

# Package ‘finlabR’

April 22, 2026

**Type** Package

**Title** Portfolio Analytics and Simulation Toolkit

**Version** 1.0.0

**Description** Tools for portfolio construction and risk analytics, including mean-variance optimization, conditional value at risk (expected shortfall) minimization, risk parity, regime clustering, correlation analysis, Monte Carlo simulation, and option pricing. Includes utilities for portfolio evaluation, clustering, and risk reporting. Methods are based in part on Markowitz (1952) <[doi:10.1111/j.1540-6261.1952.tb01525.x](https://doi.org/10.1111/j.1540-6261.1952.tb01525.x)>, Rockafellar and Uryasev (2000) <[doi:10.21314/JOR.2000.038](https://doi.org/10.21314/JOR.2000.038)>, Maillard et al. (2010) <[doi:10.3905/jpm.2010.36.4.060](https://doi.org/10.3905/jpm.2010.36.4.060)>, Black and Scholes (1973) <[doi:10.1086/260062](https://doi.org/10.1086/260062)>, and Cox et al. (1979) <[doi:10.1016/0304-405X\(79\)90015-1](https://doi.org/10.1016/0304-405X(79)90015-1)>.

**License** MIT + file LICENSE

**Encoding** UTF-8

**LazyData** true

**Imports** stats, utils, quadprog, ggplot2, PerformanceAnalytics, zoo, class, quantmod, reshape2, mclust, shiny

**Suggests** bslib, TTR, DT, xts, yfR, cryptoQuotes, tidyquant, Rtsne, umap, testthat, knitr, rmarkdown

**VignetteBuilder** knitr

**RoxygenNote** 7.3.3

**NeedsCompilation** no

**Author** Suyash Jindal [aut, cre]

**Maintainer** Suyash Jindal <[jindalsuyash7@gmail.com](mailto:jindalsuyash7@gmail.com)>

**Depends** R (>= 3.5.0)

**Repository** CRAN

**Date/Publication** 2026-04-22 13:20:14 UTC

## Contents

american_option_binomial . . . . .	4
annualize_returns . . . . .	5
asset_clustering . . . . .	5
asset_correlation . . . . .	6
asset_correlation_matrix . . . . .	7
binomial_tree_option . . . . .	8
bootstrap_returns . . . . .	8
bs_option_price . . . . .	9
calc_returns . . . . .	10
clt_demonstration . . . . .	11
clt_pnl_ci . . . . .	12
clt_sample_means . . . . .	12
cluster_book_kmeans . . . . .	13
cluster_summary . . . . .	13
compute_efficient_frontier . . . . .	14
consistency_check . . . . .	15
cross_asset_analysis . . . . .	15
cross_validate_portfolio . . . . .	16
cvar_frontier . . . . .	17
cvar_minimize . . . . .	17
detect_regimes . . . . .	18
download_prices . . . . .	19
embedding_2d . . . . .	20
em_clustering . . . . .	20
em_regime . . . . .	21
equal_risk_contribution . . . . .	22
example_prices . . . . .	23
extract_features . . . . .	23
fetch_yahoo_prices . . . . .	24
format_weights . . . . .	24
gaussian_mixture_em . . . . .	25
gbm_simulation . . . . .	25
gd_max_sharpe . . . . .	27
gd_min_variance . . . . .	28
get_example_prices . . . . .	29
get_returns . . . . .	29
gradient_descent . . . . .	30
gradient_descent_portfolio . . . . .	30
kmeans_regime . . . . .	31
knn_classify . . . . .	32
knn_money_flow . . . . .	33
knn_predict . . . . .	34
market_regime_kmeans . . . . .	34
max_sharpe_portfolio . . . . .	35
mc_price_simulation . . . . .	36
mc_return_distribution . . . . .	37

mc_statistics . . . . .	37
minimize_cvar . . . . .	38
min_variance_portfolio . . . . .	39
money_flow_knn . . . . .	39
monte_carlo_option . . . . .	40
mvo_efficient_frontier . . . . .	41
mvo_max_sharpe . . . . .	42
mvo_min_variance . . . . .	42
mvo_summary . . . . .	43
optimize_quotes_gd . . . . .	44
option_greeks . . . . .	44
option_price_simulation . . . . .	45
option_price_summary . . . . .	46
performance_summary . . . . .	47
plot_asset_clusters . . . . .	47
plot_binomial_tree . . . . .	48
plot_correlation_heatmap . . . . .	48
plot_cvar_frontier . . . . .	49
plot_efficient_frontier . . . . .	49
plot_embedding . . . . .	50
plot_gd_convergence . . . . .	51
plot_mc_paths . . . . .	51
plot_option_simulation . . . . .	52
plot_pca_biplot . . . . .	52
plot_regimes . . . . .	53
plot_risk_contribution . . . . .	53
portfolio_asset_clustering . . . . .	54
portfolio_clustering . . . . .	55
portfolio_cvar . . . . .	55
portfolio_pca . . . . .	56
portfolio_performance . . . . .	57
portfolio_tsne . . . . .	58
portfolio_umap . . . . .	59
predict_regime_knn . . . . .	59
price_option_binomial . . . . .	60
price_option_mc . . . . .	61
regime_statistics . . . . .	62
risk_contribution . . . . .	62
risk_parity_portfolio . . . . .	63
risk_parity_weights . . . . .	64
rolling_correlation . . . . .	64
rolling_cv_forecast . . . . .	65
run_quantportr_app . . . . .	66
sampling_distribution . . . . .	66
scree_plot . . . . .	67
simulate_gbm_paths . . . . .	67
simulate_orderbook . . . . .	68
unbiasedness_check . . . . .	69

var_cvar . . . . .	69
var_cvar_analysis . . . . .	70

<b>Index</b>	<b>71</b>
--------------	-----------

---

american\_option\_binomial

*American Option Pricing via Binomial Tree*

---

### Description

Extends the CRR binomial tree with early exercise at each node. Correctly prices American puts and calls.

