Package: PortfolioTesteR (via r-universe)

October 16, 2026

Type Package

Title Test Investment Strategies with English-Like Code

Version 0.1.4

Description Design, backtest, and analyze portfolio strategies using simple, English-like function chains. Includes technical indicators, flexible stock selection, portfolio construction methods (equal weighting, signal weighting, inverse volatility, hierarchical risk parity), and a compact backtesting engine for portfolio returns, drawdowns, and summary metrics.

License MIT + file LICENSE

URL https://github.com/AlbertoPallotta/PortfolioTesteR

BugReports https://github.com/AlbertoPallotta/PortfolioTesteR/issues

Depends R (>= 3.5.0)

Imports data.table, graphics, stats, TTR, utils, zoo

Suggests knitr, rmarkdown, testthat (>= 3.0.0), quantmod, RSQLite, rvest, glmnet, ranger, xgboost, keras, tensorflow

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/testthat/edition 3

ByteCompile true

VignetteBuilder knitr

Config/pak/sysreqs libssl-dev

Repository https://albertopallotta.r-universe.dev

Date/Publication 2026-08-17 14:44:15 UTC

RemoteUrl https://github.com/albertopallotta/portfoliotester

RemoteRef HEAD

RemoteSha 1cda69b18bb0328ff10e6779f42319ac567b8ca6

Contents

align_to_timeframe . . . 4
analyze_by_period . . . 5
analyze_drawdowns . . . 6
analyze_performance . . . 7
analyze_vs_benchmark . . . 8
apply_regime . . . 9
as_selection . . . 10
backtest_metrics . . . 11
bucket_returns . . . 11
calc_cci . . . 12
calc_distance . . . 13
calc_market_breadth . . . 13
calc_momentum . . . 14
calc_moving_average . . . 15
calc_relative_strength_rank . . . 15
calc_rolling_correlation . . . 16
calc_rolling_volatility . . . 17
calc_rsi . . . 18
calc_sector_breadth . . . 18
calc_sector_relative_indicators . . . 19
calc_stochastic_d . . . 20
calc_stochrsi . . . 21
calculate_daily_values . . . 22
calculate_drawdown_series . . . 23
cap_exposure . . . 24
cap_turnover . . . 25
carry_forward_weights . . . 26
combine_filters . . . 26
combine_scores . . . 27
combine_weights . . . 28
convert_to_nweeks . . . 29
coverage_by_date . . . 29
create_regime_buckets . . . 30
csv_adapter . . . 31
cv_tune_seq . . . 32
demo_sector_map . . . 33
download_sp500_sectors . . . 34
ensure_dt_copy . . . 34
evaluate_scores . . . 35
filter_above . . . 36
filter_below . . . 36
filter_between . . . 37
filter_by_percentile . . . 37
filter_rank . . . 38
filter_threshold . . . 39
filter_top_n . . . 39

filter_top_n_where . . . 40
get_data_frequency . . . 41
ic_series . . . 41
invert_signal . . . 42
join_panels . . . 43
limit_positions . . . 43
list_examples . . . 44
load_mixed_symbols . . . 45
make_labels . . . 46
manual_adapter . . . 47
membership_stability . . . 47
metric_sharpe . . . 48
ml_add_interactions . . . 48
ml_backtest . . . 49
ml_backtest_multi . . . 50
ml_backtest_seq . . . 52
ml_ic_series_on_scores . . . 54
ml_make_ensemble . . . 54
ml_make_model . . . 55
ml_make_seq_model . . . 56
ml_panel_op . . . 57
ml_panel_reduce . . . 58
ml_plot_ic_roll . . . 59
ml_prepare_features . . . 59
panel_lag . . . 60
panel_returns_simple . . . 61
perf_metrics . . . 61
plot.backtest_result . . . 62
plot.param_grid_result . . . 62
plot.performance_analysis . . . 64
plot.wf_optimization_result . . . 65
portfolio_returns . . . 67
print.backtest_result . . . 68
print.param_grid_result . . . 69
print.performance_analysis . . . 69
print.wf_optimization_result . . . 70
pt_collect_results . . . 70
rank_within_sector . . . 73
rebalance_calendar . . . 74
roll_fit_predict . . . 74
roll_fit_predict_seq . . . 75
roll_ic_stats . . . 77
run_backtest . . . 77
run_example . . . 78
run_param_grid . . . 79
run_walk_forward . . . 80
safe_divide . . . 81
sample_prices_daily . . . 81

sample_prices_weekly . . . 82
sample_sp500_sectors . . . 83
scores_oos_only . . . 84
select_top_k_scores . . . 85
select_top_k_scores_by_group . . . 85
sql_adapter . . . 86
sql_adapter_adjusted . . . 87
summary.backtest_result . . . 88
switch_weights . . . 89
transform_scores . . . 90
tune_ml_backtest . . . 90
turnover_by_date . . . 91
update_vix_in_db . . . 92
validate_data_format . . . 92
validate_group_map . . . 93
validate_no_leakage . . . 94
vol_target . . . 94
weight_by_hrp . . . 95
weight_by_rank . . . 97
weight_by_regime . . . 98
weight_by_risk_parity . . . 99
weight_by_signal . . . 100
weight_by_volatility . . . 101
weight_equally . . . 102
weight_from_scores . . . 102
wf_report . . . 103
wf_stitch . . . 104
wf_sweep_tabular . . . 104
yahoo_adapter . . . 105

Index **107**

`align_to_timeframe` *Align Data to Strategy Timeframe*

Description

Aligns higher-frequency data to match strategy timeframe.

Usage

```
align_to_timeframe(
  high_freq_data,
  low_freq_dates,
  method = c("forward_fill", "nearest", "interpolate")
)
```

Arguments

high_freq_data	Data frame to align
low_freq_dates	Date vector from strategy
method	Alignment method: "forward_fill", "nearest", or "interpolate"

Value

Aligned data frame

Examples

```
data("sample_prices_weekly")
data("sample_prices_daily")
momentum <- calc_momentum(sample_prices_weekly, lookback = 12)
selected <- filter_top_n(momentum, 10)
# Create a stability signal from daily data
daily_vol <- calc_rolling_volatility(sample_prices_daily, lookback = 20)
stability_signal <- align_to_timeframe(daily_vol, sample_prices_weekly$Date)
weights <- weight_by_signal(selected, stability_signal)
```

---

analyze_by_period *Period-level summary statistics*

Description

Aggregates portfolio results by calendar period and computes standard statistics for each period. Provide at least one of `returns` or `values`.

Usage

```
analyze_by_period(
  dates,
  returns = NULL,
  values = NULL,
  period = c("monthly", "quarterly", "yearly"),
  na_rm = TRUE
)
```

Arguments

dates	Date vector aligned to `returns` / `values`.
returns	Numeric simple returns aligned to `dates` (optional).
values	Numeric equity values aligned to `dates` (optional).
period	"monthly", "quarterly", or "yearly".
na_rm	Logical; remove NAs inside per-period aggregations.

Value

data.frame with period keys and columns: ret, start_value, end_value, n_obs.

Examples

```
data(sample_prices_weekly)
mom12 <- PortfolioTesteR::calc_momentum(sample_prices_weekly, lookback = 12)
sel5  <- PortfolioTesteR::filter_top_n(mom12, n = 5)
w_eq  <- PortfolioTesteR::weight_equally(sel5)
pr    <- PortfolioTesteR::portfolio_returns(w_eq, sample_prices_weekly)
val   <- 1e5 * cumprod(1 + pr$portfolio_return)
out   <- analyze_by_period(
  dates   = pr$Date,
  returns = pr$portfolio_return,
  values  = val,
  period  = "monthly"
)
head(out)
```

---

analyze_drawdowns *Analyze Drawdown Characteristics*

Description

Detailed analysis of drawdown periods including depth, duration, and recovery.

Usage

```
analyze_drawdowns(drawdowns, returns)
```

Arguments

drawdowns	Drawdown series (negative values)
returns	Return series for additional metrics

Value

List with drawdown statistics

Examples

```
data("sample_prices_weekly")
momentum <- calc_momentum(sample_prices_weekly, lookback = 12)
selected <- filter_top_n(momentum, n = 10)
weights <- weight_equally(selected)
result <- run_backtest(sample_prices_weekly, weights)
dd_analysis <- analyze_drawdowns(result$portfolio_value, result$dates)
```

`analyze_performance` *Analyze Backtest Performance with Daily Monitoring*

Description

Calculates comprehensive performance metrics using daily price data for enhanced accuracy. Provides risk-adjusted returns, drawdown analysis, and benchmark comparison even when strategy trades at lower frequency.

Usage

```
analyze_performance(
  backtest_result,
  daily_prices,
  benchmark_symbol = "SPY",
  rf_rate = 0,
  confidence_level = 0.95
)
```

Arguments

`backtest_result`	Result object from run_backtest()
`daily_prices`	Daily price data including all portfolio symbols
`benchmark_symbol`	Symbol for benchmark comparison (default: "SPY")
`rf_rate`	Annual risk-free rate for Sharpe/Sortino (default: 0)
`confidence_level`	Confidence level for VaR/CVaR (default: 0.95)

Value

performance_analysis object with metrics and daily tracking

Examples

```
data("sample_prices_weekly")
data("sample_prices_daily")

# Use overlapping symbols; cap to 3
syms_all <- intersect(names(sample_prices_weekly)[-1], names(sample_prices_daily)[-1])
stopifnot(length(syms_all) >= 1)
syms <- syms_all[seq_len(min(3L, length(syms_all)))]

# Subset weekly (strategy) and daily (monitoring) to the same symbols
P <- sample_prices_weekly[, c("Date", syms), with = FALSE]
D <- sample_prices_daily[,  c("Date", syms), with = FALSE]
```

```
# Simple end-to-end example
mom <- calc_momentum(P, lookback = 12)
sel <- filter_top_n(mom, n = 3)
W   <- weight_equally(sel)
res <- run_backtest(P, W)

# Pick a benchmark that is guaranteed to exist in D
perf <- analyze_performance(res, D, benchmark_symbol = syms[1])
print(perf)
summary(perf)
```

`analyze_vs_benchmark` *Benchmark-relative performance statistics*

Description

Computes standard benchmark-relative metrics (e.g., correlation, beta, alpha, tracking error, information ratio) by aligning portfolio returns with benchmark returns derived from prices.

Usage

```
analyze_vs_benchmark(
  portfolio_returns,
  benchmark_prices,
  dates,
  benchmark_symbol = "SPY"
)
```

Arguments

`portfolio_returns`	A numeric vector of portfolio simple returns aligned to `dates`.
`benchmark_prices`	A data frame (Date + symbols) of adjusted benchmark prices at the same cadence as `dates`.
`dates`	A vector of `Date` values used to align `portfolio_returns` with `benchmark_prices`.
`benchmark_symbol`	Character scalar giving the column name (symbol) in `benchmark_prices` to use as the benchmark.

Value

A list or data frame with benchmark-relative statistics according to the package's conventions, including correlation, beta, alpha, tracking error, and information ratio.

Examples

```
data(sample_prices_weekly)
mom12 <- PortfolioTesteR::calc_momentum(sample_prices_weekly, lookback = 12)
sel10 <- PortfolioTesteR::filter_top_n(mom12, n = 5)
w_eq  <- PortfolioTesteR::weight_equally(sel10)
pr    <- PortfolioTesteR::portfolio_returns(w_eq, sample_prices_weekly)

# Use SPY as the benchmark
bench <- sample_prices_weekly[, c("Date", "SPY")]
res <- analyze_vs_benchmark(
  pr$portfolio_return,
  bench,
  dates = pr$Date,
  benchmark_symbol = "SPY"
)
res
```

`apply_regime` *Apply Market Regime Filter*

Description

Applies regime-based filtering. When regime is FALSE (e.g., bear market), all selections become 0, moving portfolio to cash.

Usage

```
apply_regime(selection_df, regime_condition, partial_weight = 0)
```

Arguments

`selection_df`	Binary selection matrix
`regime_condition`	Logical vector (TRUE = trade, FALSE = cash)
`partial_weight`	Fraction to hold when regime is FALSE (default: 0)

Value

Modified selection matrix respecting regime

Examples

```
data("sample_prices_weekly")
# Create selection
momentum <- calc_momentum(sample_prices_weekly, 12)
selected <- filter_top_n(momentum, 10)
```

```
# Only trade when SPY above 20-week MA
ma20 <- calc_moving_average(sample_prices_weekly, 20)
spy_regime <- sample_prices_weekly$SPY > ma20$SPY
spy_regime[is.na(spy_regime)] <- FALSE

regime_filtered <- apply_regime(selected, spy_regime)
```

---

as_selection *Convert Conditions to Selection Format*

Description

Converts condition matrices or data frames to standard selection format with Date column and binary values. Handles NA by converting to 0.

Usage

```
as_selection(condition_matrix, date_column = NULL)
```

Arguments

condition_matrix	Matrix or data frame with conditions
date_column	Optional Date vector if not in input

Value

Data.table in selection format (Date + binary columns)

Examples

```
data("sample_prices_weekly")
ma20 <- calc_moving_average(sample_prices_weekly, 20)
above_ma <- filter_above(calc_distance(sample_prices_weekly, ma20), 0)
selection <- as_selection(above_ma, sample_prices_weekly$Date)
```

backtest_metrics *Calculate Comprehensive Backtest Metrics*

Description

Computes performance metrics including Sharpe ratio, maximum drawdown, win rate, and other statistics from backtest results.

Usage

```
backtest_metrics(result)
```

Arguments

result	Backtest result object from run_backtest()

Value

List containing performance metrics

Examples

```
# Create a backtest result to use
data(sample_prices_weekly)
momentum <- calc_momentum(sample_prices_weekly, lookback = 12)
selected <- filter_top_n(momentum, n = 10)
weights <- weight_equally(selected)
result <- run_backtest(sample_prices_weekly, weights)

# Calculate metrics
metrics <- backtest_metrics(result)
print(metrics$sharpe_ratio)
```

bucket_returns *Bucketed label analysis by score rank*

Description

Bucketed label analysis by score rank

Usage

```
bucket_returns(
  scores,
  labels,
  n_buckets = 10L,
  label_type = c("log", "simple")
)
```

Arguments

scores	Wide score panel (Date + symbols).
labels	Wide label panel aligned to scores (Date + symbols).
n_buckets	Number of equal-count buckets.
label_type	Use 'log' or 'simple' label arithmetic.

Value

data.table of per-date bucket returns; cumulative series attached as `attr(result, "cum")`.

---

calc_cci *Calculate Commodity Channel Index (CCI)*

Description

Calculates CCI using closing prices. CCI measures deviation from average price. Values above 100 indicate overbought, below -100 indicate oversold.

Usage

```
calc_cci(data, period = 20)
```

Arguments

data	Data frame with Date column and price columns
period	CCI period (default: 20)

Value

Data.table with CCI values

Examples

```
data("sample_prices_weekly")
cci <- calc_cci(sample_prices_weekly, period = 20)
```

calc_distance *Calculate Distance from Reference*

Description

data("sample_prices_weekly") Calculates percentage distance between prices and reference values (typically moving averages).

Usage

```
calc_distance(price_df, reference_df)
```

Arguments

price_df	Data frame with price data
reference_df	Data frame with reference values (same structure)

Value

Data.table with percentage distances

Examples

```
data("sample_prices_weekly")
ma20 <- calc_moving_average(sample_prices_weekly, 20)
data("sample_prices_weekly")
distance <- calc_distance(sample_prices_weekly, ma20)
```

calc_market_breadth *Calculate Market Breadth Percentage*

Description

Measures the percentage of stocks meeting a condition (market participation). Useful for assessing market health and identifying broad vs narrow moves.

