

Bull/Bear Regime-Aware Long/Short Equity Strategy

A monthly rebalancing long/short equity strategy built on the JKP Factors CTF dataset. The core idea is that factor returns behave differently in bull and bear markets, so instead of one model for everything, we maintain two separate Ridge regression models — one trained exclusively on bull-market months, one on bear-market months — and switch between them based on where the market currently stands.

Table of Contents

1. [Data](#)
 2. [Pipeline Overview](#)
 3. [Step 1: Feature Preprocessing](#)
 4. [Step 2: Feature Modes](#)
 5. [Step 3: Market Return Estimation](#)
 6. [Step 4: Bull/Bear Regime Detection](#)
 7. [Step 5: Ridge Regression Models](#)
 8. [Step 6: Portfolio Construction](#)
 9. [Step 7: Regime Blending](#)
 10. [Step 8: Volatility Scaling](#)
 11. [Step 9: Turnover Smoothing](#)
 12. [Output Format](#)
 13. [Key Parameters](#)
 14. [How to Run](#)
 15. [Design Decisions and Findings](#)
-

Data

- `ctff_chars.parquet` — 1.37 million rows, ~410 columns. Each row is one stock at one end-of-month date.
- `ctff_features.csv` — list of 402 feature names drawn from the JKP factor library.

Key columns:

Column	Description
<code>eom</code>	End-of-month date (the observation date)
<code>id</code>	Stock identifier
<code>market_equity</code>	Market capitalization, used for value-weighting
<code>ret_exc_lead1m</code>	Next month's excess return — already lead-shifted , so the value at <code>eom=t</code> is the return earned from <code>t</code> to <code>t+1</code>

Because `ret_exc_lead1m` is pre-shifted, we can use it directly as a training label without any further manipulation — the value at date `t` is the future return the model is trying to predict.

Pipeline Overview

Every step uses only information available at or before the prediction date. There is no look-ahead.

Raw data

- per-month preprocessing (fill, clip, z-score)
- optionally collapse to 6 economic composites
- compute value-weighted market return series
- label each month bull or bear (trailing 12m return + hysteresis)
- accumulate separate Ridge models for each regime (expanding window)
- refit models once per year, picking regularization via time-blocked CV
- score stocks with the current regime's model
- convert scores to long/short weights
- blend portfolio targets based on how strongly bull/bear the market looks
- scale exposure down in high-volatility periods
- blend with last month's weights to reduce turnover
- write `weights_output.csv`

The first 10 years of data (120 months) are used as a burn-in period — no predictions are made until

we have enough history to train a meaningful model.

Step 1: Feature Preprocessing

Done separately for each month, using only that month's cross-section of stocks. Three operations in sequence:

1. Median imputation

Missing values for feature j in month t are filled with the cross-sectional median of that feature across all stocks in that month:

```
 $x_{i,j} \leftarrow \text{median}(\{ x_{k,j} : x_{k,j} \text{ is not NaN } \}) \quad \text{for any } i \text{ where } x_{i,j} \text{ is NaN}$ 
```

If a feature is entirely missing for that month, it's set to 0.

2. Outlier clipping

Values are clipped to the 1st–99th percentile range of that month's cross-section:

```
 $x_{i,j} \leftarrow \text{clip}(x_{i,j}, p1, p99)$ 
```

This prevents a handful of extreme stocks from dominating the regression.

3. Z-scoring

Each feature is standardized to zero mean and unit variance within the month:

```
 $x_{i,j} \leftarrow (x_{i,j} - \text{mean}_j) / \text{std}_j$ 
```

If the standard deviation is essentially zero (a constant feature), the column is zeroed out entirely.

The result is that every feature, in every month, is on the same scale with no missing values — which is exactly what Ridge regression needs.