### Usage

```
american_option_binomial(
  S,
  K,
  time_to_expiry,
  r,
  sigma,
  n = 200,
  type = "put",
  q = 0
)
```

### Arguments

S	Current stock price.
K	Strike price.
time_to_expiry	Time to expiry (years).
r	Risk-free rate (annual, continuous).
sigma	Volatility (annual).
n	Number of time steps. Default 200.
type	"call" or "put". American puts are commonly priced here.
q	Continuous dividend yield. Default 0.

### Value

A list: price, early\_exercise\_boundary, european\_price, early\_premium.

**Examples**

```
# American put (early exercise premium should be positive for ITM put)
am <- american_option_binomial(S = 100, K = 105, time_to_expiry = 1,
                               r = 0.05, sigma = 0.25, n = 200,
                               type = "put")
cat("American put:", am$price, " European put:", am$european_price,
    " Early premium:", am$early_premium, "\n")
```

---

annualize\_returns      *Annualise Returns*

---

**Description**

Annualise Returns

**Usage**

```
annualize_returns(returns, freq = 252, type = "geometric")
```

**Arguments**

returns	Numeric. Per-period returns.
freq	Integer. Periods per year. Default 252.
type	Character. "arithmetic" or "geometric". Default "geometric".

**Value**

Numeric. Annualised return.

---

asset\_clustering      *Asset Clustering with Optional PCA Reduction*

---

**Description**

Groups assets by similarity of their return distributions using k-means or Gaussian mixture EM, with an optional PCA dimensionality reduction step applied before clustering.

**Usage**

```
asset_clustering(
  returns,
  method = c("kmeans", "em"),
  k = 3,
  reduce = c("pca", "none"),
  n_components = 3,
  seed = 123
)
```

**Arguments**

returns	Matrix or data.frame of returns (rows = time, cols = assets).
method	Character. "kmeans" (default) or "em".
k	Integer. Number of clusters. Default 3.
reduce	Character. Dimensionality reduction: "pca" (default) or "none".
n_components	Integer. Number of PCA components to retain. Default 3.
seed	Integer. Random seed. Default 123.

**Value**

A list with elements `clusters` (named integer vector), `model` (fitted model object), and `pca` (the `prcomp` object, or `NULL` if `reduce = "none"`).

**Examples**

```
set.seed(2)
R <- matrix(rnorm(300 * 6, 0.0003, 0.01), 300, 6)
colnames(R) <- paste0("A", 1:6)
res <- asset_clustering(R, method = "kmeans", k = 3)
res$clusters
```

---

asset\_correlation      *Correlation Analysis Across Asset Groups*

---

**Description**

Computes a Pearson, Spearman, or Kendall correlation matrix from either a single return matrix or a named list of return matrices (one per asset group). When a list is supplied, matrices are column-bound and column names are prefixed with the group index.

**Usage**

```
asset_correlation(returns_list, method = c("pearson", "spearman", "kendall"))
```

**Arguments**

returns_list	A numeric matrix/data.frame of returns (T x N), OR a named list of such matrices — one element per asset group.
method	Character. Correlation method: "pearson" (default), "spearman", or "kendall".

**Value**

A symmetric numeric correlation matrix (N x N).

## Examples

```
set.seed(1)
R <- matrix(rnorm(300 * 4, 0.0003, 0.01), 300, 4)
colnames(R) <- c("EQ1", "EQ2", "CMD1", "BOND1")
asset_correlation(R)
```

---

asset\_correlation\_matrix

*Compute Cross-Asset Correlation Matrix*

---

## Description

Computes the Pearson, Spearman, or Kendall correlation matrix across multiple asset classes (equities, commodities, crypto, bonds).

## Usage

```
asset_correlation_matrix(  
  returns,  
  method = "pearson",  
  use_ = "pairwise.complete.obs"  
)
```

## Arguments

returns	Numeric matrix (T x N) of asset returns.
method	Character. "pearson", "spearman", or "kendall". Default "pearson".
use_	Character. Passed to <code>cor()</code> . Default "pairwise.complete.obs".

## Value

A list: `cor_matrix`, `p_values`, `labels`.

## Examples

```
set.seed(42)
R <- matrix(rnorm(300 * 6, 0.0003, 0.01), 300, 6)
colnames(R) <- c("SPY", "GLD", "BTC", "OIL", "TLT", "ETH")
res <- asset_correlation_matrix(R)
round(res$cor_matrix, 3)
```

---

binomial\_tree\_option *Binomial Tree Option Pricing (European)*

---

**Description**

Uses the Cox-Ross-Rubinstein (CRR) binomial model.