Usage

```
calc_market_breadth(condition_df, min_stocks = 10)
```

Arguments

condition_df	Data frame with Date column and TRUE/FALSE values
min_stocks	Minimum stocks required for valid calculation (default: 10)

Value

A `data.table` with `Date` and `Breadth_[Sector]` columns (0-100 scale)

Examples

```
# Percent of stocks above 200-day MA
data("sample_prices_weekly")
ma200 <- calc_moving_average(sample_prices_weekly, 200)
above_ma <- filter_above(calc_distance(sample_prices_weekly, ma200), 0)
breadth <- calc_market_breadth(above_ma)
```

`calc_momentum` *Calculate Price Momentum*

Description

Calculates momentum as the percentage change in price over a specified lookback period. Optimized using column-wise operations (25x faster).

Usage

```
calc_momentum(data, lookback = 12)
```

Arguments

`data`	A data.frame or data.table with Date column and price columns
`lookback`	Number of periods for momentum calculation (default: 12)

Value

Data.table with momentum values (0.1 = 10% increase)

Examples

```
data("sample_prices_weekly")
momentum <- calc_momentum(sample_prices_weekly, lookback = 12)
```

calc_moving_average *Calculate Moving Average*

Description

Calculates simple moving average for each column in the data.

Usage

```
calc_moving_average(data, window = 20)
```

Arguments

data	Data frame with Date column and price columns
window	Number of periods for moving average (default: 20)

Value

Data.table with moving average values

Examples

```
data("sample_prices_weekly")
ma20 <- calc_moving_average(sample_prices_weekly, window = 20)
```

calc_relative_strength_rank *Calculate Cross-Sectional Ranking of Indicators*

Description

Ranks each stock's indicator value against all other stocks on the same date. Enables relative strength strategies that adapt to market conditions. Optimized using matrix operations for 15x speedup.

Usage

```
calc_relative_strength_rank(
  indicator_df,
  method = c("percentile", "rank", "z-score")
)
```

Arguments

indicator_df	Data frame with Date column and indicator values
method	Ranking method: "percentile" (0-100), "rank" (1-N), or "z-score"

Value

Data frame with same structure containing ranks/scores

Examples

```
# Rank RSI across all stocks
data("sample_prices_weekly")
rsi <- calc_rsi(sample_prices_weekly, 14)
rsi_ranks <- calc_relative_strength_rank(rsi, method = "percentile")

# Find relatively overbought (top 10%)
relative_overbought <- filter_above(rsi_ranks, 90)
```

---

calc_rolling_correlation

Rolling correlation of each symbol to a benchmark

Description

Computes rolling correlations between each symbol and a benchmark series (e.g., SPY) using simple returns over a fixed lookback window.

Usage

```
calc_rolling_correlation(
  data,
  benchmark_symbol = "SPY",
  lookback = 60,
  min_periods = NULL,
  method = c("pearson", "spearman")
)
```

Arguments

data	A data.frame or data.table with a Date column and one column per symbol containing prices. Must include benchmark_symbol.
benchmark_symbol	Character, the benchmark column name (default "SPY").
lookback	Integer window size (>= 2) for rolling correlations.
min_periods	Minimum number of valid observations within the window to compute a correlation. Default is ceiling(lookback * 0.67).
method	Correlation method, "pearson" (default) or "spearman".

Details

Returns are computed as simple returns $(P_t - P_{t-1})/P_{t-1}$. Windows with fewer than min_periods valid pairs are marked NA.

Value

A `data.table` with `Date` and one column per non-benchmark symbol, containing rolling correlations. Insufficient data yields `NA`s.

See Also

`calc_momentum()`, `calc_rolling_volatility()`

Examples

```
data(sample_prices_weekly)
corr <- calc_rolling_correlation(
  data = sample_prices_weekly,
  benchmark_symbol = "SPY",
  lookback = 20
)
head(corr)
```

---

calc_rolling_volatility

Calculate Rolling Volatility

Description

Calculates rolling volatility using various methods including standard deviation, range-based, MAD, or absolute returns. Supports different lookback periods.

Usage

```
calc_rolling_volatility(data, lookback = 20, method = "std")
```

Arguments

`data`	Data frame with Date column and price columns
`lookback`	Number of periods for rolling calculation (default: 20)
`method`	Volatility calculation method: "std", "range", "mad", or "abs_return"

Value

Data frame with Date column and volatility values for each symbol

Examples

```
data("sample_prices_weekly")
# Standard deviation volatility
vol <- calc_rolling_volatility(sample_prices_weekly, lookback = 20)
# Range-based volatility
vol_range <- calc_rolling_volatility(sample_prices_weekly, lookback = 20, method = "range")
```

calc_rsi *Calculate Relative Strength Index (RSI)*

Description

Calculates RSI for each column. RSI ranges from 0-100. Above 70 indicates overbought, below 30 indicates oversold.

Usage

```
calc_rsi(data, period = 14)
```

Arguments

data	Data frame with Date column and price columns
period	RSI period (default: 14)

Value

Data.table with RSI values (0-100 range)

Examples

```
data("sample_prices_weekly")
rsi <- calc_rsi(sample_prices_weekly, period = 14)
overbought <- filter_above(rsi, 70)
```

calc_sector_breadth *Calculate Market Breadth by Sector*

Description

Measures participation within each sector separately, revealing which sectors have broad strength vs concentrated leadership. Optimized using pre-splitting for speed.

Usage

```
calc_sector_breadth(
  condition_df,
  sector_mapping,
  min_stocks_per_sector = 3,
  na_sector_action = c("exclude", "separate", "market")
)
```

Arguments

condition_df	Data frame with Date column and TRUE/FALSE values
sector_mapping	Data frame with Symbol and Sector columns.
min_stocks_per_sector	Minimum stocks for valid sector breadth (default: 3)
na_sector_action	How to handle unmapped stocks: "exclude", "separate", or "market"

Value

A data.table with Date and Breadth_[Sector] columns (0-100 scale)

Examples

```
data("sample_prices_weekly")
data("sample_sp500_sectors")
ma200 <- calc_moving_average(sample_prices_weekly, 200)
above_ma <- filter_above(calc_distance(sample_prices_weekly, ma200), 0)
sector_breadth <- calc_sector_breadth(above_ma, sample_sp500_sectors)
```

---

calc_sector_relative_indicators

Calculate Indicators Relative to Sector Average

Description

Measures how each stock's indicator compares to its sector benchmark. Enables sector-neutral strategies and identifies sector outperformers.

Usage

```
calc_sector_relative_indicators(
  indicator_df,
  sector_mapping,
  method = c("difference", "ratio", "z-score"),
  benchmark = c("mean", "median"),
  ratio_threshold = 0.01,
  min_sector_size = 2
)
```

Arguments

indicator_df	Data frame with Date column and indicator values
sector_mapping	Data frame with Symbol and Sector columns.
method	"difference" (absolute), "ratio" (relative), or "z-score"
benchmark	"mean" or "median" sector average

`ratio_threshold`
: Minimum denominator for ratio method (default: 0.01)

`min_sector_size`
: Minimum stocks per sector (default: 2)

Value

Data frame with sector-relative values

Examples

```
# Find stocks outperforming their sector
data("sample_prices_weekly")
data("sample_sp500_sectors")
momentum <- calc_momentum(sample_prices_weekly, 12)
relative_momentum <- calc_sector_relative_indicators(
  momentum, sample_sp500_sectors, method = "difference"
)
```

---

calc_stochastic_d *Calculate Stochastic D Indicator*

Description

Calculates the Stochastic D indicator for momentum analysis. The %D line is the smoothed version of %K, commonly used for momentum signals in range 0-100.

Usage

```
calc_stochastic_d(data, k = 14, d = 3)
```

Arguments

`data`	Price data with Date column and symbol columns
`k`	Lookback period for stochastic K calculation
`d`	Smoothing period for D line

Value

Data.table with Stochastic D values for each symbol

Examples

```
data("sample_prices_weekly")
data(sample_prices_weekly)
data("sample_prices_weekly")
stoch_d <- calc_stochastic_d(sample_prices_weekly, k = 14, d = 3)
head(stoch_d)
```

`calc_stochrsi` *Stochastic RSI (StochRSI) for multiple price series*

Description

Computes Stochastic RSI (%K) per column over a rolling window, returning values in [0, 1]. For each symbol, RSI is computed with `TTR::RSI()` over `rsi_length` periods; then StochRSI is $(RSI_t - \min RSI_{t-L+1:t})/(\max RSI_{t-L+1:t} - \min RSI_{t-L+1:t})$, where L is `stoch_length`. If the range is zero the value is handled per `on_const_window` (default `"zero"`).

Usage

```
calc_stochrsi(
  data,
  length = 14L,
  rsi_length = NULL,
  stoch_length = NULL,
  on_const_window = c("zero", "na")
)
```

Arguments

`data`	A `data.frame` or `data.table` with a `Date` column and one price column per symbol (wide format).
`length`	Integer lookback used when `rsi_length`/`stoch_length` are NULL. Default `14`.
`rsi_length`	Optional integer RSI lookback. Default: `length`.
`stoch_length`	Optional integer stochastic window. Default: `length`.
`on_const_window`	How to handle windows where `maxRSI == minRSI`? One of `"zero"` (set to 0), `"na"` (leave `NA`). Default `"zero"`.

Value

A `data.table` with `Date` and symbol columns containing StochRSI in [0, 1], with leading `NA`s for warmup.

See Also

`TTR::RSI()`, `calc_momentum()`, `calc_moving_average()`, `filter_top_n()`, `weight_by_risk_parity()`

Examples

```
data(sample_prices_weekly)
s <- calc_stochrsi(sample_prices_weekly, length = 14)
head(s)
```

`calculate_daily_values` *Daily equity curve from positions and daily prices*

Description

Carries portfolio positions (from a weekly or lower-frequency backtest) forward to daily dates, multiplies by daily prices, and combines with cash to produce a daily portfolio value series for monitoring and analytics.

Usage

```
calculate_daily_values(
  positions,
  daily_prices,
  strategy_dates,
  initial_capital,
  cash_values
)
```

Arguments

`positions`	A `data.frame`/`data.table` of portfolio positions with columns `Date` + symbols. Values should be the backtest's **position inventory** per symbol at each rebalance date (typically shares or notional units consistent with your backtest's accounting).
`daily_prices`	A `data.frame`/`data.table` of **daily** prices with columns `Date` + the same symbol set present in `positions` (at least the intersection).
`strategy_dates`	A `Date` vector of the backtest's decision/rebalance calendar (e.g., `backtest_result$dates`).
`initial_capital`	Numeric scalar. Starting cash used for days **before** the first position exists (typically `backtest_result$initial_capital`).
`cash_values`	Optional numeric vector of cash balances at the strategy dates (e.g., `backtest_result$cash`). If `NULL`, leading days are treated as all-cash (= `initial_capital`) and post-rebalance cash defaults to 0.

Value

A list with components:

- `dates` Daily dates within the strategy span.
- `portfolio_values` Daily total portfolio value (positions + cash).
- `positions_value` Daily mark-to-market of positions only.
- `cash` Daily carried cash series.

Examples

```
# Minimal end-to-end example using bundled data and a simple weekly backtest
library(PortfolioTesteR)
data(sample_prices_weekly); data(sample_prices_daily)

# Build a tiny strategy: momentum -> top-3 -> equal weights
mom <- calc_momentum(sample_prices_weekly, lookback = 12)
sel <- filter_top_n(mom, n = 3)
W   <- weight_equally(sel)
bt  <- run_backtest(sample_prices_weekly, W, name = "Demo")

# Compute daily monitoring values from positions + cash
vals <- calculate_daily_values(
  positions       = bt$positions,
  daily_prices    = sample_prices_daily,
  strategy_dates  = bt$dates,
  initial_capital = bt$initial_capital,
  cash_values     = bt$cash
)

# Quick sanity checks
head(vals$dates)
head(vals$portfolio_values)
```

`calculate_drawdown_series` *Calculate Drawdown Time Series*

Description

Computes drawdown series from portfolio values.

Usage

```
calculate_drawdown_series(values)
```

Arguments

`values`	Numeric vector of portfolio values

Value

Numeric vector of drawdowns (as negative percentages)

Examples

```
data("sample_prices_weekly")
momentum <- calc_momentum(sample_prices_weekly, lookback = 12)
sel <- filter_top_n(momentum, n = 10)
W   <- weight_equally(sel)
res <- run_backtest(sample_prices_weekly, W)
dd_series <- calculate_drawdown_series(res$portfolio_values)
dd_stats  <- analyze_drawdowns(dd_series, res$returns)
```

---

cap_exposure *Apply post-weight exposure caps*

Description

Row-wise caps on a wide weights table (`Date` + symbols):

1. per-symbol cap (`max_per_symbol`)
2. optional per-group cap (`max_per_group`) using `group_map`

Usage

```
cap_exposure(
  weights,
  max_per_symbol = NULL,
  group_map = NULL,
  max_per_group = NULL,
  allow_short = FALSE,
  renormalize = c("none", "down", "both"),
  renormalize_down = FALSE,
  renorm = NULL,
  cash_col = NULL,
  caps = NULL
)

cap_exposure(
  weights,
  max_per_symbol = NULL,
  group_map = NULL,
  max_per_group = NULL,
  allow_short = FALSE,
  renormalize = c("none", "down", "both"),
  renormalize_down = FALSE,
  renorm = NULL,
  cash_col = NULL,
  caps = NULL
)
```

Arguments

weights	data.frame/data.table: columns Date, then symbols.
max_per_symbol	numeric scalar or named vector (absolute cap per symbol).
group_map	optional named character vector or data.frame(symbol,group).
max_per_group	optional numeric scalar (per-group gross cap).
allow_short	logical; if FALSE clamp negatives to 0.
renormalize	character: one of "none", "down", "both". Default "none".
renormalize_down	logical; deprecated alias for down-only (kept for compatibility).
renorm	logical; deprecated alias; if TRUE acts like renormalize="both".
cash_col	optional character; if provided, set to 1 - sum(pmax(w,0)).
caps	optional list; alternative to split args (see details above).

Details

Negatives are clamped to 0 unless allow_short = TRUE.

Renormalization options:

- renormalize = "none": leave gross (sum of positive weights) as is.
- renormalize = "down": if gross > 1, scale down so gross == 1.
- renormalize = "both": if 0 < gross != 1, scale so gross == 1.

The argument renorm (logical) is a deprecated alias; TRUE behaves like renormalize = "both".

The caps list form is supported:

max_per_symbol Maximum absolute weight per symbol (0-1).

max_per_group Maximum gross weight per group (0-1).

group_map Named character vector mapping symbol -> group.

Value

data.table of capped (and optionally renormalized) weights.

---

cap_turnover *Cap turnover sequentially across dates*

Description

Cap turnover sequentially across dates

Usage

```
cap_turnover(weights, max_turnover = 0.2)
```

Arguments

weights	desired weight panel.
max_turnover	maximum per-rebalance turnover (0..1).

Value

executed weights after turnover capping.

---

carry_forward_weights *Carry-forward weights between rebalances (validation helper)*

Description

Carry-forward weights between rebalances (validation helper)

Usage

```
carry_forward_weights(weights)
```

Arguments

weights	Wide weight panel (Date + symbols) with weights only on rebalance rows.

Value

weight panel with rows filled forward and each active row sum=1.

---

combine_filters *Combine Multiple Filter Conditions*

Description

Combines multiple filter conditions using AND or OR logic.

Usage

```
combine_filters(..., op = "and", apply_when = NULL, debug = FALSE)
```

Arguments

...	Two or more filter data frames to combine
op	Operation: "and" or "or"
apply_when	Optional condition vector for conditional filtering
debug	Print debug information (default: FALSE)

Value

Combined binary selection matrix

Examples

```
data("sample_prices_weekly")
momentum <- calc_momentum(sample_prices_weekly, 12)
rsi <- calc_rsi(sample_prices_weekly, 14)
# Create individual filters
high_momentum <- filter_above(momentum, 0.05)
moderate_rsi <- filter_between(rsi, 40, 60)
# Combine them
combined <- combine_filters(high_momentum, moderate_rsi, op = "and")
```

---

combine_scores *Combine multiple score panels (mean / weighted / rank-average / trimmed)*

Description

Combine several wide score panels (`Date` + symbols) into a single panel by applying one of several aggregation methods.

