linestyle='--',lw=2,label=f'Mean: {np.mean(weights\_nonzero):.3f}')

ax.axvline(0,color='black',linestyle='-',alpha=0.5)

ax.set\_xlabel('Portfolio Weight',fontsize=11);ax.set\_ylabel('Frequency',fontsize=11)

ax.set\_title('QQQ: Weight Distribution',fontsize=12,fontweight='bold');ax.legend();ax.grid(True,alpha=0.3)

ax=axes[1,0];predictions=spy\_data['mu\_kalman'].values;weights=spy\_data['weight'].values

ax.scatter(predictions,weights,alpha=0.6,s=30,color='#2E86AB')

ax.set\_xlabel('Prediction Value',fontsize=11);ax.set\_ylabel('Position Weight',fontsize=11)

ax.set\_title('SPY: Weight vs Prediction',fontsize=12,fontweight='bold');ax.grid(True,alpha=0.3)

ax.axhline(0,color='black',linestyle='-',alpha=0.5);ax.axvline(0,color='black',linestyle='-',alpha=0.5)

ax=axes[1,1];predictions=qqq\_data['mu\_kalman'].values;weights=qqq\_data['weight'].values

ax.scatter(predictions,weights,alpha=0.6,s=30,color='#A23B72')

ax.set\_xlabel('Prediction Value',fontsize=11);ax.set\_ylabel('Position Weight',fontsize=11)

ax.set\_title('QQQ: Weight vs Prediction',fontsize=12,fontweight='bold');ax.grid(True,alpha=0.3)

ax.axhline(0,color='black',linestyle='-',alpha=0.5);ax.axvline(0,color='black',linestyle='-',alpha=0.5)

ax=axes[0,2];spy\_weights=spy\_data['weight'].abs().values;qqq\_weights=qqq\_data['weight'].abs().values

categories=['0-10%','10-20%','20-30%','>30%']

spy\_dist=[np.sum(spy\_weights<=0.1)/len(spy\_weights)\*100,

np.sum((spy\_weights>0.1)&(spy\_weights<=0.2))/len(spy\_weights)\*100,

np.sum((spy\_weights>0.2)&(spy\_weights<=0.3))/len(spy\_weights)\*100,

np.sum(spy\_weights>0.3)/len(spy\_weights)\*100]

qqq\_dist=[np.sum(qqq\_weights<=0.1)/len(qqq\_weights)\*100,

np.sum((qqq\_weights>0.1)&(qqq\_weights<=0.2))/len(qqq\_weights)\*100,

np.sum((qqq\_weights>0.2)&(qqq\_weights<=0.3))/len(qqq\_weights)\*100,

np.sum(qqq\_weights>0.3)/len(qqq\_weights)\*100]

x=np.arange(len(categories));width=0.35

ax.bar(x-width/2,spy\_dist,width,label='SPY',color='#2E86AB',alpha=0.8)

ax.bar(x+width/2,qqq\_dist,width,label='QQQ',color='#A23B72',alpha=0.8)

ax.set\_xlabel('Weight Range',fontsize=11);ax.set\_ylabel('Percentage of Positions',fontsize=11)

ax.set\_title('Position Size Distribution by Category',fontsize=12,fontweight='bold')

ax.set\_xticks(x);ax.set\_xticklabels(categories);ax.legend();ax.grid(True,alpha=0.3,axis='y')

plt.suptitle('Position Sizing Analysis',fontsize=14,fontweight='bold')

plt.tight\_layout();plt.savefig('figure3\_weight\_distribution.png',dpi=300,bbox\_inches='tight');plt.show()

def generate\_all\_research\_outputs():

data=load\_actual\_results()

create\_figure\_3\_weight\_distribution(data)

return data

if \_\_name\_\_=="\_\_main\_\_":

results=generate\_all\_research\_outputs()