**Usage**

```
binomial_tree_option(S, K, time_to_expiry, r, sigma, n = 100, type = "call")
```

**Arguments**

S	Current stock price.
K	Strike price.
time_to_expiry	Time to expiry (years).
r	Risk-free rate (annual, continuous).
sigma	Volatility (annual).
n	Number of time steps. Default 100.
type	"call" or "put".

**Value**

A list: price, u, d, p, tree (stock price tree), option\_tree.

**Examples**

```
binomial_tree_option(S = 100, K = 100, time_to_expiry = 1,  
                    r = 0.05, sigma = 0.20, n = 50)
```

---

bootstrap\_returns *Bootstrap Returns*

---

**Description**

Generates a bootstrap distribution of portfolio statistics.

**Usage**

```
bootstrap_returns(
  returns,
  weights = NULL,
  stat_fn = NULL,
  n_boot = 1000,
  block_size = 1,
  freq = 252,
  seed = 42
)
```

**Arguments**

returns	Numeric vector or matrix of returns.
weights	Optional weight vector. Default equal weight.
stat_fn	Function applied to each bootstrap sample. Default: mean * freq.
n_boot	Integer. Number of bootstrap replicates. Default 1000.
block_size	Integer. Block bootstrap size (1 = iid). Default 1.
freq	Integer. Default 252.
seed	Integer. Default 42.

**Value**

A list: boot\_stat, mean, se, ci\_95.

---

bs_option_price	<i>Black-Scholes Option Price</i>
-----------------	-----------------------------------

---

**Description**

Black-Scholes Option Price

**Usage**

```
bs_option_price(S, K, time_to_expiry, r, sigma, type = "call", q = 0)
```

**Arguments**

S	Current stock price.
K	Strike price.
time_to_expiry	Time to expiry in years.
r	Risk-free rate (continuous, annualised).
sigma	Volatility (annualised).
type	"call" or "put". Default "call".
q	Dividend yield. Default 0.

**Value**

Numeric. Option price.

**Examples**

```
bs_option_price(100, 100, 1, 0.05, 0.20, "call")
bs_option_price(100, 105, 0.5, 0.04, 0.25, "put")
```

---

calc\_returns

*Compute Asset Returns from a Price Series*

---

**Description**

Converts a matrix or data frame of asset prices into a matrix of period returns. The first row is dropped (no prior price to diff against), so the result has one fewer row than the input.

**Usage**

```
calc_returns(prices, method = c("log", "simple"), na.rm = TRUE)
```

**Arguments**

prices	Numeric matrix, data.frame, xts, or zoo of prices (T x N). Remove any non-numeric columns (e.g. a date column) before calling: <code>calc_returns(prices[, -1])</code> .
method	Character. Return type: "log" (default, continuously compounded) or "simple" (arithmetic).
na.rm	Logical. If TRUE (default), rows that still contain NA after differencing are dropped.

**Value**

A numeric matrix of returns with `nrow(prices) - 1` rows and `ncol(prices)` columns. Column names are preserved.

**Examples**

```
prices <- get_example_prices()
rets <- calc_returns(prices[, -1])
rets_s <- calc_returns(prices[, -1], method = "simple")
dim(rets)
```

---

clt\_demonstration      *CLT Demonstration*

---

## Description

Illustrates the Central Limit Theorem by drawing repeated samples of increasing size from a population and plotting the distribution of sample means against the theoretical Normal curve.

## Usage

```
clt_demonstration(  
  data,  
  n_samples = c(5, 10, 30, 100),  
  n_reps = 2000,  
  seed = 123  
)
```

## Arguments

data	Numeric vector. Population data (e.g. historical returns).
n_samples	Integer vector. Sample sizes to test. Default c(5, 10, 30, 100).
n_reps	Integer. Repetitions per sample size. Default 2000.
seed	Integer. Default 123.

## Value

A list with:

results	data.frame of sample means and Shapiro-Wilk statistics.
clt_plot	A ggplot2 faceted histogram with Normal overlay.
pop_mean	Population mean.
pop_sd	Population standard deviation.

## Examples

```
set.seed(1)  
r <- rnorm(500, 0.0004, 0.012)  
clt <- clt_demonstration(r)  
clt$clt_plot
```

---

clt_pnl_ci	<i>Evaluate Strategy PnL using CLT Confidence Intervals</i>
------------	---

---

**Description**

Applies the Central Limit Theorem to a vector of simulated daily profit-and-loss values and returns the sample mean together with a symmetric confidence interval.

**Usage**

```
clt_pnl_ci(pnl_vector, confidence_level = 0.95)
```

**Arguments**

pnl_vector	Numeric vector of simulated daily PnL values.
confidence_level	Numeric. Desired confidence level. Default 0.95.

**Value**

A list with three elements: mean\_pnl, lower\_ci, and upper\_ci.

**Examples**

```
set.seed(10)
pnl <- rnorm(252, mean = 50, sd = 300)
clt_pnl_ci(pnl, confidence_level = 0.95)
```

---

clt_sample_means	<i>CLT Sample Means</i>
------------------	-------------------------

---

**Description**

Returns sample mean statistics for the CLT demonstration.