Usage

```
combine_scores(
  scores_list,
  method = c("mean", "weighted", "rank_avg", "trimmed_mean"),
  weights = NULL,
  trim = 0.1
)
```

Arguments

`scores_list`	List of wide score panels to combine (each has columns `Date` + symbols).
`method`	Character, one of `"mean"`, `"weighted"`, `"rank_avg"`, `"trimmed_mean"`.
`weights`	Optional numeric vector of length equal to `length(scores_list)`; used only when `method = "weighted"`.
`trim`	Numeric in `[0, 0.5)`; fraction to trim from each tail for `method = "trimmed_mean"`.

Details

- `method = "mean"`: simple column-wise mean across panels.
- `method = "weighted"`: weighted mean; see `weights`.
- `method = "rank_avg"`: average of within-date normalized ranks.
- `method = "trimmed_mean"`: mean with `trim` fraction removed at both tails.

Value

A `data.table` with columns `Date` + symbols, containing the combined scores.

`combine_weights` *Combine Multiple Weighting Schemes*

Description

Blends multiple weight matrices with specified weights. Useful for multi-factor strategies that combine different allocation approaches. Optimized using matrix operations for 1000x+ speedup.

Usage

```
combine_weights(weight_matrices, weights = NULL)
```

Arguments

`weight_matrices`	List of weight data frames to combine
`weights`	Numeric vector of weights for each matrix (default: equal)

Value

Data.table with blended portfolio weights

Examples

```
data("sample_prices_weekly")
# Calculate signals
momentum <- calc_momentum(sample_prices_weekly, lookback = 12)
selected <- filter_top_n(momentum, n = 10)
volatility <- calc_rolling_volatility(sample_prices_weekly, lookback = 20)

# Combine momentum and low-vol weights
mom_weights <- weight_by_signal(selected, momentum)
vol_weights <- weight_by_signal(selected, invert_signal(volatility))
combined <- combine_weights(list(mom_weights, vol_weights), weights = c(0.7, 0.3))
```

convert_to_nweeks *Convert Data to N-Week Frequency*

Description

Resamples daily or weekly data to n-week periods. Handles week-ending calculations and various aggregation methods.

Usage

```
convert_to_nweeks(data, n = 1, method = "last")
```

Arguments

data	Data.table with Date column and price columns
n	Number of weeks to aggregate (default: 1 for weekly)
method	Aggregation method: "last" or "mean" (default: "last")

Value

Data.table resampled to n-week frequency

Examples

```
data("sample_prices_daily")
# Convert daily to weekly
weekly <- convert_to_nweeks(sample_prices_daily, n = 1)
# Convert to bi-weekly
biweekly <- convert_to_nweeks(sample_prices_daily, n = 2)
```

coverage_by_date *Count finite entries per date*

Description

Count finite entries per date

Usage

```
coverage_by_date(panel)
```

Arguments

panel	wide panel Date + symbols.

Value

data.table with Date, n_finite.

`create_regime_buckets` *Convert Continuous Indicator to Discrete Regimes*

Description

Transforms continuous indicators into discrete regime categories.

Usage

```
create_regime_buckets(
  indicator,
  breakpoints,
  labels = NULL,
  use_percentiles = FALSE
)
```

Arguments

`indicator`	Numeric vector or data frame with indicator values
`breakpoints`	Numeric vector of breakpoints
`labels`	Optional character vector of regime names
`use_percentiles`	Use percentiles instead of fixed breakpoints (default: FALSE)

Value

Integer vector of regime classifications

Examples

```
data("sample_prices_weekly")
momentum <- calc_momentum(sample_prices_weekly, lookback = 12)
selected <- filter_top_n(momentum, 10)
# Create VIX-like indicator from volatility
vol <- calc_rolling_volatility(sample_prices_weekly, lookback = 20)
vix_proxy <- vol$SPY * 100  # Scale to VIX-like values
regimes <- create_regime_buckets(vix_proxy, c(15, 25))
```

csv_adapter *Load Price Data from CSV File*

Description

Reads stock price data from CSV files with flexible column naming. Automatically standardizes to library format.

Usage

```
csv_adapter(
  file_path,
  date_col = "Date",
  symbol_col = "Symbol",
  price_col = "Price",
  frequency = "daily",
  symbol_order = NULL
)
```

Arguments

file_path	Path to CSV file
date_col	Name of date column (default: "date")
symbol_col	Name of symbol column (default: "symbol")
price_col	Name of price column (default: "close")
frequency	Target frequency: "daily" or "weekly" (default: "daily")
symbol_order	Optional vector to order symbols

Value

Data.table with Date column and price columns

Examples

```
# Create a temporary tidy CSV from included weekly sample data (offline, fast)
data("sample_prices_weekly")
PW <- as.data.frame(sample_prices_weekly)
syms <- setdiff(names(PW), "Date")[1:2]

stk  <- stack(PW[1:10, syms])
tidy <- data.frame(
  Date   = rep(PW$Date[1:10], times = length(syms)),
  Symbol = stk$ind,
  Price  = stk$values
)

tmp <- tempfile(fileext = ".csv")
```

```
write.csv(tidy, tmp, row.names = FALSE)
prices <- csv_adapter(tmp)
head(prices)
unlink(tmp)
```

`cv_tune_seq` *Purged/embargoed K-fold CV for sequence models (inside IS window)*

Description

Purged/embargoed K-fold CV for sequence models (inside IS window)

Usage

```
cv_tune_seq(
  features_list,
  labels,
  is_start,
  is_end,
  steps_grid = c(12L, 26L, 52L),
  horizon = 4L,
  fit_fn,
  predict_fn,
  k = 3L,
  purge = NULL,
  embargo = NULL,
  group = c("pooled"),
  max_train_samples = NULL,
  max_val_samples = NULL,
  na_action = c("omit", "zero"),
  metric = c("spearman", "rmse")
)
```

Arguments

`features_list`	List of feature panels (Date + symbols) for sequences.
`labels`	Label panel aligned to features (Date + symbols).
`is_start, is_end`	Inclusive IS window indices (on aligned dates).
`steps_grid`	Integer vector of candidate sequence lengths.
`horizon`	Forecast horizon (periods ahead).
`fit_fn`	Function (X, y) -> model for sequences.
`predict_fn`	Function (model, Xnew) -> numeric scores.
`k`	Number of CV folds.
`purge`	Gap (obs) removed between train/val within folds (default uses steps).

embargo	Embargo (obs) after validation to avoid bleed (default uses horizon).
group	'pooled', 'per_symbol', or 'per_group'.
max_train_samples	Optional cap on IS samples per fold.
max_val_samples	Optional cap on validation samples per fold.
na_action	How to handle NA features ('omit' or 'zero').
metric	IC metric ('spearman' or 'pearson').

Value

data.table with columns like steps, folds, and CV score.

---

demo_sector_map *Demo sector (group) map for examples/tests*

Description

Demo sector (group) map for examples/tests

Usage

```
demo_sector_map(symbols, n_groups = 2L)
```

Arguments

symbols	character vector of tickers.
n_groups	integer number of groups to assign (cycled).

Value

A data.table with columns Symbol and Group (title-case), for demo use.

Examples

```
# Minimal usage
syms <- c("AAPL","MSFT","AMZN","XOM","JPM")
gdf  <- demo_sector_map(syms, n_groups = 3L)  # columns: Symbol, Group
print(gdf)

# Use with cap_exposure(): convert to a named vector (names = symbols)
gmap <- stats::setNames(gdf$Group, gdf$Symbol)

data(sample_prices_weekly)
mom12 <- calc_momentum(sample_prices_weekly, 12)
sel10 <- filter_top_n(mom12, 10)
w_eq  <- weight_equally(sel10)
```

```
w_cap <- cap_exposure(
  weights          = w_eq,
  max_per_symbol   = 0.10,
  group_map        = gmap,     # <- named vector, OK for current cap_exposure()
  max_per_group    = 0.45,
  renormalize_down = TRUE
)
head(w_cap)
```

`download_sp500_sectors` *Download S&P 500 Sector Mappings from Wikipedia*

Description

Scrapes current S&P 500 constituent list with sector classifications from Wikipedia and returns as a data.table.

Usage

```
download_sp500_sectors()
```

Value

Data.table with columns: Symbol, Security, Sector, SubIndustry, Industry

Examples

```
sectors <- download_sp500_sectors()
head(sectors)
```

`ensure_dt_copy` *Ensure Data.Table Without Mutation*

Description

Converts input to data.table if needed, always returning a copy to prevent accidental data mutation. Core safety function used throughout the library.

Usage

```
ensure_dt_copy(data)
```

Arguments

data	Data.frame or data.table

Value

Copy of data as data.table

Examples

```
data("sample_prices_weekly")
dt <- ensure_dt_copy(sample_prices_weekly)  # Safe to modify dt
```

---

evaluate_scores *Evaluate scores vs labels (IC and hit-rate)*

Description

Evaluate scores vs labels (IC and hit-rate)

Usage

```
evaluate_scores(
  scores,
  labels,
  top_frac = 0.2,
  method = c("spearman", "pearson")
)
```

Arguments

scores	Wide score panel.
labels	Wide label panel aligned to scores.
top_frac	fraction in the top bucket for hit-rate.
method	"spearman" or "pearson".

Value

data.table with Date, IC, hit_rate; ICIR on attr(result,"ICIR").

filter_above *Filter Stocks Above Threshold*

Description

Convenience function to select stocks with signal above a value.

Usage

```
filter_above(signal_df, value)
```

Arguments

signal_df	Data frame with signal values
value	Threshold value

Value

Binary selection matrix

Examples

```
data("sample_prices_weekly")
rsi <- calc_rsi(sample_prices_weekly, 14)
high_rsi <- filter_above(rsi, 70)
```

filter_below *Filter Stocks Below Threshold*

Description

Convenience function to select stocks with signal below a value.

Usage

```
filter_below(signal_df, value)
```

Arguments

signal_df	Data frame with signal values
value	Threshold value

Value

Binary selection matrix

Examples

```
data("sample_prices_weekly")
rsi <- calc_rsi(sample_prices_weekly, 14)
oversold <- filter_below(rsi, 30)
```

---

filter_between *Filter Stocks Between Two Values*

Description

Selects stocks with signal values between lower and upper bounds.

Usage

```
filter_between(signal_df, lower, upper)
```

Arguments

signal_df	Data frame with signal values
lower	Lower bound (inclusive)
upper	Upper bound (inclusive)

Value

Binary selection matrix

Examples

```
data("sample_prices_weekly")
rsi <- calc_rsi(sample_prices_weekly, 14)
# Select stocks with RSI between 30 and 70
neutral_rsi <- filter_between(rsi, 30, 70)
```

---

filter_by_percentile *Filter by Percentile*

Description

Select securities in the top or bottom X percentile. More intuitive than filter_top_n when universe size varies.

Usage

```
filter_by_percentile(signal_df, percentile, type = c("top", "bottom"))
```

Arguments

signal_df	DataFrame with signal values
percentile	Percentile threshold (0-100)
type	"top" for highest signals, "bottom" for lowest

Value

Binary selection matrix

Examples

```
data("sample_prices_weekly")
momentum <- calc_momentum(sample_prices_weekly, 12)
# Select top 20th percentile
top_20pct <- filter_by_percentile(momentum, 20, type = "top")
```

---

filter_rank *Select Top or Bottom N Stocks by Signal*

Description

Selects the top N (best) or worst N stocks based on signal strength. Optimized using matrix operations for 5-10x speedup.

Usage

```
filter_rank(signal_df, n, type = c("top", "worst"))
```

Arguments

signal_df	Data frame with Date column and signal values
n	Number of stocks to select
type	"top" for highest values, "worst" for lowest values

Value

Binary selection matrix (1 = selected, 0 = not selected)

Examples

```
data("sample_prices_weekly")
momentum <- calc_momentum(sample_prices_weekly, 12)
# Select 10 highest momentum stocks
top10 <- filter_rank(momentum, 10, type = "top")
```

filter_threshold *Filter by Threshold Value*

Description

Selects stocks above or below a threshold value.

Usage

```
filter_threshold(signal_df, value, type = c("above", "below"))
```

Arguments

signal_df	Data frame with signal values
value	Threshold value
type	"above" or "below"

Value

Binary selection matrix

Examples

```
data("sample_prices_weekly")
momentum <- calc_momentum(sample_prices_weekly, 12)
# Select stocks with positive momentum
positive <- filter_threshold(momentum, 0, type = "above")
```

filter_top_n *Select Top N Stocks by Signal Value*

Description

Most commonly used filter function. Selects top N (highest) or bottom N (lowest) stocks by signal value. Optimized for 5-10x faster performance.

Usage

```
filter_top_n(signal_df, n, ascending = FALSE)
```

Arguments

signal_df	Data frame with Date column and signal values
n	Number of stocks to select
ascending	FALSE (default) selects highest, TRUE selects lowest

Value

Binary selection matrix (1 = selected, 0 = not selected)

Examples

```
data("sample_prices_weekly")
momentum <- calc_momentum(sample_prices_weekly, 12)
# Select 10 highest momentum stocks
top_momentum <- filter_top_n(momentum, n = 10)
```

---

filter_top_n_where *Select Top N from Qualified Stocks*

Description

Selects top N stocks by signal, but only from those meeting a condition. Combines qualification and ranking in one step.

Usage

```
filter_top_n_where(
  signal_df,
  n,
  condition_df,
  min_qualified = 1,
  ascending = FALSE
)
```

Arguments

signal_df	Signal values for ranking
n	Number to select
condition_df	Binary matrix of qualified stocks
min_qualified	Minimum qualified stocks required (default: 1)
ascending	FALSE for highest, TRUE for lowest

Value

Binary selection matrix

Examples

```
data("sample_prices_weekly")
# Calculate indicators
momentum <- calc_momentum(sample_prices_weekly, 12)
ma20 <- calc_moving_average(sample_prices_weekly, 20)
distance_from_ma <- calc_distance(sample_prices_weekly, ma20)

# Top 10 momentum stocks from those above MA
above_ma <- filter_above(distance_from_ma, 0)
top_qualified <- filter_top_n_where(momentum, 10, above_ma)
```

---

get_data_frequency *Detect Data Frequency from Dates*

Description

Automatically detects whether data is daily, weekly, monthly, or quarterly based on date spacing.

Usage

```
get_data_frequency(dates)
```

Arguments

dates	Vector of Date objects

Value

Character string: "daily", "weekly", "monthly", or "quarterly"

Examples

```
data("sample_prices_weekly")
freq <- get_data_frequency(sample_prices_weekly$Date)
```

---

ic_series *Information Coefficient time series*

Description

Information Coefficient time series

Usage

```
ic_series(scores, labels, method = c("spearman", "pearson"))
```

Arguments

scores	Wide score panel (Date + symbols).
labels	Wide label panel aligned to scores.
method	IC method ('spearman' or 'pearson').

Value

data.table with Date and IC.

---

invert_signal *Invert Signal Values for Preference Reversal*

Description

Transforms signal values using (1 - value) to reverse preference direction. Useful when high values indicate something to avoid. For example, inverting volatility makes low-vol stocks appear as high signals.

Usage

```
invert_signal(signal_df)
```

Arguments

signal_df	Data frame with Date column and signal columns

Value

Data frame with inverted signal values

Examples

```
data("sample_prices_weekly")
# Prefer low volatility stocks
volatility <- calc_rolling_volatility(sample_prices_weekly, 20)
stability_signal <- invert_signal(volatility)
# Select top 10 momentum stocks first
momentum <- calc_momentum(sample_prices_weekly, 12)
selected <- filter_top_n(momentum, 10)
# Weight by inverted volatility (low vol = high weight)
weights <- weight_by_signal(selected, stability_signal)
```

join_panels *Join multiple panels on intersecting dates (unique symbol names)*

Description

Align and join a list of wide panels on their common dates. Each input panel must have one `Date` column and a disjoint set of symbol columns.

Usage

```
join_panels(panels)
```

Arguments

`panels`	List of wide panels (`data.frame` or `data.table`), each with columns `Date` + symbols.

Details

All panels are first aligned to the intersection of their `Date` values. Symbol names must be unique across panels; if the same symbol appears in multiple inputs, an error is raised.