Step 2: Feature Modes

The strategy can run in two modes:

Mode A: Raw Feature Subset (default, USE_COMPOSITES=False, USE_FEATURE_SUBSET=True)

Uses a curated set of 40 features selected for their cross-sectional predictive power. These are split by the direction of their signal:

Long signals (higher value → predict better returns):

Feature	What it captures
<code>cop_mev</code> , <code>cop_at</code> , <code>cop_me</code>	Cash operating profitability
<code>ocf_at</code> , <code>ocf_mev</code> , <code>ocf_me</code> , <code>ocf_be</code>	Operating cash flow scaled various ways
<code>ebitda_mev</code> , <code>ebitda_me</code>	EBITDA-based value
<code>mispricing_mgmt</code> , <code>mispricing_perf</code>	Composite mispricing scores
<code>ope_me</code>	Operating profits to equity
<code>qmj_prof</code>	Quality minus junk (profitability component)
<code>ret_12_1</code>	12-month momentum (skipping most recent month)
<code>resff3_12_1</code>	Residual momentum after FF3
<code>eqnpo_me</code> , <code>eqnpo_at</code>	Equity net payout
<code>fcf_be</code> , <code>fcf_ppen</code> , <code>fcf_sale</code>	Free cash flow ratios

Short signals (higher value → predict worse returns):

Feature	What it captures
<code>inv_gr1</code> , <code>at_gr1</code> , <code>ppeinv_gr1a</code>	Asset/investment growth (investment anomaly)
<code>opex_gr1</code> , <code>oa_gr1a</code> , <code>noa_gr1a</code>	Operating expense and accrual growth
<code>ivol_ff3_21d</code> , <code>ivol_capm_21d</code> , <code>ivol_capm_252d</code> , <code>ivol_hxz4_21d</code>	Idiosyncratic volatility
<code>rvol_21d</code> , <code>rvol_252d</code> , <code>rmax1_21d</code> , <code>rmax5_21d</code>	Realized volatility and max return
<code>ret_1_0</code>	Short-term reversal

Feature

What it captures

eqnetis_at, netis_at, fincf_at, fincf_mev

Net equity/financial issuance

seas_2_5na

Seasoning (return seasonality)

Mode B: 6 Economic Composites (USE_COMPOSITES=True)

Collapses the full 402 features into 6 interpretable signals. Each composite averages its constituent z-scores, with sign applied:

```
composite_i = nanmean( [+z_j for j in pos_members] + [-z_k for k in neg_members] )
```

Composite

Positive members

Negative members

VALUE

be_me, sale_me, cash_mev, be_mev,
sale_bev

—

QUALITY

gp_at, ni_at, gp_sale, ebit_sale,
ebitda_sale

earnings_variability

MOMENTUM

ret_12_1, ret_6_1, ret_36_12

—

INVESTMEN

T

—

at_gr1, capx_at, sale_gr1

DISTRESS

o_score

z_score, f_score, debt_at, debt_be

LOWRISK

—

beta_60m, ivol_capm_60m, rvol_252d,
ivol_capm_252d

Using composites is much faster (P=6 vs P=40) but gives up some signal by collapsing features together.

Step 3: Market Return Estimation

Each month, we compute a value-weighted market return proxy:

$$R_m[t] = \frac{\sum_i (ME[i,t] \times \text{ret_exc_lead1m}[i,t])}{\sum_i (ME[i,t])}$$

where the sum is over all stocks with valid market cap and return data at date t . If market cap data is missing entirely for a month, it falls back to an equal-weighted average.

Since $\text{ret_exc_lead1m}[i,t]$ is the return from t to $t+1$, $R_m[t]$ represents the market's performance over the next month. This means that when we're at date t , $R_m[t-1]$ is the most recent *realized* market return — and $R_m[t-12]$ through $R_m[t-1]$ form the trailing 12-month history.

Step 4: Bull/Bear Regime Detection

Raw Signal

For each month t , we look at the past 12 months of market returns:

$$\text{cum12}[t] = \sum_{k=t-12}^{t-1} R_m[k]$$

This uses only returns that have already been realized — no look-ahead.

```
raw_signal[t] = "bull"   if cum12[t] > 0
               = "bear"   otherwise
               = None     if fewer than 12 months of history exist
```

Hysteresis Filter

Markets are noisy, and a raw trailing-return signal can flip back and forth rapidly around the zero line. To avoid over-trading on those signals, we require that the market send the same regime signal for **2 consecutive months** before we officially switch:

```
if raw_signal[t] == current_regime:
    reset pending counter

else:
    if raw_signal[t] == pending_signal:
        pending_count += 1
    else:
        pending_signal = raw_signal[t]
        pending_count = 1
```

```
if pending_count >= 2:
    current_regime = raw_signal[t]
    pending_count = 0
```

The result is a series where regime labels are "sticky" — they don't change unless the market has been unambiguously signaling the new regime for at least two months in a row.