**Usage**

```
clt_sample_means(data, n, n_reps = 1000, seed = 42)
```

**Arguments**

data	Numeric vector.
n	Integer. Sample size.
n_reps	Integer. Replications. Default 1000.
seed	Integer. Default 42.

**Value**

Numeric vector of sample means.

---

cluster\_book\_kmeans     *Cluster Order Book States using K-Means*

---

**Description**

Identifies market regimes by running k-means clustering on the four microstructure features produced by `extract_features()`. Features are z-score scaled before clustering.

**Usage**

```
cluster_book_kmeans(features, centers = 4)
```

**Arguments**

`features`     A data.frame of extracted features (output of `extract_features()`).  
`centers`     Integer. Number of clusters (regimes). Default 4.

**Value**

A list with two elements: `model` (the kmeans object) and `data` (the input data frame with a new regime column).

**Examples**

```
set.seed(7)
book <- extract_features(simulate_orderbook(500))
result <- cluster_book_kmeans(book, centers = 3)
table(result$data$regime)
```

---

cluster\_summary     *Cluster Summary*

---

**Description**

Cluster Summary

**Usage**

```
cluster_summary(cl_obj)
```

**Arguments**

cl\_obj            Output from asset\_clustering().

**Value**

Data.frame of cluster statistics.

---

compute\_efficient\_frontier

*Compute the Efficient Frontier*

---

**Description**

Solves a sequence of minimum-variance QP problems (via quadprog) across a grid of target returns to trace the efficient frontier. Returns and risk are annualised.

**Usage**

```
compute_efficient_frontier(
  returns,
  n_points = 100,
  allow_short = FALSE,
  risk_free = 0,
  freq = 252
)
```

**Arguments**

returns            Numeric matrix or xts of asset returns (T x N).  
n\_points            Integer. Number of frontier points. Default 100.  
allow\_short        Logical. Allow short selling? Default FALSE.  
risk\_free          Numeric. Risk-free rate (annualised). Default 0.  
freq                Integer. Periods per year (252 = daily, 12 = monthly). Default 252.

**Value**

A list with:

frontier           data.frame of Return, Risk, Sharpe, and asset weights.  
cov\_matrix         Annualised sample covariance matrix.  
mu                  Annualised expected return vector.  
assets              Asset names.

**Examples**

```
set.seed(42)
R <- matrix(rnorm(500 * 5, 0.0005, 0.01), 500, 5)
colnames(R) <- paste0("Asset", 1:5)
ef <- compute_efficient_frontier(R)
head(ef$frontier)
```

---

consistency\_check      *Consistency Check*

---

**Description**

Tests whether an estimator is consistent: SE decreases as n grows.

**Usage**

```
consistency_check(estimates_by_n)
```

**Arguments**

`estimates_by_n` Named list where names are sample sizes and values are numeric vectors of estimates.

**Value**

A data.frame with SE by n and consistency result.

---

cross\_asset\_analysis      *Cross-Asset Correlation Analysis*

---

**Description**

Comprehensive cross-asset study across equities, commodities, and crypto.

**Usage**

```
cross_asset_analysis(returns, asset_classes = NULL, window = 60, freq = 252)
```

**Arguments**

`returns` Matrix (T x N) with named columns.

`asset_classes` Named list mapping asset names to class labels, e.g. list(SPY="Equity", GLD="Commodity", BTC="Crypto").

`window` Integer. Rolling window for rolling correlations. Default 60.

`freq` Integer. Periods per year. Default 252.

**Value**

A list: static\_cor, rolling\_cors, within\_class, between\_class, plots.

---

cross\_validate\_portfolio

*Time-Series Cross-Validation for Portfolio Models*

---

**Description**

Implements rolling-window and expanding-window cross-validation for portfolio construction, measuring out-of-sample Sharpe and consistency / unbiasedness of estimates.

**Usage**

```
cross_validate_portfolio(
  returns,
  model_fn,
  n_folds = 5,
  window_type = "expanding",
  min_train = 60,
  eval_period = 21,
  freq = 252
)
```

**Arguments**

returns	Numeric matrix (T x N) of returns.
model_fn	Function taking a training returns matrix and returning a weight vector. E.g. function(R) min_variance_portfolio(R)\$weights.
n_folds	Integer. Number of CV folds. Default 5.
window_type	Character. "rolling" or "expanding". Default "expanding".
min_train	Integer. Minimum training observations. Default 60.
eval_period	Integer. Evaluation (test) window per fold. Default 21.
freq	Integer. Periods per year. Default 252.

**Value**

A list: fold\_results, oos\_sharpe, consistency, unbiasedness.

**Examples**

```

set.seed(42)
R <- matrix(rnorm(500 * 4, 0.0003, 0.012), 500, 4)
colnames(R) <- paste0("A", 1:4)
cv <- cross_validate_portfolio(
  R,
  model_fn = function(r) min_variance_portfolio(r)$weights,
  n_folds = 5
)
cat("00S Sharpe:", cv$00s_sharpe, "\n")

```

---

cvar\_frontier

*CVaR-Return Frontier*


---

**Description**

Sweeps over target returns to build a CVaR-efficient frontier.