Value

A `data.table` with `Date` plus the union of all symbol columns, restricted to common dates.

limit_positions *Limit per-date selections to top-K (legacy API)*

Description

Legacy selector used across examples/vignettes. Works on a WIDE table (`Date` + one column per symbol) and returns a WIDE logical mask with at most `max_positions` TRUE values per row.

Usage

```
limit_positions(
  selection_df,
  max_positions,
  ranking_signal = NULL,
  verbose = FALSE
)
```

Arguments

selection_df	data.frame/data.table with columns: Date, then one column per symbol; logical (preferred) or numeric (non-NA/ >0 means selected).
max_positions	positive integer, maximum selections to keep per row.
ranking_signal	optional data.frame/data.table, same dims & columns as selection_df, numeric scores used to rank within the selected set.
verbose	logical; if TRUE, prints minor diagnostics. Default FALSE.

Details

If ranking_signal is supplied, it must have **the same shape** and columns as selection_df; the function keeps the top-K (largest) scores among the *currently selected* columns in that row. If ranking_signal is NULL, it falls back to the values in selection_df (if numeric), otherwise keeps the first K selected symbols in column order (deterministic).

Value

A data.table with the same columns as selection_df, where symbol columns are logical and at most max_positions are TRUE in each row.

Examples

```
data(sample_prices_weekly)
mom12 <- PortfolioTesteR::calc_momentum(sample_prices_weekly, 12)  # WIDE numeric
sel10 <- PortfolioTesteR::filter_top_n(mom12, 10)                # WIDE logical/numeric
# Ensure logical mask
syms <- setdiff(names(sel10), "Date")
sel_mask <- data.table::as.data.table(sel10)
sel_mask[, (syms) := lapply(.SD, function(z) as.logical(as.integer(z))), .SDcols = syms]

# Keep at most 10 per date using momentum as the ranking signal
lim <- limit_positions(selection_df = sel_mask, max_positions = 10L,
                       ranking_signal = mom12, verbose = FALSE)
head(lim)
```

---

list_examples *List available example scripts*

Description

Shows all example scripts included with the PortfolioTesteR package. These examples demonstrate various strategy patterns and library functions.

Usage

```
list_examples()
```

Value

Character vector of example filenames

Examples

```
# See available examples
list_examples()

# Run a specific example
# run_example("example_momentum_basic.R")
```

`load_mixed_symbols` *Load Mixed Symbols Including VIX*

Description

Handles loading regular stocks and VIX together, with VIX loaded separately without auto-update to avoid issues.

Usage

```
load_mixed_symbols(
  db_path,
  symbols,
  start_date,
  end_date,
  frequency = "weekly",
  use_adjusted = TRUE
)
```

Arguments

`db_path`	Path to SQLite database
`symbols`	Character vector including regular stocks and optionally "VIX"
`start_date`	Start date for data
`end_date`	End date for data
`frequency`	Data frequency (default: "weekly")
`use_adjusted`	Use adjusted prices (default: TRUE)

Value

data.table with all symbols properly loaded

Examples

```
mixed <- load_mixed_symbols(
  db_path  = "sp500.db",
  symbols  = c("AAPL", "MSFT", "VIX"),
  start_date = "2020-01-01",
  end_date   = "2020-12-31",
  frequency  = "weekly"
)
head(mixed)
```

`make_labels` *Make future-return labels aligned to the decision date*

Description

Compute forward returns over a fixed horizon and align them to the decision date.

Usage

```
make_labels(prices, horizon = 4L, type = c("log", "simple", "sign"))
```

Arguments

`prices`	Wide price panel (`Date` + symbols).
`horizon`	Integer `>= 1`, number of forward steps for the label.
`type`	Character, one of `"log"`, `"simple"`, `"sign"`.

Details

For each date t, the label is computed from prices at t and $t + h$.

- `type = "simple"`: $p_{t+h}/p_t - 1$
- `type = "log"`: $\log(p_{t+h}) - \log(p_t)$
- `type = "sign"`: `sign(simple return)`

Trailing dates that do not have `horizon` steps ahead are set to `NA`.

Value

A `data.table` with `Date` + symbols containing the labels.

manual_adapter *Adapter for User-Provided Data*

Description

Simple adapter for when users provide their own data frame. Ensures proper Date formatting and sorting.

Usage

```
manual_adapter(data, date_col = "Date")
```

Arguments

data	User-provided data frame
date_col	Name of date column (default: "Date")

Value

Standardized data.table

Examples

```
# Use your own data frame
data("sample_prices_weekly")
my_prices <- manual_adapter(sample_prices_weekly)
```

membership_stability *Membership stability across dates*

Description

Membership stability across dates

Usage

```
membership_stability(mask)
```

Arguments

mask	logical mask panel from selection.

Value

data.table with `Date, stability` (Jaccard vs previous date).

metric_sharpe *Calculate Sharpe Ratio with Frequency Detection*

Description

Calculate Sharpe Ratio with Frequency Detection

Usage

```
metric_sharpe(bt)
```

Arguments

bt	Backtest result object with $returns and (optionally) $dates

Value

Annualized Sharpe ratio

ml_add_interactions *Add interaction panels to a feature list*

Description

Builds new panels from existing ones via elementwise operations. Specification accepts either a shorthand `list(new = c("A","B"))` (defaults to product), or a structured form `list(new = list(panels=c("A","B","C")`

Usage

```
ml_add_interactions(features, interactions)
```

Arguments

features	List of existing panels (wide data frames with Date).
interactions	Named list describing interactions to add.

Value

The input features list with additional interaction panels.

Examples

```
## Not run:
X2 <- ml_add_interactions(X, list(mom_vol = c("mom12","vol")))

## End(Not run)
```

`ml_backtest` *One-call backtest wrapper (tabular features)*

Description

One-call backtest wrapper (tabular features)

Usage

```
ml_backtest(
  features_list,
  labels,
  fit_fn,
  predict_fn,
  schedule = list(is = 156L, oos = 4L, step = 4L),
  group = c("pooled", "per_symbol", "per_group"),
  transform = c("none", "zscore", "rank"),
  selection = list(top_k = 15L, max_per_group = NULL),
  group_map = NULL,
  weighting = list(method = "softmax", temperature = 12, floor = 0),
  caps = list(max_per_symbol = 0.1, max_per_group = NULL),
  turnover = NULL,
  prices,
  initial_capital = 1e+05,
  name = "ML backtest"
)
```

Arguments

`features_list`	list of wide panels (each: `Date` + symbols).
`labels`	Wide label panel (Date + symbols).
`fit_fn`	function `(X, y) -> model` trained on in-sample stacked rows.
`predict_fn`	function `(model, Xnew) -> numeric` scores.
`schedule`	list with elements `is`, `oos`, `step`.
`group`	one of `"pooled"`, `"per_symbol"`, `"per_group"`.
`transform`	`"none"`, `"zscore"`, `"rank"` for per-date score transform.
`selection`	list: `top_k`, `max_per_group` (optional).
`group_map`	optional `data.frame(Symbol, Group)` if `group = "per_group"`.
`weighting`	list: `method`, `temperature`, `floor`.
`caps`	list: `max_per_symbol`, optionally `max_per_group`.
`turnover`	Optional turnover cap settings (currently advisory/unused).
`prices`	price panel used by the backtester (`Date` + symbols).
`initial_capital`	starting capital.
`name`	string for the backtest result.

Value

list: scores, mask, weights, backtest.

Examples

```
data(sample_prices_weekly); data(sample_prices_daily)
mom <- panel_lag(calc_momentum(sample_prices_weekly, 12), 1)
vol <- panel_lag(align_to_timeframe(
  calc_rolling_volatility(sample_prices_daily, 20),
  sample_prices_weekly$Date, "forward_fill"), 1)
Y <- make_labels(sample_prices_weekly, 4, "log")
fit_lm  <- function(X,y){ Xc <- cbind(1,X); list(coef=stats::lm.fit(Xc,y)$coefficients) }
pred_lm <- function(m,X){ as.numeric(cbind(1,X) %*% m$coef) }
res <- ml_backtest(list(mom=mom, vol=vol), Y, fit_lm, pred_lm,
                   schedule = list(is=52,oos=4,step=4),
                   transform = "zscore",
                   selection = list(top_k=10),
                   weighting = list(method="softmax", temperature=12),
                   caps = list(max_per_symbol=0.10),
                   prices = sample_prices_weekly, initial_capital = 1e5)
res$backtest
```

---

ml_backtest_multi *Run multi-horizon ML backtests (pooled or sector-neutral)*

Description

Convenience wrapper around ml_backtest() that repeats the same specification across multiple horizons, returning a named list of backtest objects keyed as "H1w", "H4w", "H13w", etc.

Usage

```
ml_backtest_multi(
  features_list,
  prices_weekly,
  horizons,
  fit_fn,
  predict_fn,
  schedule,
  transform = "zscore",
  selection = list(top_k = 20L),
  weighting = list(method = "softmax", temperature = 12),
  caps = list(max_per_symbol = 0.1),
  group_mode = c("pooled", "per_group"),
  group_map = NULL,
  initial_capital = 1e+05,
  name_prefix = "",
```

```
  seed = NULL,
  ...
)
```

Arguments

features_list	Named list of data.tables with factor scores (each with a Date column and one column per symbol). Typically from ml_prepare_features().
prices_weekly	Wide price table (weekly) with Date + one column per symbol (adjusted prices). Used both to create labels and run the backtest.
horizons	Integer vector of horizons in *weeks* (e.g., c(1L,4L,13L)).
fit_fn, predict_fn	Model fit/predict closures as returned by ml_make_model().
schedule	Walk-forward schedule list with elements is, oos, step.
transform	Feature transform (default "zscore"). Passed to ml_backtest().
selection	List describing selection rules (e.g., list(top_k=20L, max_per_group=3L)).
weighting	List describing weighting rules (e.g., list(method="softmax", temperature=12)).
caps	List with exposure caps (e.g., list(max_per_symbol=0.10, max_per_group=NULL)).
group_mode	"pooled" or "per_group". If "per_group", you must pass group_map.
group_map	A two-column table with columns Symbol and Group defining the grouping (e.g., sectors) for "per_group" mode.
initial_capital	Numeric. Starting capital for the backtest (default 1e5).
name_prefix	Optional string prefixed to each backtest title.
seed	Optional integer. If provided, the same seed is set before each horizon's backtest call to ensure deterministic tie-breaks.
...	Additional arguments forwarded to ml_backtest() (kept for future extensibility; no effect if unused).

Details

This function does **not** change core behavior; it only removes boilerplate when running identical specs across horizons and (optionally) grouping regimes. It preserves all defaults you pass for selection, weighting, transforms, caps, and schedule.

Value

A named list of backtest objects (as returned by ml_backtest()), with names like "H1w", "H4w", ...

Examples

```
library(PortfolioTesteR)
data(sample_prices_weekly, package = "PortfolioTesteR")

# Minimal features for the example
X <- ml_prepare_features(
  prices_weekly = sample_prices_weekly,
  include = c("mom12","mom26")
)

# Simple deterministic model
model <- ml_make_model("linear")
sched <- list(is = 156L, oos = 4L, step = 4L)

set.seed(42)
bt_list <- ml_backtest_multi(
  features_list = X,
  prices_weekly = sample_prices_weekly,
  horizons = c(1L, 4L),
  fit_fn = model$fit,
  predict_fn = model$predict,
  schedule = sched,
  selection = list(top_k = 5L),
  weighting = list(method = "softmax", temperature = 12),
  caps = list(max_per_symbol = 0.10),
  group_mode = "pooled",
  name_prefix = "Demo ",
  seed = 42
)

names(bt_list)   # "H1w" "H4w"
```

`ml_backtest_seq` *One-call backtest wrapper (sequence features)*

Description

One-call backtest wrapper (sequence features)

Usage

```
ml_backtest_seq(
  features_list,
  labels,
  steps = 26L,
  horizon = 4L,
  fit_fn,
```

```
  predict_fn,
  schedule = list(is = 156L, oos = 4L, step = 4L),
  group = c("pooled", "per_symbol", "per_group"),
  group_map = NULL,
  normalize = c("none", "zscore", "minmax"),
  selection = list(top_k = 15L, max_per_group = NULL),
  weighting = list(method = "softmax", temperature = 12, floor = 0),
  caps = list(max_per_symbol = 0.1, max_per_group = NULL),
  prices,
  initial_capital = 1e+05,
  name = "SEQ backtest"
)
```

Arguments

features_list	list of panels to be stacked over steps history.
labels	future-return panel aligned to the features.
steps	int; lookback length (e.g., 26).
horizon	int; label horizon (e.g., 4).
fit_fn	function (X, y) -> model trained on in-sample stacked rows.
predict_fn	function (model, Xnew) -> numeric scores.
schedule	list with elements is, oos, step.
group	one of "pooled", "per_symbol", "per_group".
group_map	optional data.frame(Symbol, Group) if group = "per_group".
normalize	"none", "zscore", or "minmax" applied using IS data only.
selection	list: top_k, max_per_group (optional).
weighting	list: method, temperature, floor.
caps	list: max_per_symbol, optionally max_per_group.
prices	price panel used by the backtester (Date + symbols).
initial_capital	starting capital.
name	string for the backtest result.

Value

list: scores, mask, weights, backtest.

Examples

```
# as above, but with steps/horizon and normalize
```

ml_ic_series_on_scores *Rank-IC series computed on score (rebalance) dates*

Description

Wrapper around `ic_series()` that first filters scores to formation dates using an internal filter that keeps rows where at least one score is finite.

Usage

```
ml_ic_series_on_scores(scores_dt, labels_dt, method = "spearman")
```

Arguments

`scores_dt`	Scores table (wide).
`labels_dt`	Labels table (wide).
`method`	Correlation method; `"spearman"` (default) or `"pearson"`.

Value

A data frame/data.table with `Date` and IC values.

Examples

```
## Not run:
ic <- ml_ic_series_on_scores(res_xgb$scores, Y, method = "spearman")

## End(Not run)
```

ml_make_ensemble *NA-tolerant ensemble blender (row-wise)*

Description

Creates an equal- or user-weighted blend of multiple model objects produced by `ml_make_model()`. The returned `fit` trains each component; `predict` combines component predictions with an NA-safe weighted average.

Usage

```
ml_make_ensemble(..., weights = NULL)
```

Arguments

...	Two or more model objects each with $fit/$predict.
weights	Optional numeric vector of blend weights (recycled).

Value

A list with $fit and $predict closures for the ensemble.

Examples

```
## Not run:
ens <- ml_make_ensemble(ml_make_model("ridge"),
                        ml_make_model("rf"),
                        ml_make_model("xgboost"))

## End(Not run)
```

---

ml_make_model *Model factory for tabular cross-sectional learners*

Description

Returns a pair of closures fit(X,y) / predict(model, X) implementing a chosen learner. Implementations are NA-aware and conservative: glmnet ridge drops rows with any non-finite input; ranger and xgboost keep NA in X as missing; the linear baseline uses lm.fit.

Usage

```
ml_make_model(
  type = c("ridge", "rf", "xgboost", "linear"),
  params = list(),
  nrounds = 200L,
  ...
)
```

Arguments

type	One of "ridge", "rf", "xgboost", "linear".
params	List of model parameters (passed to backend; used by xgboost).
nrounds	Integer boosting rounds (xgboost).
...	Additional arguments forwarded to the backend.

Details

Optional dependencies: glmnet (ridge), ranger (rf), xgboost (xgboost). If a backend is not available, use "linear" or install the package.

Value

A list with functions `fit` and `predict`.

Examples

```
## Not run:
ridge <- ml_make_model("ridge")
m <- ridge$fit(X_is, y_is)
s <- ridge$predict(m, X_oos)

## End(Not run)
```

`ml_make_seq_model` *Deterministic sequence model factory (GRU/LSTM/CNN1D with linear fallback)*

Description

Returns `fit/predict` closures for sequence models that consume flattened tabular inputs (n × (steps × p)) and internally reshape to (n, steps, p). If Keras/TensorFlow is unavailable, falls back to a linear baseline so examples remain runnable on CPU-only machines.