Why This Matters

Factor exposures are not symmetric across market environments: - **Value, quality, and cash flow factors** tend to earn stronger risk-adjusted returns in bear markets, when investors flee to fundamentals. - **Momentum** often reverses sharply at market bottoms (momentum crashes). - **Volatility factors** misbehave in high-vol regimes.

Training separate models on each regime lets the Ridge regression learn these different factor relationships rather than averaging over all environments together.

Step 5: Ridge Regression Models

Model Structure

We maintain two models — one for bull months, one for bear months. Each is a standard Ridge regression:

$$\hat{y}[i,t] = \beta \cdot x[i,t]$$

where $x[i,t]$ is the vector of preprocessed feature values for stock i at date t , and $\hat{y}[i,t]$ is the predicted excess return.

Training Data

Only months with the matching regime label contribute to each model. So if we're at date t and it's a bull month, the bull model has been trained on all past bull months; the bear model on all past bear months.

Closed-Form Solution

Ridge coefficients are solved analytically using the normal equations:

$$\beta = (X'X + \alpha \cdot I)^{-1} \cdot X'y$$

This is computed via `np.linalg.solve(X'X +  $\alpha$ ·I, X'y)` rather than explicit matrix inversion, which is more numerically stable. No sklearn required — this runs on pure numpy.

Incremental Accumulation

Rather than restacking the full training matrix every time we refit, we maintain running sums:

```
XTX_total[regime] += X[tau]' · X[tau]    for each new training month tau
Xty_total[regime] += X[tau]' · y[tau]
```

When it's time to refit, we solve $(XTX_total + \alpha \cdot I) \beta = Xty_total$ directly. This makes refitting essentially free — we only need to do $O(P^3)$ work for the solve, not $O(N \cdot P^2)$ to rebuild the full matrix.

Regularization via Time-Blocked CV

Each time we refit (once per year), we pick the regularization strength α from the grid `[0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]` using 5-fold time-blocked cross-validation:

1. Split the training months into 5 contiguous blocks (in chronological order).
2. For fold k , train on all months **before** block k , validate on block k .
3. Score each fold with Spearman rank correlation (IC) between predicted and actual returns.
4. Pick the α with the highest mean IC across folds.

The fold-subtraction trick carries over to CV: because we've already precomputed per-month XTX and Xty , we can build any training subset's XTX by subtracting the excluded months from the running total — no re-stacking needed.

Spearman IC is used instead of R^2 or MSE because we care about rank ordering of stocks, not the level of return predictions. Even a weak positive IC (0.02–0.05) translates to meaningful long/short portfolio returns when applied consistently across hundreds of stocks each month.

Expanding Window

Models are retrained every 12 months on an ever-growing dataset. At month $t=130$ (10 years of burn-in + 10 months), the bull model has seen every bull month since the start of the dataset; same for bear. We never discard old data. This is appropriate because factor relationships are relatively stable over decades, and more data generally means better-calibrated coefficients.

Step 6: Portfolio Construction

Scoring

At each prediction date t , we run the current regime's model to score every stock:

```
score[i] = x[i,t] · β_regime
```

We use the bear model as a fallback if the current regime's model hasn't been trained yet (only relevant in the very early prediction period).

Rank-Based Weighting

Rather than using raw prediction scores directly, we rank-transform them. This makes the portfolio robust to prediction outliers and ensures that the weight distribution depends only on the ordering of stocks, not the magnitude of predictions:

```
rank[i]      = rank of score[i] among all stocks at date t    (0 to N-1)
mapped[i]    = rank[i] / (N - 1) - 0.5                        (maps to [-0.5, +0.5])
signal[i]    = (mapped[i] - mean(mapped)) / std(mapped)       (unit variance)
```

Stocks with the highest predicted returns get large positive weights (long positions); stocks with the lowest predicted returns get large negative weights (short positions).

Rescaling to Targets

The raw signal is then rescaled to hit the regime's gross and net exposure targets. We iterate three times through:

1. **Scale gross:** multiply all weights by `gross_target / sum(|w|)`
2. **Shift net:** subtract `(sum(w) - net_target) / N` from every weight uniformly
3. **Clip:** cap each weight at `±MAX_ABS_WEIGHT = 0.03`

Three iterations converge very close to both targets (clipping in step 3 perturbs the gross and net, which is why we repeat).