**Usage**

```
cvar_frontier(returns, alpha = 0.95, n_points = 30, freq = 252)
```

**Arguments**

returns	Numeric matrix of asset returns.
alpha	Confidence level. Default 0.95.
n_points	Integer. Number of frontier points. Default 30.
freq	Integer. Periods per year. Default 252.

**Value**

A data.frame with columns: Return, Risk, CVaR, VaR, Sharpe, weights.

---

cvar\_minimize

*CVaR-Minimising Portfolio (Softmax / Lightweight)*


---

**Description**

A simpler CVaR minimiser. For long-only portfolios the weights are parameterised through a softmax transformation so the simplex constraint is satisfied automatically (BFGS). For short-selling the weights are optimised directly with box constraints and a quadratic penalty for  $\sum w = 1$  (L-BFGS-B).

**Usage**

```
cvar_minimize(
  returns,
  alpha = 0.95,
  allow_short = FALSE,
  bounds = c(-1, 1),
  penalty = 1000,
  max_iter = 500
)
```

**Arguments**

returns	Matrix/data.frame of asset returns (T x N).
alpha	Confidence level (e.g. 0.95). Default 0.95.
allow_short	Logical. Allow short positions. Default FALSE.
bounds	Length-2 numeric vector of [lower, upper] weight bounds when allow_short = TRUE. Default c(-1, 1).
penalty	Quadratic penalty weight for the sum-to-one constraint when allow_short = TRUE. Default 1000.
max_iter	Maximum iterations passed to stats::optim. Default 500.

**Value**

A list with elements:

weights	Named numeric vector of portfolio weights.
cvar	Scalar CVaR value at the optimal portfolio.

**Examples**

```
set.seed(7)
R <- matrix(rnorm(400 * 4, 0.0003, 0.01), 400, 4)
colnames(R) <- c("A", "B", "C", "D")
res <- cvar_minimize(R)
res$weights
res$cvar
```

---

detect\_regimes

*Detect Regimes (General Interface)*

---

**Description**

Detect Regimes (General Interface)

**Usage**

```
detect_regimes(returns, method = "kmeans", k = 3, ...)
```

**Arguments**

returns	Numeric vector or matrix column of returns.
method	Character. "kmeans" or "em". Default "kmeans".
k	Integer. Number of regimes.
...	Additional args passed to the method function.

**Value**

Result list from the chosen method.

---

download_prices	<i>Download Prices via quantmod</i>
-----------------	-------------------------------------

---

**Description**

Wraps `quantmod::getSymbols` for clean price downloads.

**Usage**

```
download_prices(
  symbols,
  from = "2020-01-01",
  to = Sys.Date(),
  src = "yahoo",
  return_type = "returns"
)
```

**Arguments**

symbols	Character vector of ticker symbols.
from	Date string "YYYY-MM-DD". Default "2020-01-01".
to	Date string. Default today.
src	Character. Data source. Default "yahoo".
return_type	Character. "prices" or "returns". Default "returns".

**Value**

An xts object.

**Examples**

```
prices <- download_prices(c("SPY", "GLD", "BTC-USD"), from="2021-01-01")
```

---

`embedding_2d`*2D Embedding for Visualisation*

---

**Description**

Projects a numeric matrix down to two dimensions using PCA, UMAP, or t-SNE for visual exploration of asset or regime clusters.

**Usage**

```
embedding_2d(X, method = c("pca", "umap", "tsne"), seed = 123)
```

**Arguments**

<code>X</code>	Numeric matrix (N x D).
<code>method</code>	Character. "pca" (default), "umap", or "tsne".
<code>seed</code>	Integer. Random seed. Default 123.

**Value**

A numeric matrix with two columns (N x 2).

**Examples**

```
set.seed(3)
X <- matrix(rnorm(100 * 5), 100, 5)
emb <- embedding_2d(X, method = "pca")
plot(emb, pch = 19, col = "steelblue")
```

---

`em_clustering`*EM (Gaussian Mixture) Clustering*

---

**Description**

Fits a Gaussian Mixture Model via Expectation-Maximisation using the **mclust** package. Automatically selects the number of components using BIC if `k = NULL`.

**Usage**

```
em_clustering(X, k = 3, scale_ = TRUE, max_k = 8)
```

**Arguments**

X	Numeric matrix (T x P) of features.
k	Integer or NULL. Number of components. If NULL, auto-selects. Default 3.
scale_	Logical. Standardise features. Default TRUE.
max_k	Integer. Max components to try when auto-selecting. Default 8.

**Value**

A list: labels, probabilities, bic, means, covariances, model.

**Examples**

```
if (requireNamespace("mclust", quietly = TRUE)) {
  ok <- tryCatch({
    mclust::Mclust(matrix(rnorm(20), 10, 2), G = 2, verbose = FALSE)
    TRUE
  }, error = function(e) FALSE)
  if (ok) {
    set.seed(42)
    X <- rbind(matrix(rnorm(100*2, c(-2,0), 0.5), 100),
                  matrix(rnorm(100*2, c( 2,0), 0.5), 100))
    res <- em_clustering(X, k = 2)
    table(res$labels)
  }
}
```

---

em\_regime

*EM Algorithm (Gaussian Mixture) Regime Detection*


---

**Description**

Uses Expectation-Maximisation to fit a Gaussian Mixture Model on rolling features. Requires the **mclust** package.