Usage

```
ml_make_seq_model(
  type = c("linear", "gru", "lstm", "cnn1d"),
  steps = 26L,
  units = 16L,
  dense = NULL,
  dropout = 0.1,
  epochs = 12L,
  batch_size = 128L,
  lr = 0.01,
  patience = 2L,
  seed = 123L,
  deterministic = TRUE,
  pred_batch_size = 256L
)
```

Arguments

`type`	One of `"linear"`, `"gru"`, `"lstm"`, `"cnn1d"`.
`steps`	Integer sequence length (e.g., 26 for 6 months of weeks).
`units`	Hidden units for GRU/LSTM or filters for CNN1D.
`dense`	Optional integer vector of additional dense layers.
`dropout`	Dropout rate for recurrent/CNN core.

epochs, batch_size
: Training settings.

lr
: Learning rate.

patience
: Early-stopping patience.

seed
: Integer seed.

deterministic
: Logical; set determinism knobs when TRUE.

pred_batch_size
: Fixed batch size used at prediction time.

Details

Determinism knobs: fixed seeds, TF_DETERMINISTIC_OPS=1, no shuffle, workers=1, and a fixed pred_batch_size to minimise retracing.

Optional dependencies: keras and tensorflow. When not available, the factory returns the linear fallback.

Value

A list with $fit and $predict closures.

Examples

```
## Not run:
seq_gru <- ml_make_seq_model("gru", steps = 26, units = 16, epochs = 12)

## End(Not run)
```

---

ml_panel_op *Panel-safe binary operation on aligned wide panels*

Description

Applies an elementwise binary operator to two date-aligned wide panels (first column Date, other columns are symbols), preserving the Date column and a consistent symbol set. Supports intersection or union of column sets; missing entries introduced by how="union" are filled.

Usage

```
ml_panel_op(A, B, op = `*`, how = c("intersect", "union"), fill = NA_real_)
```

Arguments

A, B
: Data frames with a Date column and one column per symbol.

op
: Binary function to apply elementwise (e.g., *, /, +).

how
: Character; "intersect" (default) or "union" for the set of symbol columns to operate on.

fill
: Numeric; value used to fill gaps when how="union".

Value

A data.frame with `Date` and the operated symbol columns.

Examples

```
## Not run:
out <- ml_panel_op(mom12_panel, vol_panel, op = `*`)

## End(Not run)
```

`ml_panel_reduce` *Reduce multiple panels with a binary operator*

Description

Folds a list of panels using `ml_panel_op()` across a set of named panels.

Usage

```
ml_panel_reduce(features, panels, op = `*`, how = "intersect", fill = NA_real_)
```

Arguments

`features`	List of panels (each a wide data frame with `Date`).
`panels`	Character vector of names in `features` to reduce (length ≥ 2).
`op`	Binary function to apply elementwise.
`how`	Column-set policy passed to `ml_panel_op()`.
`fill`	Fill value for `how="union"`.

Value

A data.frame panel (wide) with the reduced result.

Examples

```
## Not run:
# product of three panels
prod_panel <- ml_panel_reduce(X, c("mom12","vol","rsi14"), op = `*`)

## End(Not run)
```

ml_plot_ic_roll *Rolling rank-IC plot (rebalance dates; leakage-safe)*

Description

Computes the IC time series via `ml_ic_series_on_scores()` and plots the rolling mean IC over a specified window. Returns the rolling statistics invisibly for further inspection.

Usage

```
ml_plot_ic_roll(scores_dt, labels_dt, window = 26L, method = "spearman")
```

Arguments

`scores_dt`	Scores (wide).
`labels_dt`	Labels (wide).
`window`	Integer window length (default 26).
`method`	Correlation method; `"spearman"` (default) or `"pearson"`.

Value

(Invisibly) a data frame with `Date`, `roll_mean`, `roll_sd`, `roll_ICIR`.

Examples

```
## Not run:
ris <- ml_plot_ic_roll(res_xgb$scores, Y, window = 8L)

## End(Not run)
```

ml_prepare_features *Prepare tabular features (weekly + aligned daily volatility)*

Description

Constructs a minimal, leakage-safe set of tabular features commonly used in cross-sectional ML: weekly momentum (12/26/52), RSI(14), distance from 20-week MA, and daily rolling volatility aligned to weekly dates (tokens `vol{N}d_walign`, e.g., `"vol20d_walign"`).

Usage

```
ml_prepare_features(
  prices_weekly,
  prices_daily = NULL,
  include = c("mom12", "mom26", "mom52", "vol20d_walign", "dist20", "rsi14"),
  interactions = NULL
)
```

Arguments

prices_weekly	Wide panel with Date and symbol columns (weekly).
prices_daily	Optional wide panel (daily) if vol*d_walign are included.
include	Character vector of feature tokens to include.
interactions	Optional named list passed to ml_add_interactions().

Details

All outputs are **lagged by one period** to avoid look-ahead in backtests.

Value

A named list of panels (each a wide data.frame with Date).

Examples

```
## Not run:
X <- ml_prepare_features(sample_prices_weekly, sample_prices_daily,
                         include = c("mom12","vol20d_walign","rsi14"))

## End(Not run)
```

---

panel_lag *Lag each symbol column by k steps*

Description

Given a wide panel with a Date column followed by symbol columns, returns the same shape with each symbol column lagged by k periods. The Date column is preserved; leading values introduced by the lag are NA_real_.

Usage

```
panel_lag(df, k = 1L)
```

Arguments

df	data.frame or data.table with columns Date then symbols.
k	Integer lag (>= 1).

Value

A data.table with the same columns as df, lagged by k.

Examples

```
x <- data.frame(Date = as.Date("2020-01-01") + 0:2, A = 1:3, B = 11:13)
panel_lag(x, 1L)
```

panel_returns_simple *Panel simple returns from prices*

Description

Converts a wide price panel (Date + symbols) into arithmetic simple returns at the same cadence, dropping the first row per symbol.

Usage

```
panel_returns_simple(prices)
```

Arguments

prices	A data frame or data.table with columns Date and one column per symbol containing adjusted prices at a common frequency (daily, weekly, monthly).

Value

A data frame with Date and one column per symbol containing simple returns $R_t = P_t/P_{t-1} - 1$.

Examples

```
data(sample_prices_weekly)
rets <- panel_returns_simple(sample_prices_weekly)
head(rets)
```

perf_metrics *Portfolio performance metrics*

Description

Portfolio performance metrics

Usage

```
perf_metrics(ret_dt, freq = 52)
```

Arguments

ret_dt	data.table/data.frame with columns Date and ret.
freq	Integer periods-per-year for annualization (e.g., 52 or 252).

Value

list with total_return, ann_return, ann_vol, sharpe, max_drawdown.

plot.backtest_result *Plot Backtest Results*

Description

S3 plot method for visualizing backtest performance.

Usage

```
## S3 method for class 'backtest_result'
plot(x, type = "performance", ...)
```

Arguments

x	backtest_result object
type	Plot type: "performance", "drawdown", "weights", or "all"
...	Additional plotting parameters

Value

NULL (creates plot)

Examples

```
data("sample_prices_weekly")
mom <- calc_momentum(sample_prices_weekly, lookback = 12)
sel <- filter_top_n(mom, n = 10)
W   <- weight_equally(sel)
res <- run_backtest(sample_prices_weekly, W)
if (interactive()) plot(res, type = "performance")
```

plot.param_grid_result

Plot Parameter Grid Results (1D/2D/3D and Facets)

Description

Generic plotter for objects returned by `run_param_grid()`. Supported types:

- "line": 1D metric vs one parameter.
- "heatmap": 2D heatmap over two parameters.
- "surface": 3D surface (persp) over two parameters.
- "slices": 2D heatmaps faceted by a third parameter.
- "surface_slices": 3D surfaces faceted by a third parameter.
- "auto": chosen from the above based on the number of parameters.

Usage

```
## S3 method for class 'param_grid_result'
plot(
  x,
  y = NULL,
  type = c("auto", "line", "heatmap", "surface", "slices", "surface_slices"),
  metric = NULL,
  params = NULL,
  fixed = list(),
  agg = c("mean", "median", "max"),
  na.rm = TRUE,
  palette = NULL,
  zlim = NULL,
  clip = c(0.02, 0.98),
  main = NULL,
  sub = NULL,
  xlab = NULL,
  ylab = NULL,
  ...
)
```

Arguments

x	A param_grid_result.
y	Ignored.
type	One of "auto","line","heatmap","surface","slices","surface_slices".
metric	Column name to plot (defaults to "score" if present).
params	Character vector of parameter columns to use for axes/facets. If NULL, uses all parameter columns detected in x$all_results.
fixed	Optional named list of parameter values to condition on. Rows not matching are dropped before plotting.
agg	Aggregation when multiple rows map to a cell: "mean","median", or "max".
na.rm	Logical; drop NA metric rows before plotting.
palette	Optional color vector used for heatmaps/surfaces. Defaults to grDevices::hcl.colors(..., "YlOrRd", rev = TRUE).
zlim	Optional two-element numeric range for color scaling.
clip	Two quantiles used to winsorize z-limits for stable coloring.
main, sub, xlab, ylab	Base plotting annotations.
...	Additional options depending on type, e.g.: legend, label_cells, contour, mark_best, theta, phi, shade, expand, ticktype, shared_zlim, facet, facet_values, ncol, debug, impute_for_surface.

Value

An invisible list describing the plot (kind/params/zlim/etc.).

See Also

run_param_grid(), print.param_grid_result()

Examples

```
data(sample_prices_weekly)
b <- function(prices, params, ...) {
  weight_equally(filter_top_n(calc_momentum(prices, params$lookback), 10))
}
opt <- run_param_grid(
  prices  = sample_prices_weekly,
  grid    = list(lookback = c(8, 12, 26)),
  builder = b
)
plot(opt, type = "line", params = "lookback")
```

---

plot.performance_analysis

Plot Performance Analysis Results

Description

S3 method for visualizing performance metrics. Supports multiple plot types including summary dashboard, return distributions, risk evolution, and rolling statistics.

Usage

```
## S3 method for class 'performance_analysis'
plot(x, type = "summary", ...)
```

Arguments

x	performance_analysis object
type	Plot type: "summary", "returns", "risk", "drawdown"
...	Additional plotting parameters

Value

NULL (creates plot)

Examples

```
data("sample_prices_weekly")
data("sample_prices_daily")
syms_all <- intersect(names(sample_prices_weekly)[-1], names(sample_prices_daily)[-1])
syms <- syms_all[seq_len(min(3L, length(syms_all)))]
P <- sample_prices_weekly[, c("Date", syms), with = FALSE]
D <- sample_prices_daily[,  c("Date", syms), with = FALSE]
mom <- calc_momentum(P, lookback = 12)
sel <- filter_top_n(mom, n = 3)
W   <- weight_equally(sel)
res <- run_backtest(P, W)
perf <- analyze_performance(res, D, benchmark_symbol = syms[1])
if (interactive()) {
  plot(perf, type = "summary")
}
```

---

plot.wf_optimization_result

Plot Walk-Forward Results

Description

Visual diagnostics for a `wf_optimization_result` returned by `run_walk_forward()`. Supported types:

- "parameters": best parameter values chosen per window.
- "is_oos": in-sample vs out-of-sample scores by window.
- "equity": stitched out-of-sample equity curve.
- "drawdown": drawdown of the stitched OOS curve.
- "windows": per-window bar/line chart of an OOS metric (see `metric`).
- "stability": summary of parameter stability.
- "distributions": distributions of IS/OOS scores across windows.

Usage

```
## S3 method for class 'wf_optimization_result'
plot(
  x,
  y = NULL,
  type = c("parameters", "is_oos", "equity", "drawdown", "windows", "stability",
    "distributions"),
  param = NULL,
  metric = NULL,
  main = NULL,
  sub = NULL,
  xlab = NULL,
```

```
  ylab = NULL,
  ...
)
```

Arguments

x	A wf_optimization_result from run_walk_forward().
y	Ignored.
type	One of "parameters","is_oos","equity","drawdown", "windows","stability","distributions".
param	Character vector of parameter names to include for "parameters"/"stability"/"distributions". If NULL, uses all.
metric	Character; column to plot for "is_oos" or "windows" (e.g., "OOS_Return" or "OOS_Score"). Ignored for other types.
main, sub, xlab, ylab	Base plotting annotations.
...	Additional plot options (type-specific).

Value

Invisibly, a small list describing the plot.

See Also

run_walk_forward(), wf_report(), print.wf_optimization_result()

Examples

```
data(sample_prices_weekly)
b <- function(prices, params, ...) {
  weight_equally(filter_top_n(calc_momentum(prices, params$lookback),
                              params$top_n))
}
wf <- run_walk_forward(
  prices = sample_prices_weekly,
  grid   = list(lookback = c(8, 12, 26), top_n = c(5, 10)),
  builder = b,
  is_periods = 52, oos_periods = 26, step = 26
)
plot(wf, type = "parameters")
plot(wf, type = "is_oos", metric = "OOS_Score")
```

`portfolio_returns` *Portfolio returns from weights and prices (CASH-aware)*

Description

Computes the portfolio simple return series by applying (lagged) portfolio weights to next-period asset returns, optionally net of proportional costs.

Usage

```
portfolio_returns(weights, prices, cost_bps = 0)
```

Arguments

`weights`	A data.frame/data.table of portfolio weights on rebalance dates: first column `Date`, remaining columns one per symbol (numeric weights). Weights decided at $t-1$ are applied to returns over t.
`prices`	A data.frame/data.table of adjusted prices at the same cadence: first column `Date`, remaining columns one per symbol.
`cost_bps`	One-way proportional cost per side in basis points (e.g., `10` for 10 bps). Default `0`. If `> 0` and your package exposes a turnover helper, it will be used; otherwise costs are ignored with a warning.

Details

CASH support: if `weights` contains a column named `"CASH"` (case-insensitive) but `prices` has no matching column, a synthetic flat price series is added internally (price = 1 $\Rightarrow$ return = 0). In that case the function does **not** re-normalise the non-CASH weights; the row is treated as a complete budget (symbols + CASH = 1).

The function carries forward the latest available weights to each return row via the usual one-period decision lag. Transaction cost handling is conservative: if a turnover helper is not available, costs are skipped.

Value

A `data.table` with columns `Date` and `ret` (portfolio simple return).

See Also

PortfolioTesteR::panel_returns_simple

Examples

```
data(sample_prices_weekly)
mom12 <- PortfolioTesteR::calc_momentum(sample_prices_weekly, 12)
sel10 <- PortfolioTesteR::filter_top_n(mom12, 10)
w_eq  <- PortfolioTesteR::weight_equally(sel10)

pr <- portfolio_returns(w_eq, sample_prices_weekly, cost_bps = 0)
head(pr)
```

---

print.backtest_result *Print Backtest Results*

Description

S3 print method for backtest results. Shows key performance metrics.

Usage

```
## S3 method for class 'backtest_result'
print(x, ...)
```

Arguments

x	backtest_result object
...	Additional arguments (unused)

Value

Invisible copy of x

Examples

```
data("sample_prices_weekly")
mom <- calc_momentum(sample_prices_weekly, lookback = 12)
sel <- filter_top_n(mom, n = 10)
W   <- weight_equally(sel)
res <- run_backtest(sample_prices_weekly, W)
print(res)
```

print.param_grid_result

Print a param_grid_result

Description

Print a param_grid_result

Usage

```
## S3 method for class 'param_grid_result'
print(x, ...)
```

Arguments

x	A param_grid_result object returned by run_param_grid().
...	Additional arguments passed to methods (ignored).

Value

Invisibly returns x.

print.performance_analysis

Print Performance Analysis Results

Description

S3 method for printing performance analysis with key metrics including risk-adjusted returns, drawdown statistics, and benchmark comparison.