**Listing 5: Heatmap**

def generate\_heatmaps\_for\_ticker(tkr,F\_all,clean,TOP\_K=35,MAX\_CORR=0.95):

ret\_col=f"{tkr.lower()}\_ret";other="qqq" if tkr=="SPY" else "spy"

exclude\_cols=[c for c in clean.columns if other.lower() in c.lower() or c==ret\_col]

base=clean.drop(columns=exclude\_cols,errors='ignore')

n\_display=min(15,len(base.columns));top\_var\_cols=base.var().nlargest(n\_display).index

plt.figure(figsize=(14,12))

before\_df=base[top\_var\_cols].dropna();corr\_before=before\_df.corr()

ax=sns.heatmap(corr\_before,vmin=-1,vmax=1,cmap='RdBu\_r',center=0,square=True,linewidths=0.5,

linecolor='gray',cbar\_kws={'label':'Correlation','shrink':0.8},annot=True,fmt='.2f',annot\_kws={'size':8})

plt.title(f"{tkr} Features BEFORE Pruning\n(Top {n\_display} by variance from {len(base.columns)} total)",

fontsize=14,fontweight='bold')

plt.xlabel('');plt.ylabel('');plt.xticks(rotation=45,ha='right',fontsize=9);plt.yticks(rotation=0,fontsize=9)

plt.tight\_layout()

before\_filename=f'{tkr.lower()}\_before\_prune\_heatmap.png'

plt.savefig(before\_filename,dpi=150,bbox\_inches='tight');plt.show()

returns\_series=F\_all[[ret\_col]].copy();returns\_series.columns=['target\_return']

df\_for\_prune=base.join(returns\_series,how='inner');df\_for\_prune['y']=df\_for\_prune['target\_return'].shift(-1)

df\_for\_prune=df\_for\_prune.dropna(subset=['y'])

if len(df\_for\_prune)<30:return pd.DataFrame()

cut=int(len(df\_for\_prune)\*0.75);train\_data=df\_for\_prune.iloc[:cut]

features\_to\_prune=train\_data.drop(columns=['y','target\_return'],errors='ignore')

sel\_cols=strict\_prune(features\_to\_prune,max\_corr=MAX\_CORR,k=TOP\_K)

if len(sel\_cols.columns)==0:return pd.DataFrame()

n\_display\_after=min(15,len(sel\_cols.columns))

if len(sel\_cols.columns)>n\_display\_after:

top\_pruned\_cols=sel\_cols.var().nlargest(n\_display\_after).index

after\_df=sel\_cols[top\_pruned\_cols].dropna()

display\_text=f"(Top {n\_display\_after} by variance from {len(sel\_cols.columns)} selected)"

else:

after\_df=sel\_cols.dropna();display\_text=f"({len(sel\_cols.columns)} selected features)"

plt.figure(figsize=(14,12));corr\_after=after\_df.corr()

ax=sns.heatmap(corr\_after,vmin=-1,vmax=1,cmap='RdBu\_r',center=0,square=True,linewidths=0.5,

linecolor='gray',cbar\_kws={'label':'Correlation','shrink':0.8},annot=True,fmt='.2f',annot\_kws={'size':8})

plt.title(f"{tkr} Features AFTER Pruning\n{display\_text}",fontsize=14,fontweight='bold')

plt.xlabel('');plt.ylabel('');plt.xticks(rotation=45,ha='right',fontsize=9);plt.yticks(rotation=0,fontsize=9)

plt.tight\_layout()

after\_filename=f'{tkr.lower()}\_after\_prune\_heatmap.png'

plt.savefig(after\_filename,dpi=150,bbox\_inches='tight');plt.show()

feature\_file=f'{tkr.lower()}\_selected\_features\_list.csv'

pd.DataFrame({'feature':sel\_cols.columns}).to\_csv(feature\_file,index=False)

return sel\_cols

def main():

F\_all=pd.read\_csv("feature\_matrix\_full.csv",index\_col=0,parse\_dates=True)

clean=pd.read\_csv("02\_clean\_preprune.csv",index\_col=0,parse\_dates=True)

TOP\_K=35;MAX\_CORR=0.95;results={}

for ticker in ['SPY','QQQ']:

selected\_features=generate\_heatmaps\_for\_ticker(ticker,F\_all,clean,TOP\_K=TOP\_K,MAX\_CORR=MAX\_CORR)

results[ticker]=selected\_features

return results

if \_\_name\_\_=="\_\_main\_\_":

selected\_features\_dict=main()