**Usage**

```
em_regime(returns, k = 3, window = 21, freq = 252)
```

**Arguments**

returns	Numeric vector of returns.
k	Integer. Number of components. Default 3.
window	Integer. Rolling window. Default 21.
freq	Integer. Periods per year. Default 252.

**Value**

A list: labels, probabilities, model, stats.

**Examples**

```
if (requireNamespace("mclust", quietly = TRUE)) {
  ok <- tryCatch({
    mclust::Mclust(matrix(rnorm(20), 10, 2), G = 2, verbose = FALSE)
    TRUE
  }, error = function(e) FALSE)
  if (ok) {
    set.seed(1)
    r <- c(rnorm(200, 0.001, 0.01), rnorm(150, -0.002, 0.02))
    res <- em_regime(r, k = 2)
  }
}
```

---

equal\_risk\_contribution

*Equal Risk Contribution (Risk Parity) Portfolio*

---

**Description**

Solves the equal-risk-contribution problem using cyclical coordinate descent. Supports custom risk budgets. Each asset (by default) contributes equally to total portfolio risk.

**Usage**

```
equal_risk_contribution(
  returns,
  budget = NULL,
  freq = 252,
  tol = 1e-08,
  max_iter = 1000
)
```

**Arguments**

returns	Numeric matrix (T x N) of asset returns.
budget	Numeric vector of risk budget weights (default: equal, length N). Will be normalised to sum to 1.
freq	Integer. Periods per year. Default 252.
tol	Numeric. Convergence tolerance. Default 1e-8.
max_iter	Integer. Maximum iterations. Default 1000.

**Value**

A list with elements: weights, risk\_contrib, risk, return, budget, iterations, method.

**Examples**

```
set.seed(42)
R <- matrix(rnorm(300 * 4, 0.0003, 0.01), 300, 4)
colnames(R) <- c("SPY", "GLD", "TLT", "VNQ")
rp <- equal_risk_contribution(R)
rp$weights
```

---

example_prices	<i>Example synthetic price dataset</i>
----------------	--

---

**Description**

Example synthetic price dataset

**Usage**

```
example_prices
```

**Format**

A data frame with 1000 rows (dates) and 6 assets. Columns: date, EQ1, EQ2, EQ3, CMD1, CRYPTO1, BOND1

---

extract_features	<i>Extract Market Microstructure Features</i>
------------------	---

---

**Description**

Appends derived microstructure columns — spread, order-flow imbalance, total depth, and a volatility proxy — to an order book snapshot data frame.

**Usage**

```
extract_features(book)
```

**Arguments**

book            A data.frame produced by simulate\_orderbook().

**Value**

The input data.frame with four additional columns: spread, imbalance, depth\_total, and volatility.

**Examples**

```
set.seed(1)
book <- simulate_orderbook(200)
book <- extract_features(book)
head(book[, c("spread", "imbalance", "volatility")])
```

---

fetch_yahoo_prices	<i>Fetch Yahoo Finance close prices (wrapper around quantmod)</i>
--------------------	---

---

**Description**

Fetch Yahoo Finance close prices (wrapper around quantmod)

**Usage**

```
fetch_yahoo_prices(symbols, from, to)
```

**Arguments**

symbols	Character vector of tickers.
from	Start date.
to	End date.

**Value**

xts object of close prices.

---

format_weights	<i>Format Portfolio Weights</i>
----------------	---------------------------------

---

**Description**

Prints a formatted weight table with percentages.

**Usage**

```
format_weights(weights, digits = 4)
```

**Arguments**

weights	Named numeric vector of portfolio weights.
digits	Integer. Decimal places. Default 4.

**Value**

Invisibly returns the formatted data.frame.

---

gaussian\_mixture\_em      *Gaussian Mixture Model via EM (Diagonal Covariance)*

---

### Description

Fits a Gaussian mixture model with diagonal covariance using the Expectation-Maximisation algorithm. Suitable for soft clustering of asset return features.

### Usage

```
gaussian_mixture_em(X, k = 3, max_iter = 100, tol = 1e-06, seed = 123)
```

### Arguments

X	Numeric matrix (N x D) of observations.
k	Integer. Number of mixture components. Default 3.
max_iter	Integer. Maximum EM iterations. Default 100.
tol	Numeric. Log-likelihood convergence tolerance. Default 1e-6.
seed	Integer. Random seed. Default 123.

### Value

A list with elements weights, means, variances, loglik, and cluster (hard assignments via which.max).

### Examples

```
set.seed(5)
X <- matrix(rnorm(200 * 3), 200, 3)
res <- gaussian_mixture_em(X, k = 2)
table(res$cluster)
```

---

gbm\_simulation      *Geometric Brownian Motion Simulation (Rich Output)*

---

### Description

Simulates stock price paths under GBM with a choice of exact log-normal or Euler-Maruyama discretisation. Returns paths as a paths x time matrix together with terminal-price summary statistics.