Usage

```
## S3 method for class 'performance_analysis'
print(x, ...)
```

Arguments

x	performance_analysis object
...	Additional arguments (unused)

Value

Invisible copy of x

Examples

```
data("sample_prices_weekly")
data("sample_prices_daily")
syms_all <- intersect(names(sample_prices_weekly)[-1], names(sample_prices_daily)[-1])
syms <- syms_all[seq_len(min(3L, length(syms_all)))]
P <- sample_prices_weekly[, c("Date", syms), with = FALSE]
D <- sample_prices_daily[,  c("Date", syms), with = FALSE]
mom <- calc_momentum(P, lookback = 12)
sel <- filter_top_n(mom, n = 3)
W   <- weight_equally(sel)
res <- run_backtest(P, W)
perf <- analyze_performance(res, D, benchmark_symbol = syms[1])
print(perf)  # or just: perf
```

---

print.wf_optimization_result *Print a wf_optimization_result*

Description

Print a wf_optimization_result

Usage

```
## S3 method for class 'wf_optimization_result'
print(x, ...)
```

Arguments

x	A `wf_optimization_result` object returned by `run_walk_forward()`.
...	Additional arguments passed to methods (ignored).

Value

Invisibly returns `x`.

---

pt_collect_results *Collect diagnostics from two* `ml_backtest_multi()` *runs*

Description

Builds a compact set of outputs—coverage, IC series, OOS-only rolling IC, performance tables (Full/Pre/Post), turnover, and a cost sweep—given two lists of backtests (pooled and per-group) produced by `ml_backtest_multi()`.

Usage

```
pt_collect_results(
  bt_pooled_list,
  bt_neutral_list,
  weekly_prices,
  horizons,
  split_date,
  cost_bps = c(5, 10, 15, 20, 25, 50),
  freq = 52,
  prices = NULL,
  ic_roll_window = 26L,
  mask_scores_to_decision_dates = TRUE,
  cost_model = function(turnover, bps) (bps/10000) * turnover
)
```

Arguments

bt_pooled_list	Named list of backtests (keys like H4w, H13w) produced with group_mode = "pooled".
bt_neutral_list	Named list of backtests (same keys) produced with group_mode = "per_group".
weekly_prices	Deprecated alias for prices; kept for backwards compatibility.
horizons	Integer vector of horizons (in weeks) expected in the lists.
split_date	Date used to split performance into Pre/Post.
cost_bps	Numeric vector of per-rebalance cost levels (in basis points) for the turnover-based cost sweep. Default c(5, 10, 15, 20, 25, 50).
freq	Integer frequency used by perf_metrics() (e.g., 52 for weekly).
prices	Optional price table (preferred). If NULL, weekly_prices is used.
ic_roll_window	Integer window length (weeks) for rolling IC on OOS decision dates. Default 26L.
mask_scores_to_decision_dates	Logical; if TRUE (default) scores are masked to OOS decision dates only (see scores_oos_only()).
cost_model	Function (turnover, bps) returning per-period cost to subtract from returns in the sweep. Default scales linearly with turnover.

Details

Both input lists must have identical horizon keys (paste0("H", h, "w")). Coverage and IC series are computed from stored scores; rolling IC is built on OOS decision dates only; performance is summarised for the full sample and Pre/Post relative to split_date; turnover is derived from realised sector-neutral weights; and a turnover-based cost sweep is evaluated on the sector-neutral run across cost_bps.

Value

A named list with one element per horizon, each containing:

- `bt_pooled`, `bt_neutral` — the input backtests;
- `coverage` — coverage by date for pooled/neutral;
- `ic_series` — raw IC series for pooled/neutral;
- `icroll_oos_26w` — rolling IC (OOS-only) for pooled/neutral;
- `masked_scores` — OOS-masked score tables for pooled/neutral;
- `perf_tables` — performance tables (Full/Pre/Post);
- `turnover_neutral` — turnover series for the sector-neutral run;
- `cost_sweep_neutral` — performance under gross/net across `cost_bps`.

See Also

`ml_backtest_multi()`, `scores_oos_only()`, `perf_metrics()`

Other Chapter3-helpers: `scores_oos_only()`

Examples

```
if (requireNamespace("PortfolioTesteR", quietly = TRUE)) {
  library(PortfolioTesteR)
  data(sample_prices_weekly, package = "PortfolioTesteR")
  # Simple feature
  mom12 <- PortfolioTesteR::calc_momentum(sample_prices_weekly, 12)
  feats <- list(mom12 = mom12)
  fit_first     <- function(X, y, ...) list()
  predict_first <- function(model, Xnew, ...) as.numeric(Xnew[[1]])
  sch <- list(is = 52L, oos = 26L, step = 26L)
  syms <- setdiff(names(sample_prices_weekly), "Date")
  gmap <- data.frame(Symbol = syms,
                     Group  = rep(c("G1","G2"), length.out = length(syms)))
  bt_pooled  <- ml_backtest_multi(feats, sample_prices_weekly, c(4L),
                                  fit_first, predict_first, sch,
                                  selection = list(top_k = 5L),
                                 weighting = list(method = "softmax", temperature = 12),
                                  caps = list(max_per_symbol = 0.10),
                                  group_mode = "pooled")
  bt_neutral <- ml_backtest_multi(feats, sample_prices_weekly, c(4L),
                                  fit_first, predict_first, sch,
                                  selection = list(top_k = 5L),
                                 weighting = list(method = "softmax", temperature = 12),
                                  caps = list(max_per_symbol = 0.10),
                                  group_mode = "per_group",
                                  group_map = gmap)
  out <- pt_collect_results(
    bt_pooled_list  = bt_pooled,
    bt_neutral_list = bt_neutral,
    prices          = sample_prices_weekly,
    horizons        = c(4L),
```

```
    split_date      = as.Date("2019-01-01"),
    cost_bps        = c(5, 15),
    freq            = 52,
    ic_roll_window  = 13L
  )
  names(out)
  str(out[["H4w"]]$perf_tables)
}
```

`rank_within_sector` *Rank Indicators Within Each Sector*

Description

Ranks stocks within their sector for sector-neutral strategies. Enables selecting best stocks from each sector regardless of sector performance. Optimized using matrix operations within groups.

Usage

```
rank_within_sector(
  indicator_df,
  sector_mapping,
  method = c("percentile", "rank", "z-score"),
  min_sector_size = 3
)
```

Arguments

`indicator_df`	Data frame with Date column and indicator values
`sector_mapping`	Data frame with `Symbol` and `Sector` columns.
`method`	"percentile" (0-100), "rank" (1-N), or "z-score"
`min_sector_size`	Minimum stocks per sector (default: 3)

Value

Data frame with within-sector ranks/scores

Examples

```
data("sample_prices_weekly")
data("sample_sp500_sectors")
momentum <- calc_momentum(sample_prices_weekly, 12)
sector_ranks <- rank_within_sector(momentum, sample_sp500_sectors)
```

rebalance_calendar *Rebalance calendar (rows with non-zero allocation)*

Description

Rebalance calendar (rows with non-zero allocation)

Usage

```
rebalance_calendar(weights)
```

Arguments

weights	Wide weight panel (Date + symbols).

Value

data.table with `Date, row` indices where a rebalance occurred.

roll_fit_predict *Rolling fit/predict for tabular features (pooled / per-symbol / per-group)*

Description

Rolling fit/predict for tabular features (pooled / per-symbol / per-group)

Usage

```
roll_fit_predict(
  features_list,
  label,
  fit_fn,
  predict_fn,
  is_periods = 156L,
  oos_periods = 4L,
  step = 4L,
  group = c("pooled", "per_symbol", "per_group"),
  group_map = NULL,
  na_action = c("omit", "zero")
)
```

Arguments

features_list	list of wide panels (each: Date + symbols).
label	wide panel of future returns (same symbol set as features).
fit_fn	function (X, y) -> model trained on in-sample stacked rows.
predict_fn	function (model, Xnew) -> numeric scores.
is_periods, oos_periods, step	ints; in-sample length, out-of-sample length, and step size for the rolling window.
group	one of "pooled", "per_symbol", "per_group".
group_map	optional data.frame(Symbol, Group) if group = "per_group".
na_action	"omit" or "zero" for feature NA handling.

Value

wide panel of scores: Date + symbols.

Examples

```
data(sample_prices_weekly); data(sample_prices_daily)
mom <- panel_lag(calc_momentum(sample_prices_weekly, 12), 1)
vol <- panel_lag(align_to_timeframe(
  calc_rolling_volatility(sample_prices_daily, 20),
  sample_prices_weekly$Date, "forward_fill"), 1)
Y <- make_labels(sample_prices_weekly, horizon = 4, type = "log")
fit_lm  <- function(X,y){ Xc <- cbind(1,X); list(coef=stats::lm.fit(Xc,y)$coefficients) }
pred_lm <- function(m,X){ as.numeric(cbind(1,X) %*% m$coef) }
S <- roll_fit_predict(list(mom=mom, vol=vol), Y, fit_lm, pred_lm, 52, 4, 4)
head(S)
```

---

roll_fit_predict_seq *Rolling fit/predict for sequence models (flattened steps-by-p features)*

Description

Rolling fit/predict for sequence models (flattened steps-by-p features)

Usage

```
roll_fit_predict_seq(
  features_list,
  labels,
  steps = 26L,
  horizon = 4L,
  fit_fn,
  predict_fn,
```

```
  is_periods = 156L,
  oos_periods = 4L,
  step = 4L,
  group = c("pooled", "per_symbol", "per_group"),
  group_map = NULL,
  normalize = c("none", "zscore", "minmax"),
  min_train_samples = 50L,
  na_action = c("omit", "zero")
)
```

Arguments

features_list	list of panels to be stacked over steps history.
labels	future-return panel aligned to the features.
steps	int; lookback length (e.g., 26).
horizon	int; label horizon (e.g., 4).
fit_fn	function (X, y) -> model trained on in-sample stacked rows.
predict_fn	function (model, Xnew) -> numeric scores.
is_periods, oos_periods, step	ints; in-sample length, out-of-sample length, and step size for the rolling window.
group	one of "pooled", "per_symbol", "per_group".
group_map	optional data.frame(Symbol, Group) if group = "per_group".
normalize	"none", "zscore", or "minmax" applied using IS data only.
min_train_samples	Optional minimum IS samples required to fit; if not met, skip fit.
na_action	"omit" or "zero" for feature NA handling.

Value

wide panel of scores.

Examples

```
data(sample_prices_weekly); data(sample_prices_daily)
mom <- panel_lag(calc_momentum(sample_prices_weekly, 12), 1)
vol <- panel_lag(align_to_timeframe(
  calc_rolling_volatility(sample_prices_daily, 20),
  sample_prices_weekly$Date, "forward_fill"), 1)
Y <- make_labels(sample_prices_weekly, horizon = 4, type = "log")
fit_lm  <- function(X,y){ Xc <- cbind(1,X); list(coef=stats::lm.fit(Xc,y)$coefficients) }
pred_lm <- function(m,X){ as.numeric(cbind(1,X) %*% m$coef) }
S <- roll_fit_predict_seq(list(mom=mom, vol=vol), Y,
                          steps = 26, horizon = 4,
                          fit_fn = fit_lm, predict_fn = pred_lm,
                          is_periods = 104, oos_periods = 4, step = 4)
head(S)
```

roll_ic_stats *Rolling IC mean, standard deviation, and ICIR*

Description

Compute rolling information coefficient (IC) statistics from a per-date IC series.

Usage

```
roll_ic_stats(ic_dt, window = 26L)
```

Arguments

ic_dt	Data frame/data.table produced by ic_series with columns Date and IC.
window	Integer window length for the rolling statistics.

Details

For each rolling window, compute the mean IC, the standard deviation of IC, and the information coefficient ratio (ICIR = mean / sd). Windows with fewer than two finite IC values yield NA for ICIR.

Value

A data.table with columns Date, IC_mean, IC_sd, and ICIR.

run_backtest *Run Portfolio Backtest*

Description

Main backtesting engine that simulates portfolio performance over time. Handles position tracking, transaction costs, and performance calculation.

Usage

```
run_backtest(
  prices,
  weights,
  initial_capital = 1e+05,
  name = "Strategy",
  verbose = FALSE,
  stop_loss = NULL,
  stop_monitoring_prices = NULL
)
```

Arguments

prices	Price data (data.frame with Date column)
weights	Weight matrix from weighting functions
initial_capital	Starting capital (default: 100000)
name	Strategy name for reporting
verbose	Print progress messages (default: FALSE)
stop_loss	Optional stop loss percentage as decimal
stop_monitoring_prices	Optional daily prices for stop monitoring

Value

backtest_result object with performance metrics

Examples

```
data("sample_prices_weekly")
momentum <- calc_momentum(sample_prices_weekly, lookback = 12)
selected <- filter_top_n(momentum, n = 10)
weights <- weight_equally(selected)
result <- run_backtest(sample_prices_weekly, weights)
```

---

run_example *Run an Example Script*

Description

Executes an example script bundled in the package `inst/examples/` folder.

Usage

```
run_example(example_name, echo = TRUE)
```

Arguments

example_name	Character scalar with the example filename (e.g. `"basic.R"`).
echo	Logical; print code as it runs (default `TRUE`).

Value

Invisibly returns `NULL`. Runs the example for its side effects.

Examples

```
# Example (requires a real file under inst/examples):
# run_example("basic.R")
```

`run_param_grid` *Run Parameter Grid Optimization (safe + ergonomic)*

Description

Run Parameter Grid Optimization (safe + ergonomic)

Usage

```
run_param_grid(
  prices,
  grid,
  builder,
  metric = NULL,
  name_prefix = "Strategy",
  verbose = FALSE,
  light_mode = TRUE,
  precompute_returns = TRUE,
  builder_args = list(),
  n_cores = 1,
  fixed = NULL
)
```

Arguments

`prices`	Data frame with Date + symbol columns
`grid`	Data frame (each row = a combo) OR a **named list** of vectors
`builder`	Function(prices, params, ...) -> weights (Date + symbols)
`metric`	Scoring function(backtest) -> numeric. Defaults to metric_sharpe.
`name_prefix`	String prefix for backtest names
`verbose`	Logical
`light_mode`	Logical: speed-ups in backtest
`precompute_returns`	Logical: precompute log-returns once (light_mode only)
`builder_args`	List of extra args forwarded to builder (e.g., caches)
`n_cores`	Integer (kept for API compatibility; ignored here)
`fixed`	Optional named list of **constant** parameters merged into every combo. If a name appears in both `grid` and `fixed`, the **fixed value wins** and that column is **pruned from the grid** (fewer duplicate combos; clearer counts).

Value

param_grid_result

`run_walk_forward` *Walk-Forward Optimization Analysis*

Description

Runs rolling IS/OOS optimization, reselects params each window, and backtests OOS performance (optionally with warmup tails).

Usage

```
run_walk_forward(
  prices,
  grid,
  builder,
  metric = NULL,
  is_periods = 52,
  oos_periods = 13,
  step = NULL,
  warmup_periods = 0,
  verbose = FALSE,
  light_mode = TRUE,
  precompute_all = TRUE,
  builder_args = list(),
  n_cores = 1
)
```

Arguments

`prices`	Data frame with Date column and symbol columns
`grid`	Data frame OR named list; each row/combination is a parameter set
`builder`	Function(prices, params, ...) -> weights data.frame (Date + assets)
`metric`	Function(backtest_result) -> scalar score (higher is better). Defaults to `metric_sharpe` if omitted/NULL.
`is_periods`	Integer, number of in-sample periods
`oos_periods`	Integer, number of out-of-sample periods
`step`	Integer, step size for rolling windows (default = oos_periods)
`warmup_periods`	Integer, warmup periods appended before each OOS
`verbose`	Logical, print progress
`light_mode`	Logical, passed to run_param_grid (kept for compatibility)
`precompute_all`	Logical, precompute indicators once and slice per window
`builder_args`	List, extra args passed to builder (e.g., indicator_cache)
`n_cores`	Integer (kept for API compatibility; ignored here)

Value

An object of class `wf_optimization_result`.

`safe_divide` *Safe Division with NA and Zero Handling*

Description

Performs division with automatic handling of NA values, zeros, and infinity. Returns 0 for division by zero and NA cases.