Bull vs Bear Portfolio Targets

Parameter	Bull	Bear
Gross exposure	1.20	0.80
Net exposure	+0.4	-0.1
	0	0
Per-stock cap	0.03	0.03

In bull markets, we run a larger book (1.2x gross) with a meaningful long bias (0.4 net). The thinking: factor-based stock selection works well in bull markets, and we want to capture upside from both the longs and the factor premium.

In bear markets, we shrink the book (0.8x gross) and flip to a slight short bias (-0.1 net). Factor signals — especially quality, low-risk, and cash flow — still identify the better stocks to own, but we're less confident in the overall market direction and want to reduce risk. The slight short bias reflects that the average stock tends to underperform in sustained bear markets.

Step 7: Regime Blending

A hard switch between bull and bear at the exact moment the regime label flips creates discontinuities in the portfolio. Instead, we smoothly interpolate between the two sets of targets based on the trailing 12-month cumulative return:

```
strength = clip(cum12 / BLEND_BAND, -1, +1)      # -1 = deep bear, +1 = deep bull
alpha    = (strength + 1) / 2                    # 0 = full bear, 1 = full bull

gross_t   = gross_bear + alpha * (gross_bull - gross_bear)
net_t     = net_bear   + alpha * (net_bull   - net_bear)
```

`BLEND_BAND = 0.04` means that when the trailing 12-month return is between -4% and +4%, we're blending between regimes. Outside that band, we're fully in one regime.

Early Bull Bonus

When the market is just turning up ($0 < \text{strength} < 1$), we add a tapering gross exposure bonus:

```
gross_t += EARLY_BULL_GROSS_BONUS * (1 - strength)
        = 0.2 * (1 - strength)
```

This extra exposure tapers to zero as the bull strengthens, capturing the tendency for high-beta stocks to surge strongly in the early months of a new bull market.

Step 8: Volatility Scaling

We scale the entire portfolio's exposure based on how volatile the market currently is, targeting a stable level of market risk across time:

```
sigma_m[t] = std(Rm[t-12 : t])          # trailing 12-month market vol
sigma_star = median(sigma_m) over all prediction-period dates # long-run target vol

scale_t = min(1.0, sigma_star / sigma_m[t])
w_scaled = w_raw * scale_t
```

When the market is unusually volatile (e.g., during a crash), `sigma_m[t] > sigma_star` and we shrink the book proportionally. We cap the scale at 1.0 so we never lever up just because vol is low — only de-lever when vol is high.

The `min(1.0, ...)` constraint is deliberate: we don't want to blindly increase exposure when markets are calm, because calm markets can turn volatile quickly. We only reduce when it's clearly dangerous.

Step 9: Turnover Smoothing

Each month, the new raw weights are blended with the previous month's final weights:

```
w_blended[i,t] = (1 - alpha) * w_prev[i,t-1] + alpha * w_scaled[i,t]
```

with `alpha = SMOOTHING_ALPHA = 0.5`. Stocks that were in the portfolio last month but not this month get a weight of 0.0 in the previous-weight lookup (they're being phased out over multiple months rather than immediately dropped).

After blending, the gross/net targets are re-imposed via the same iterative rescaling procedure, since

blending perturbs them.

This reduces portfolio churn substantially. Instead of completely re-positioning every month (which has trading costs), we blend 50/50 between old and new — effectively spreading the turnover over two months.

Output Format

The strategy writes `weights_output.csv` with three columns:

```
date,id,weight
1995-01-31,10001,0.012345
1995-01-31,10002,-0.008901
...
```

- `date` is the end-of-month at which the weights are set
- `id` is the stock identifier
- `weight` is the portfolio weight (positive = long, negative = short)

Weights with absolute value below $1e-8$ are not written. The gross exposure across all weights at each date should be close to the regime target (1.2 in bull, 0.8 in bear), and the net should be near +0.4 or -0.1 respectively.