**Usage**

```
gbm_simulation(
  S0 = 100,
  mu = 0.08,
  sigma = 0.2,
  time_horizon = 1,
  n_steps = 252,
  n_paths = 1000,
  seed = NULL,
  method = "exact"
)
```

**Arguments**

<code>S0</code>	Initial stock price. Default 100.
<code>mu</code>	Annual drift. Default 0.08.
<code>sigma</code>	Annual volatility. Default 0.20.
<code>time_horizon</code>	Time horizon in years. Default 1.
<code>n_steps</code>	Number of time steps. Default 252.
<code>n_paths</code>	Number of simulated paths. Default 1000.
<code>seed</code>	Integer seed. Default NULL.
<code>method</code>	Character. "exact" (default) or "euler".

**Value**

A list with:

<code>paths</code>	Numeric matrix ( <code>n_paths</code> x <code>n_steps+1</code> ) of simulated prices.
<code>time_grid</code>	Numeric vector of time points.
<code>stats</code>	Named list of terminal-price statistics.

**Examples**

```
sim <- gbm_simulation(S0 = 100, mu = 0.08, sigma = 0.20, n_paths = 500)
sim$stats$prob_profit
```

---

`gd_max_sharpe`*Gradient Descent Maximum Sharpe Portfolio*

---

**Description**

Maximises the Sharpe ratio using Adam-style gradient descent on the negative Sharpe objective.

**Usage**

```
gd_max_sharpe(  
  returns,  
  risk_free = 0.02,  
  lr = 0.001,  
  max_iter = 3000,  
  tol = 1e-08,  
  freq = 252,  
  beta1 = 0.9,  
  beta2 = 0.999,  
  eps_adam = 1e-08  
)
```

**Arguments**

<code>returns</code>	Numeric matrix of returns.
<code>risk_free</code>	Numeric. Risk-free rate. Default 0.02.
<code>lr</code>	Numeric. Adam learning rate. Default 0.001.
<code>max_iter</code>	Integer. Default 3000.
<code>tol</code>	Numeric. Default 1e-8.
<code>freq</code>	Integer. Default 252.
<code>beta1</code>	Numeric. Adam beta1. Default 0.9.
<code>beta2</code>	Numeric. Adam beta2. Default 0.999.
<code>eps_adam</code>	Numeric. Adam epsilon. Default 1e-8.

**Value**

A list: `weights`, `loss_history`, `sharpe`, `risk`, `return`.

---

`gd_min_variance`*Gradient Descent Minimum Variance Portfolio*

---

**Description**

Minimises portfolio variance using projected gradient descent with simplex projection (long-only constraint).

**Usage**

```
gd_min_variance(  
  returns,  
  lr = 0.01,  
  max_iter = 2000,  
  tol = 1e-08,  
  freq = 252,  
  momentum = 0.9,  
  verbose = FALSE  
)
```

**Arguments**

<code>returns</code>	Numeric matrix (T x N) of asset returns.
<code>lr</code>	Numeric. Learning rate. Default 0.01.
<code>max_iter</code>	Integer. Max iterations. Default 2000.
<code>tol</code>	Numeric. Convergence tolerance. Default 1e-8.
<code>freq</code>	Integer. Periods per year. Default 252.
<code>momentum</code>	Numeric. Momentum parameter (0 = vanilla GD). Default 0.9.
<code>verbose</code>	Logical. Print iteration info. Default FALSE.

**Value**

A list: `weights`, `loss_history`, `convergence`, `risk`, `return`.

**Examples**

```
set.seed(42)  
R <- matrix(rnorm(300 * 5, 0.0003, 0.012), 300, 5)  
colnames(R) <- paste0("A", 1:5)  
gd <- gd_min_variance(R, lr = 0.05, max_iter = 1000)  
gd$risk
```

---

get\_example\_prices      *Example Price Data*

---

**Description**

Returns a data frame of simulated daily closing prices for six synthetic assets (three equities, one commodity, one crypto, one bond) covering approximately 1 000 trading days starting 2018-01-01.

**Usage**

```
get_example_prices()
```

**Value**

A data frame with columns date, EQ1, EQ2, EQ3, CMD1, CRYPTO1, and BOND1.

**Examples**

```
prices <- get_example_prices()
head(prices)
```

---

get\_returns      *Compute Returns from Prices*

---

**Description**

Compute Returns from Prices

**Usage**

```
get_returns(prices, type = "log", lag = 1)
```

**Arguments**

prices	Numeric vector, matrix, or xts of prices.
type	Character. "log" or "simple". Default "log".
lag	Integer. Return lag. Default 1.

**Value**

Returns of the same type as input.

**Examples**

```
prices <- c(100, 102, 101, 105, 103)
get_returns(prices)
get_returns(prices, type = "simple")
```

---

gradient_descent	<i>Simple gradient descent optimizer</i>
------------------	--

---

**Description**

Simple gradient descent optimizer

**Usage**

```
gradient_descent(f, grad = NULL, x0, lr = 0.01, max_iter = 1000, tol = 1e-06)
```

**Arguments**

f	Objective function.
grad	Gradient function (optional).
x0	Initial parameter vector.
lr	Learning rate.
max_iter	Maximum iterations.
tol	Convergence tolerance.

**Value**

List with optimized parameters and history.