Usage

```
safe_divide(numerator, denominator)
```

Arguments

`numerator`	Numeric vector
`denominator`	Numeric vector

Value

Numeric vector with safe division results

Examples

```
safe_divide(c(10, 0, NA, 5), c(2, 0, 5, NA))  # Returns c(5, 0, 0, 0)
```

`sample_prices_daily` *Sample Daily Stock Prices*

Description

Daily closing prices for 20 stocks from 2017-2019. Contains the same symbols as sample_prices_weekly but at daily frequency for more granular analysis and performance calculations.

Usage

```
data(sample_prices_daily)
```

Format

A data.table with 754 rows and 21 columns:

Date Date object, trading date

AAPL Apple Inc. adjusted closing price

AMZN Amazon.com Inc. adjusted closing price

BA Boeing Co. adjusted closing price

BAC Bank of America Corp. adjusted closing price

... Additional stock symbols with adjusted closing prices

Source

Yahoo Finance historical data, adjusted for splits and dividends

Examples

```
data(sample_prices_daily)
head(sample_prices_daily)
# Get date range
range(sample_prices_daily$Date)
```

---

sample_prices_weekly *Sample Weekly Stock Prices*

Description

Weekly closing prices for 20 stocks from 2017-2019. Data includes major stocks from various sectors and is suitable for demonstrating backtesting and technical analysis functions.

Usage

```
data(sample_prices_weekly)
```

Format

A data.table with 158 rows and 21 columns:

Date Date object, weekly closing date (typically Friday)

AAPL Apple Inc. adjusted closing price

AMZN Amazon.com Inc. adjusted closing price

BA Boeing Co. adjusted closing price

BAC Bank of America Corp. adjusted closing price

... Additional stock symbols with adjusted closing prices

Source

Yahoo Finance historical data, adjusted for splits and dividends

Examples

```
data(sample_prices_weekly)
head(sample_prices_weekly)
# Calculate momentum
momentum <- calc_momentum(sample_prices_weekly, lookback = 12)
```

---

sample_sp500_sectors *S&P 500 Sector Mappings*

Description

Sector classifications for the stock symbols in the sample datasets. Note: ETFs (SPY, QQQ, etc.) are not included as they represent indices or sectors themselves rather than individual companies.

Usage

```
data(sample_sp500_sectors)
```

Format

A data.table with 18 rows and 2 columns:

Symbol Character, stock ticker symbol

Sector Character, GICS sector classification

Source

S&P 500 constituent data

Examples

```
data(sample_sp500_sectors)
head(sample_sp500_sectors)
# Count stocks per sector
table(sample_sp500_sectors$Sector)
```

scores_oos_only *Mask score tables to out-of-sample decision dates*

Description

Utility used in the chapter's diagnostics: keep scores only on dates when a portfolio decision was actually made (non-zero realised weights); set other dates to NA. Inputs are wide by symbol with a Date column.

Usage

```
scores_oos_only(scores_dt, weights_wide)
```

Arguments

scores_dt	Wide table of model scores with columns Date, SYM1, SYM2, ... (one column per symbol).
weights_wide	Wide table of realised portfolio weights with columns Date, SYM1, SYM2, ... (one column per symbol). Decision dates are inferred as rows where any symbol weight is non-zero.

Value

A copy of scores_dt where rows not matching decision dates are set to NA (except the Date column). If either input is empty, returns scores_dt[0].

See Also

pt_collect_results()

Other Chapter3-helpers: pt_collect_results()

Examples

```
# Toy example
dates <- as.Date("2020-01-01") + 7*(0:5)
scores <- data.frame(
  Date = dates,
  AAA = seq(0.1, 0.6, length.out = 6),
  BBB = rev(seq(0.1, 0.6, length.out = 6))
)
weights <- data.frame(
  Date = dates,
  AAA  = c(0, 0.1, 0, 0.2, 0, 0.15),
  BBB  = c(0, 0,   0, 0,   0, 0   )
)
scores_oos_only(scores, weights)
```

select_top_k_scores *Select top-K scores per date*

Description

Select top-K scores per date

Usage

```
select_top_k_scores(scores, k, ties = "first")
```

Arguments

scores	score panel.
k	integer; how many to keep per date.
ties	Ties method passed to base::rank (e.g., 'first','average').

Value

logical mask panel (Date + symbols) marking selected names.

select_top_k_scores_by_group *Select top-k symbols per group by score*

Description

For each date, choose the top k symbols **within each group** based on a score panel. Returns a logical selection panel aligned to the input.

Usage

```
select_top_k_scores_by_group(scores, k, group_map, max_per_group = 3L)
```

Arguments

scores	Wide score panel (Date + symbols).
k	Positive integer: number of symbols to select per group.
group_map	Named character vector or 2-column data.frame (symbol, group) mapping symbols to groups.
max_per_group	Integer cap per group (default 3L).

Details

- Group membership comes from `group_map` (symbol -> group).
- Selection is computed independently by group on each date.
- Ties follow the ordering implied by `order(..., method = "radix")`.

Value

Logical selection panel (`Date` + symbols) where `TRUE` marks selected symbols.

Examples

```
set.seed(42)
scores <- data.frame(
  Date = as.Date("2020-01-01") + 0:1,
  A = runif(2), B = runif(2), C = runif(2), D = runif(2), E = runif(2), F = runif(2)
)
gmap <- data.frame(Symbol = c("A","B","C","D","E","F"),
                   Group  = c("G1","G1","G2","G2","G3","G3"))
sel <- select_top_k_scores_by_group(scores, k = 4, group_map = gmap, max_per_group = 2)
head(sel)
```

---

sql_adapter *Load Price Data from SQL Database*

Description

Loads stock price data from SQLite database with automatic frequency conversion.

Usage

```
sql_adapter(
  db_path,
  symbols,
  start_date = NULL,
  end_date = NULL,
  auto_update = TRUE,
  frequency = "daily"
)
```

Arguments

`db_path`	Path to SQLite database file
`symbols`	Character vector of stock symbols to load
`start_date`	Start date (YYYY-MM-DD) or NULL
`end_date`	End date (YYYY-MM-DD) or NULL
`auto_update`	Auto-update database before loading (default: TRUE)
`frequency`	"daily", "weekly", or "monthly" (default: "daily")

Value

data.table with Date column and one column per symbol

Examples

```
prices <- sql_adapter(
  db_path   = "sp500.db",
  symbols   = c("AAPL", "MSFT"),
  start_date = "2020-01-01",
  end_date   = "2020-12-31",
  frequency  = "weekly"
)
head(prices)
```

---

sql_adapter_adjusted *Load Adjusted Price Data from SQL Database*

Description

Loads adjusted stock prices (for splits/dividends) from SQLite.

Usage

```
sql_adapter_adjusted(
  db_path,
  symbols,
  start_date = NULL,
  end_date = NULL,
  auto_update = FALSE,
  frequency = "daily",
  use_adjusted = TRUE
)
```

Arguments

db_path	Path to SQLite database file
symbols	Character vector of stock symbols to load
start_date	Start date (YYYY-MM-DD) or NULL
end_date	End date (YYYY-MM-DD) or NULL
auto_update	Auto-update database (default: FALSE)
frequency	"daily", "weekly", or "monthly" (default: "daily")
use_adjusted	Use adjusted prices if available (default: TRUE)

Value

data.table with Date column and adjusted prices per symbol

Examples

```
prices <- sql_adapter_adjusted(
  db_path   = "sp500.db",
  symbols   = c("AAPL", "MSFT"),
  start_date = "2020-01-01",
  end_date   = "2020-12-31",
  frequency  = "monthly"
)
head(prices)
```

---

summary.backtest_result

Summary method for backtest results

Description

Summary method for backtest results

Usage

```
## S3 method for class 'backtest_result'
summary(object, ...)
```

Arguments

object	A backtest_result object
...	Additional arguments (unused)

Value

Invisible copy of the object

`switch_weights` *Switch Between Weighting Schemes*

Description

Dynamically switches between two weighting schemes based on a signal. Enables tactical allocation changes.

Usage

```
switch_weights(weights_a, weights_b, use_b_condition, partial_blend = 1)
```

Arguments

`weights_a`	Primary weight matrix
`weights_b`	Alternative weight matrix
`use_b_condition`	Logical vector (TRUE = use weights_b)
`partial_blend`	Blend factor 0-1 (default: 1 = full switch)

Value

Combined weight matrix

Examples

```
data("sample_prices_weekly")
momentum <- calc_momentum(sample_prices_weekly, lookback = 12)
selected <- filter_top_n(momentum, n = 10)
weights_equal <- weight_equally(selected)
weights_signal <- weight_by_signal(selected, momentum)

# Create switching signal (example: use SPY momentum as regime indicator)
spy_momentum <- momentum$SPY
switch_signal <- as.numeric(spy_momentum > median(spy_momentum, na.rm = TRUE))
switch_signal[is.na(switch_signal)] <- 0

# Switch between strategies
final_weights <- switch_weights(weights_equal, weights_signal, switch_signal)
```

transform_scores *Per-date score transform (z-score or rank)*

Description

Per-date score transform (z-score or rank)

Usage

```
transform_scores(
  scores,
  method = c("zscore", "rank"),
  per_date = TRUE,
  robust = FALSE
)
```

Arguments

scores	wide panel Date + symbols.
method	"zscore" or "rank".
per_date	logical; currently must be TRUE.
robust	logical; robust z-score via median/MAD.

Value

panel of transformed scores.

tune_ml_backtest *Quick grid tuning for tabular pipeline*

Description

Quick grid tuning for tabular pipeline

Usage

```
tune_ml_backtest(
  features_list,
  labels,
  prices,
  fit_fn,
  predict_fn,
  schedule = list(is = 104L, oos = 4L, step = 4L),
 grid = list(top_k = c(10L, 15L), temperature = c(8, 12), method = c("softmax", "rank"),
    transform = c("zscore")),
```

```
  group = "pooled",
  selection_defaults = list(top_k = 15L, max_per_group = NULL),
  weighting_defaults = list(method = "softmax", temperature = 12, floor = 0),
  caps = list(max_per_symbol = 0.08),
  group_map = NULL,
  cost_bps = 0,
  freq = 52
)
```

Arguments

features_list	List of feature panels.
labels	Label panel.
prices	Price panel used for backtests (Date + symbols).
fit_fn, predict_fn	Model fit and predict functions.
schedule	List with elements is, oos, step.
grid	list of vectors: top_k, temperature, method, transform.
group	Grouping mode for roll_fit_predict ('pooled'/'per_symbol'/'per_group').
selection_defaults	Default selection settings (e.g., top_k).
weighting_defaults	Default weighting settings (e.g., method, temperature).
caps	Exposure caps (e.g., max_per_symbol/max_per_group).
group_map	Optional Symbol->Group mapping.
cost_bps	optional one-way cost in basis points for net performance.
freq	re-annualization frequency (e.g., 52).

Value

data.table with metrics per grid row.

---

turnover_by_date *Turnover by date*

Description

Turnover by date

Usage

```
turnover_by_date(weights)
```

Arguments

weights	weight panel.

Value

data.table with Date, turnover (0.5 * L1 change).

---

update_vix_in_db *Update VIX data in database*

Description

Update VIX data in database

Usage

```
update_vix_in_db(db_path, from_date = NULL)
```

Arguments

db_path	Path to SQLite database
from_date	Start date for update (NULL = auto-detect)

Value

Number of rows updated (invisible)

---

validate_data_format *Validate Data Format for Library Functions*

Description

Checks that data meets library requirements: proper Date column, at least one symbol, correct data types. Prints diagnostic info.

Usage

```
validate_data_format(data)
```

Arguments

data	Data frame to validate

Value

TRUE if valid, stops with error if not

Examples

```
data("sample_prices_weekly")
# Check if data is properly formatted
validate_data_format(sample_prices_weekly)
```

---

validate_group_map *Validate a symbol-to-group mapping*

Description

Normalizes and checks a symbol → group mapping for a given set of symbols. Accepts either a data.frame/data.table with columns `Symbol` and `Group`, or a named character vector `c(symbol = "group", ...)`. Errors if any requested symbol is missing or mapped more than once.

Usage

```
validate_group_map(symbols, group_map)
```

Arguments

`symbols`	Character vector of symbols to validate/keep.
`group_map`	Data frame/data.table with columns `Symbol`,`Group`, or a named character vector mapping `symbol -> group`.

Value

A two-column `data.frame` with columns `Symbol` and `Group` (one row per symbol), sorted by `Symbol`.

Examples

```
validate_group_map(
  c("A","B"),
  data.frame(Symbol = c("A","B"), Group = c("G1","G1"))
)
```

validate_no_leakage *Quick leakage guard: date alignment & NA expectations*

Description

Quick leakage guard: date alignment & NA expectations

Usage

```
validate_no_leakage(features, labels, verbose = TRUE)
```

Arguments

features	Wide feature panel with a Date column.
labels	Wide label panel (same Date index / symbols as features).
verbose	If TRUE, prints diagnostic messages.

Value

TRUE/FALSE (invisible), with messages if verbose = TRUE.

vol_target *Volatility targeting (row-wise) with optional down-only cap*

Description

Scales each row of a wide weight table (Date + symbols) so the estimated annualised portfolio volatility matches a target. Volatility is estimated from a rolling covariance of simple asset returns computed from prices.

Usage

```
vol_target(
  weights,
  prices,
  lookback = 26L,
  target_annual = 0.12,
  periods_per_year = 52L,
  cap = TRUE
)
```

Arguments

weights	data.frame/data.table with columns: Date, then one column per symbol.
prices	data.frame/data.table of adjusted prices at the same cadence as weights: first column Date, remaining columns one per symbol (matching weight symbols).
lookback	Integer window length (in periods) for the rolling covariance. Default 26.
target_annual	Annualised volatility target (e.g., 0.12 for 12%). Must be > 0.
periods_per_year	Number of periods per year used for annualisation (e.g., 52 for weekly).
cap	If TRUE (default), scale down only: exposure is reduced when the estimated vol exceeds the target, and untouched otherwise. In this down-only mode, the function adds/overwrites a CASH column equal to 1 - rowSums(pmax(scaled_weights, 0)) so that symbols + CASH is approximately 1. If FALSE, the scaler may be > 1 (leverage allowed) and no CASH is added.

Details

Weights decided at t-1 apply to returns over t.

The covariance at row i is computed from the last lookback rows of simple returns up to i (inclusive), estimated on the intersection of symbols present in both weights and prices. The row scaler is s_i = min(1, target_vol / est_vol_i) when cap = TRUE, and s_i = target_vol / est_vol_i when cap = FALSE, with safeguards for zero or non-finite variances.

Value

A data.table with the same Date and symbol columns as weights (plus CASH when cap = TRUE).

Examples

```
data(sample_prices_weekly)
mom12 <- PortfolioTesteR::calc_momentum(sample_prices_weekly, 12)
sel10 <- PortfolioTesteR::filter_top_n(mom12, 10)
w_eq  <- PortfolioTesteR::weight_equally(sel10)

w_vt  <- vol_target(w_eq, sample_prices_weekly, lookback = 26,
                    target_annual = 0.12, periods_per_year = 52, cap = TRUE)
head(w_vt)
```

---

weight_by_hrp *Hierarchical Risk Parity Weighting*

Description

Calculates portfolio weights using Hierarchical Risk Parity (HRP) methodology. HRP combines hierarchical clustering with risk-based allocation to create diversified portfolios that don't rely on unstable correlation matrix inversions.

Usage

```
weight_by_hrp(
  selected_df,
  prices_df,
  lookback_periods = 252,
  cluster_method = "ward.D2",
  distance_method = "euclidean",
  min_periods = 60,
  use_correlation = FALSE
)
```

Arguments

selected_df	Binary selection matrix (data.frame with Date column)
prices_df	Price data for covariance calculation (typically daily) Returns are calculated internally from prices
lookback_periods	Number of periods for covariance estimation (default: 252)
cluster_method	Clustering linkage method (default: "ward.D2")
distance_method	Distance measure for clustering (default: "euclidean")
min_periods	Minimum periods required for calculation (default: 60)
use_correlation	If TRUE, cluster on correlation instead of covariance

Details

The HRP algorithm:

1. Calculate returns from input prices
2. Compute covariance matrix from returns
3. Cluster assets based on distance matrix
4. Apply recursive bisection with inverse variance weighting
5. Results in naturally diversified portfolio without matrix inversion

The function accepts price data and calculates returns internally, matching the pattern of other library functions like calc_momentum().