Key Parameters

Parameter	Value	What it controls
<code>BURN_IN_MONTHS</code>	120	Months of history required before first prediction
<code>REFIT_EVERY</code>	12	How often (in months) to retrain models
<code>CV_FOLDS</code>	5	Folds for regularization selection
<code>ALPHA_GRID</code>	[0.01..1000]	Ridge regularization candidates
<code>REGIME_WINDOW</code>	12	Months for trailing return signal

Parameter	Value	What it controls
HYSTERESIS_K	2	Consecutive months needed to flip regime
GROSS_BULL	1.20	Total long+short exposure in bull
NET_BULL	0.40	Long minus short in bull
GROSS_BEAR	0.80	Total exposure in bear
NET_BEAR	-0.10	Net in bear (slight short bias)
MAX_ABS_WEIGHT	0.03	Per-stock weight cap
BLEND_BAND	0.04	Range over which bull/bear targets are blended
EARLY_BULL_GROSS_BONUS	0.20	Extra gross at start of new bull market
VOL_WINDOW	12	Months for vol estimation
SMOOTHING_ALPHA	0.50	Weight given to new vs old weights
WINSOR_P	(0.01, 0.99)	Outlier clipping percentiles

How to Run

```
cd DS194HW4
python strategy.py
```

Output goes to `weights_output.csv`. The script also prints: - Progress during preprocessing and the prediction loop - Model refit logs (date, regime, number of training months, selected alpha) - A verification summary showing average gross/net/max weight by regime - Annualized Sharpe ratio for the full period and broken down by bull/bear

Dependencies: `numpy`, `pandas`, `pyarrow` (no `sklearn` or `scipy` needed).

Design Decisions and Findings

Why separate bull/bear models instead of one model with a regime dummy?

A dummy variable would let the model shift the intercept between regimes but would still force the same factor coefficients everywhere. The empirical literature (and common sense) suggests that the *relative importance* of factors changes across regimes — value is relatively more important in bear markets, momentum crashes at market bottoms, etc. Two separate models can learn entirely different coefficient vectors, which is a much richer representation.

Why Ridge and not Lasso or neural networks?

Ridge has several practical advantages here: - It has a closed-form solution that can be computed from $X'X$ and $X'y$ alone, enabling the incremental accumulation trick. - It's stable: with $\alpha > 0$, the system is always positive definite, so the solve never fails. - With 40+ correlated features and limited effective sample size (after splitting by regime), Ridge's shrinkage toward zero is more appropriate than Lasso's sparsity. - Neural networks would require far more data per regime and careful training/regularization. With the expanding-window setup and regime splits, some early bear-market training sets have fewer than 1,000 months $\times$ 1,500 stocks of usable data.

Why rank-transform scores into weights?

Raw regression predictions can have extreme outliers — a single unusual stock might get a predicted return of 50% when everyone else is near 0. Rank-transforming first ensures the weight distribution is always approximately uniform across the portfolio, and no single outlier prediction dominates the sizing.

Why the early bull gross bonus?

The observation underlying this: when a sustained bear market ends and the market turns, the first few months of the new bull tend to have outsized gains — particularly in beaten-down, high-beta names. Being a bit more aggressive in those early months (higher gross exposure, tapering off as the bull matures) attempts to capture that effect. The bonus is capped via the blend-band logic so it only activates when the market is near the transition zone.

Why smooth weights over two months?

Real portfolios have transaction costs. Even assuming the backtested alpha is fully achievable, high monthly turnover erodes returns. The 50/50 smoothing halves the effective turnover rate at the cost

of a ~1-month lag in responding to new signals. Given that the factor signals here are all at least weekly in formation, a one-month lag is acceptable.

Expanding vs rolling window for training?

We use an expanding window — all historical regime-matched months, going back to the beginning. The main alternative is a rolling window (e.g., the last 5 years of bull months only). An expanding window produces more stable coefficient estimates and rewards the strategy for collecting more data over time. The main risk is that factor relationships may drift, but in practice this happens slowly enough that expanding window tends to outperform rolling in long-horizon factor studies.

No look-ahead guarantee

Every computation is strictly causal: - Preprocessing uses only the current month's cross-section. - Market returns and regime labels at date t use only $R_m[t-12 .. t-1]$. - Training data at date t consists of months 0 through $t-1$ only. - The per-month XTX accumulation asserts that each training date is strictly before the prediction date.