Value

Weight matrix with same dates as selected_df

Examples

```
data("sample_prices_daily")
data("sample_prices_weekly")
# Create a selection first
momentum <- calc_momentum(sample_prices_weekly, lookback = 12)
selected <- filter_top_n(momentum, n = 10)

# Using daily prices for risk calculation
weights <- weight_by_hrp(selected, sample_prices_daily, lookback_periods = 252)

# Using correlation-based clustering
weights <- weight_by_hrp(selected, sample_prices_daily, use_correlation = TRUE)
```

---

weight_by_rank *Rank-Based Portfolio Weighting*

Description

Weights securities based on their rank rather than raw signal values. Useful when signal magnitudes are unreliable but ordering is meaningful.

Usage

```
weight_by_rank(
  selected_df,
  signal_df,
  method = c("linear", "exponential"),
  ascending = FALSE
)
```

Arguments

selected_df	Binary selection matrix
signal_df	Signal values for ranking
method	Weighting method: "linear" or "exponential"
ascending	Sort order for ranking (default: FALSE)

Value

Data.table with rank-based weights

Examples

```
data("sample_prices_weekly")
momentum <- calc_momentum(sample_prices_weekly, lookback = 12)
selected <- filter_top_n(momentum, 10)
# Linear rank weighting (best gets most)
weights <- weight_by_rank(selected, momentum, method = "linear")
# Exponential (heavy on top stocks)
weights_exp <- weight_by_rank(selected, momentum, method = "exponential")
```

---

weight_by_regime *Regime-Based Adaptive Weighting*

Description

Applies different weighting methods based on market regime classification. Enables adaptive strategies that change allocation approach in different market conditions.

Usage

```
weight_by_regime(
  selected_df,
  regime,
  weighting_configs,
  signal_df = NULL,
  vol_timeframe_data = NULL,
  strategy_timeframe_data = NULL
)
```

Arguments

selected_df	Binary selection matrix (1 = selected, 0 = not)
regime	Regime classification (integer values per period)
weighting_configs	List with method-specific parameters
signal_df	Signal values (required for signal/rank methods)
vol_timeframe_data	Volatility data (required for volatility method)
strategy_timeframe_data	Strategy timeframe alignment data

Value

Data.table with regime-adaptive weights

Examples

```
data("sample_prices_weekly")
# Create selection and signals
momentum <- calc_momentum(sample_prices_weekly, lookback = 12)
selected <- filter_top_n(momentum, n = 10)

# Create a simple regime (example: based on market trend)
ma20 <- calc_moving_average(sample_prices_weekly, 20)
spy_price <- sample_prices_weekly$SPY
spy_ma <- ma20$SPY
regime <- ifelse(spy_price > spy_ma, 1, 2)

# Different weights for bull/bear markets
weighting_configs <- list(
  "1" = list(method = "equal"),
  "2" = list(method = "signal")
)
weights <- weight_by_regime(selected, regime, weighting_configs,
                            signal_df = momentum)
```

---

weight_by_risk_parity *Risk Parity Weighting Suite*

Description

Collection of risk-based weighting methods for portfolio construction. Each method allocates capital based on risk characteristics rather than market capitalization or equal weights.

Usage

```
weight_by_risk_parity(
  selected_df,
  prices_df,
  method = c("inverse_vol", "equal_risk", "max_div"),
  lookback_periods = 252,
  min_periods = 60
)
```

Arguments

selected_df	Binary selection matrix (data.frame with Date column).
prices_df	Price data for risk calculations (typically daily). Returns are calculated internally from prices.
method	Optimization method for risk parity.
lookback_periods	Number of periods for risk estimation (default: 252).
min_periods	Minimum periods required (default: 60).

Details

Methods

`inverse_vol` Weights proportional to 1 / volatility (e.g., 1 / sd of returns over `lookback_periods`). Lower volatility stocks receive higher weights.

`equal_risk` Equal Risk Contribution (ERC): finds weights so each position contributes equally to total portfolio risk (iterative optimization).

`max_div` Maximum Diversification: maximizes the ratio of weighted average volatility to portfolio volatility.

The function accepts price data and calculates returns internally, ensuring consistency with other library functions. Daily prices are recommended for accurate volatility estimation.

Value

Weight matrix with the same dates as `selected_df`; each row sums to 1.

Examples

```
data("sample_prices_daily")
data("sample_prices_weekly")
momentum <- calc_momentum(sample_prices_weekly, lookback = 12)
selected <- filter_top_n(momentum, n = 10)
weight_by_risk_parity(selected, sample_prices_daily, method = "inverse_vol")
weight_by_risk_parity(selected, sample_prices_daily, method = "equal_risk")
weight_by_risk_parity(selected, sample_prices_daily, method = "max_div")
```

`weight_by_signal` *Signal-Based Portfolio Weighting*

Description

Weights selected securities proportionally to their signal strength. Stronger signals receive higher allocations.

Usage

```
weight_by_signal(selected_df, signal_df)
```

Arguments

`selected_df`	Binary selection matrix
`signal_df`	Signal values for weighting

Value

Data.table with signal-proportional weights

Examples

```
data("sample_prices_weekly")
momentum <- calc_momentum(sample_prices_weekly, lookback = 12)
selected <- filter_top_n(momentum, 10)
# Weight by momentum strength
weights <- weight_by_signal(selected, momentum)
```

---

weight_by_volatility *Volatility-Based Portfolio Weighting*

Description

Weights securities based on their volatility characteristics. Can prefer low-volatility (defensive) or high-volatility (aggressive) stocks.

Usage

```
weight_by_volatility(
  selected_df,
  vol_timeframe_data,
  strategy_timeframe_data = NULL,
  lookback_periods = 26,
  low_vol_preference = TRUE,
  vol_method = "std",
  weighting_method = c("rank", "equal", "inverse_variance")
)
```

Arguments

selected_df	Binary selection matrix (1 = selected, 0 = not)
vol_timeframe_data	Price data for volatility calculation (usually daily)
strategy_timeframe_data	Price data matching strategy frequency
lookback_periods	Number of periods for volatility (default: 26)
low_vol_preference	TRUE = lower vol gets higher weight (default: TRUE)
vol_method	"std", "range", "mad", or "abs_return"
weighting_method	"rank", "equal", or "inverse_variance"

Value

Data.table with volatility-based weights

Examples

```
data("sample_prices_weekly")
data("sample_prices_daily")
momentum <- calc_momentum(sample_prices_weekly, lookback = 12)
selected <- filter_top_n(momentum, 10)
daily_vol <- calc_rolling_volatility(sample_prices_daily, lookback = 252)
aligned_vol <- align_to_timeframe(daily_vol, sample_prices_weekly$Date)
weights <- weight_by_volatility(selected, aligned_vol, low_vol_preference = TRUE)
```

---

weight_equally *Equal Weight Portfolio Construction*

Description

Creates equal-weighted portfolio from selection matrix. The simplest and often most robust weighting scheme.

Usage

```
weight_equally(selected_df)
```

Arguments

selected_df	Binary selection matrix (1 = selected, 0 = not)

Value

Data.table with equal weights for selected securities

Examples

```
data("sample_prices_weekly")
momentum <- calc_momentum(sample_prices_weekly, lookback = 12)
selected <- filter_top_n(momentum, 10)
weights <- weight_equally(selected)
```

---

weight_from_scores *Map scores to portfolio weights*

Description

Map scores to portfolio weights

Usage

```
weight_from_scores(
  scores,
  method = c("softmax", "rank", "linear", "equal"),
  temperature = 10,
  floor = 0
)
```

Arguments

scores	Wide score panel (Date + symbols).
method	"softmax", "rank", "linear", "equal".
temperature	softmax temperature (higher=flatter).
floor	non-negative number added before normalization.

Value

weight panel (Date + symbols) with non-negative rows summing to 1 over active dates.

---

wf_report *Generate Walk-Forward Report*

Description

Prints a concise summary of a wf_optimization_result: configuration, stitched OOS performance, and parameter stability.

Usage

```
wf_report(wf, digits = 4)
```

Arguments

wf	A wf_optimization_result object (from run_walk_forward()).
digits	Integer; number of digits when printing numeric values (default 4).

Value

Invisibly returns the optimization summary data frame.

wf_stitch *Stitch Out-of-Sample Results (overlap-safe)*

Description

Concatenates OOS backtests and safely compounds returns on overlapping dates.

Usage

```
wf_stitch(oos_results, initial_value = 1e+05)
```

Arguments

oos_results	List of backtest_result objects, each with $portfolio_values and $dates.
initial_value	Numeric starting value for the stitched equity curve (default 100000).

Value

Data frame with columns: Date, Value.

wf_sweep_tabular *Walk-forward sweep of tabular configs (window-wise distribution of metrics)*

Description

Walk-forward sweep of tabular configs (window-wise distribution of metrics)

Usage

```
wf_sweep_tabular(
  features_list,
  labels,
  prices,
  fit_fn,
  predict_fn,
  schedule = list(is = 104L, oos = 4L, step = 4L),
 grid = list(top_k = c(10L, 15L), temperature = c(8, 12), method = c("softmax", "rank"),
    transform = c("zscore")),
  caps = list(max_per_symbol = 0.08, max_per_group = NULL),
  group_map = NULL,
  freq = 52,
  cost_bps = 0,
  max_windows = NULL,
  ic_method = "spearman"
)
```

Arguments

features_list	List of feature panels.
labels	Label panel.
prices	Price panel for backtests.
fit_fn, predict_fn	Model fit and predict functions.
schedule	List with elements is, oos, step.
grid	Named list of parameter vectors to sweep (e.g., top_k, temperature, method, transform).
caps	Exposure caps (e.g., max_per_symbol/max_per_group).
group_map	Optional Symbol->Group mapping for group caps/selection.
freq	Compounding frequency for annualization (e.g., 52 for weekly).
cost_bps	One-way trading cost in basis points (applied on rebalance).
max_windows	optional limit for speed.
ic_method	IC method ('spearman' or 'pearson').

Value

data.table with medians/means across OOS windows.

---

yahoo_adapter *Download Price Data from Yahoo Finance*

Description

Downloads stock price data directly from Yahoo Finance using quantmod. No database required - perfect for quick analysis and experimentation. Get started with real data in under 5 minutes.

Usage

```
yahoo_adapter(symbols, start_date, end_date, frequency = "daily")
```

Arguments

symbols	Character vector of stock symbols
start_date	Start date in "YYYY-MM-DD" format
end_date	End date in "YYYY-MM-DD" format
frequency	"daily" or "weekly" (default: "daily")

Value

Data.table with Date column and one column per symbol

Examples

```
# Use included sample data
data(sample_prices_weekly)

# Build a quick momentum strategy with offline data
momentum <- calc_momentum(sample_prices_weekly, lookback = 12)
selected <- filter_top_n(momentum, n = 2)
weights <- weight_equally(selected)
result <- run_backtest(sample_prices_weekly, weights, initial_capital = 100000)


# Download tech stocks (requires internet, skipped on CRAN)
if (requireNamespace("quantmod", quietly = TRUE)) {
  prices <- yahoo_adapter(
    symbols = c("AAPL", "MSFT", "GOOGL"),
    start_date = "2023-01-01",
    end_date = "2023-12-31",
    frequency = "weekly"
  )
  momentum <- calc_momentum(prices, lookback = 12)
}
```

Index

∗ **Chapter3-helpers**
    pt_collect_results, 70
    scores_oos_only, 84
∗ **datasets**
    sample_prices_daily, 81
    sample_prices_weekly, 82
    sample_sp500_sectors, 83

align_to_timeframe, 4
analyze_by_period, 5
analyze_drawdowns, 6
analyze_performance, 7
analyze_vs_benchmark, 8
apply_regime, 9
as_selection, 10

backtest_metrics, 11
bucket_returns, 11

calc_cci, 12
calc_distance, 13
calc_market_breadth, 13
calc_momentum, 14
calc_momentum(), *17*, *21*
calc_moving_average, 15
calc_moving_average(), *21*
calc_relative_strength_rank, 15
calc_rolling_correlation, 16
calc_rolling_volatility, 17
calc_rolling_volatility(), *17*
calc_rsi, 18
calc_sector_breadth, 18
calc_sector_relative_indicators, 19
calc_stochastic_d, 20
calc_stochrsi, 21
calculate_daily_values, 22
calculate_drawdown_series, 23
cap_exposure, 24
cap_turnover, 25
carry_forward_weights, 26
combine_filters, 26
combine_scores, 27
combine_weights, 28
convert_to_nweeks, 29
coverage_by_date, 29
create_regime_buckets, 30
csv_adapter, 31
cv_tune_seq, 32

demo_sector_map, 33
download_sp500_sectors, 34

ensure_dt_copy, 34
evaluate_scores, 35

filter_above, 36
filter_below, 36
filter_between, 37
filter_by_percentile, 37
filter_rank, 38
filter_threshold, 39
filter_top_n, 39
filter_top_n(), *21*
filter_top_n_where, 40

get_data_frequency, 41

ic_series, 41, *77*
invert_signal, 42

join_panels, 43

limit_positions, 43
list_examples, 44
load_mixed_symbols, 45

make_labels, 46
manual_adapter, 47
membership_stability, 47
metric_sharpe, 48
ml_add_interactions, 48

ml_add_interactions(), *60*
ml_backtest, 49
ml_backtest(), *50, 51*
ml_backtest_multi, 50
ml_backtest_multi(), *72*
ml_backtest_seq, 52
ml_ic_series_on_scores, 54
ml_ic_series_on_scores(), *59*
ml_make_ensemble, 54
ml_make_model, 55
ml_make_model(), *51*, *54*
ml_make_seq_model, 56
ml_panel_op, 57
ml_panel_op(), *58*
ml_panel_reduce, 58
ml_plot_ic_roll, 59
ml_prepare_features, 59
ml_prepare_features(), *51*

panel_lag, 60
panel_returns_simple, 61
perf_metrics, 61
perf_metrics(), *72*
plot.backtest_result, 62
plot.param_grid_result, 62
plot.performance_analysis, 64
plot.wf_optimization_result, 65
portfolio_returns, 67
print.backtest_result, 68
print.param_grid_result, 69
print.param_grid_result(), *64*
print.performance_analysis, 69
print.wf_optimization_result, 70
print.wf_optimization_result(), *66*
pt_collect_results, 70, *84*
pt_collect_results(), *84*

rank_within_sector, 73
rebalance_calendar, 74
roll_fit_predict, 74
roll_fit_predict_seq, 75
roll_ic_stats, 77
run_backtest, 77
run_example, 78
run_param_grid, 79
run_param_grid(), *62*, *64*, *69*
run_walk_forward, 80
run_walk_forward(), *65, 66*, *70*

safe_divide, 81
sample_prices_daily, 81
sample_prices_weekly, 82
sample_sp500_sectors, 83
scores_oos_only, *72*, 84
scores_oos_only(), *71, 72*
select_top_k_scores, 85
select_top_k_scores_by_group, 85
sql_adapter, 86
sql_adapter_adjusted, 87
summary.backtest_result, 88
switch_weights, 89

transform_scores, 90
TTR::RSI(), *21*
tune_ml_backtest, 90
turnover_by_date, 91

update_vix_in_db, 92

validate_data_format, 92
validate_group_map, 93
validate_no_leakage, 94
vol_target, 94

weight_by_hrp, 95
weight_by_rank, 97
weight_by_regime, 98
weight_by_risk_parity, 99
weight_by_risk_parity(), *21*
weight_by_signal, 100
weight_by_volatility, 101
weight_equally, 102
weight_from_scores, 102
wf_report, 103
wf_report(), *66*
wf_stitch, 104
wf_sweep_tabular, 104

yahoo_adapter, 105