---

gradient_descent_portfolio	<i>General Gradient Descent Portfolio (wrapper)</i>
----------------------------	---

---

**Description**

General Gradient Descent Portfolio (wrapper)

**Usage**

```
gradient_descent_portfolio(returns, objective = "min_variance", ...)
```

**Arguments**

returns	Numeric matrix of returns.
objective	Character. "min_variance" or "max_sharpe". Default "min_variance".
...	Additional arguments passed to the objective function.

**Value**

List from the chosen optimisation.

---

kmeans_regime	<i>K-Means Market Regime Detection</i>
---------------	--

---

**Description**

Clusters market observations into regimes using k-means on engineered features: rolling return, rolling volatility, and rolling Sharpe ratio.

**Usage**

```
kmeans_regime(
  returns,
  k = 3,
  window = 21,
  n_starts = 50,
  seed = 42,
  freq = 252
)
```

**Arguments**

returns	Numeric vector or single-column matrix of returns.
k	Integer. Number of regimes. Default 3.
window	Integer. Rolling window for feature computation. Default 21.
n_starts	Integer. k-means restarts. Default 50.
seed	Integer. Random seed. Default 42.
freq	Integer. Periods per year. Default 252.

**Value**

A list: labels, centers, features, stats, dates (if xts).

### Examples

```
set.seed(42)
r <- c(rnorm(200, 0.001, 0.01), # bull
       rnorm(100, -0.002, 0.025), # bear
       rnorm(150, 0.0005, 0.008)) # low-vol
res <- kmeans_regime(r, k = 3)
table(res$labels)
```

---

knn\_classify

*kNN Classifier for Financial Signals*

---

### Description

Trains a k-Nearest Neighbours classifier on financial features. Supports time-series aware train/test splitting.

### Usage

```
knn_classify(X, y, k = 5, train_frac = 0.7, scale_ = TRUE, seed = 42)
```

### Arguments

X	Numeric matrix (T x P) of features.
y	Factor or integer vector of labels.
k	Integer. Number of neighbours. Default 5.
train_frac	Numeric. Training set fraction. Default 0.7.
scale_	Logical. Standardise features. Default TRUE.
seed	Integer. Default 42.

### Value

A list: predictions, actual, accuracy, confusion\_matrix, k.

### Examples

```
set.seed(42)
X <- matrix(rnorm(500 * 5), 500, 5)
y <- factor(ifelse(rowSums(X[, 1:2]) > 0, "Up", "Down"))
res <- knn_classify(X, y, k = 7)
res$accuracy
```

---

`knn_money_flow`*kNN Money Flow Analysis*

---

### Description

Uses k-Nearest Neighbours to classify future market direction based on historical money-flow features (volume, price momentum, volatility). Provides a regime-based money flow signal.

### Usage

```
knn_money_flow(  
  returns,  
  volume = NULL,  
  k = 5,  
  window = 10,  
  horizon = 5,  
  train_frac = 0.7,  
  seed = 42  
)
```

### Arguments

<code>returns</code>	Numeric vector or matrix column of returns.
<code>volume</code>	Optional numeric vector of trading volume.
<code>k</code>	Integer. Number of nearest neighbours. Default 5.
<code>window</code>	Integer. Feature rolling window. Default 10.
<code>horizon</code>	Integer. Prediction horizon (steps ahead). Default 5.
<code>train_frac</code>	Numeric. Fraction for training. Default 0.7.
<code>seed</code>	Integer. Default 42.

### Value

A list: predictions, accuracy, confusion\_matrix, feature\_importance.

### Examples

```
set.seed(1)  
r <- rnorm(500, 0.0003, 0.012)  
res <- knn_money_flow(r, k = 7, horizon = 5)  
cat("Accuracy:", res$accuracy, "\n")
```

---

knn_predict	<i>k-Nearest Neighbors prediction</i>
-------------	---------------------------------------

---

**Description**

k-Nearest Neighbors prediction

**Usage**

```
knn_predict(train_X, train_y, new_X, k = 5)
```

**Arguments**

train_X	Training features matrix.
train_y	Training labels (numeric or factor).
new_X	New features matrix.
k	Number of neighbors.

**Value**

Predictions.

---

market_regime_kmeans	<i>Market Regime Clustering with K-Means on Rolling Features</i>
----------------------	--

---

**Description**

Detects market regimes by computing rolling distributional features (mean, volatility, skewness, kurtosis) from a return series and applying k-means clustering to the resulting feature matrix.

**Usage**

```
market_regime_kmeans(  
  returns,  
  k = 3,  
  window = 60,  
  features = c("mean", "vol"),  
  seed = 123  
)
```

**Arguments**

returns	Matrix, data.frame, or xts of returns (T x N).
k	Integer. Number of clusters (regimes). Default 3.
window	Integer. Rolling window length. Default 60.
features	Character vector of features to compute. Any subset of c("mean", "vol", "skew", "kurt"). Default c("mean", "vol").
seed	Integer. Random seed for reproducibility. Default 123.

**Value**

A list with elements features (feature matrix), kmeans (the fitted kmeans object), and labels (integer cluster assignments).

**Examples**

```
set.seed(1)
R <- matrix(rnorm(500 * 4, 0.0003, 0.01), 500, 4)
colnames(R) <- paste0("A", 1:4)
res <- market_regime_kmeans(R, k = 3, window = 60)
table(res$labels)
```

---

max\_sharpe\_portfolio *Maximum Sharpe Ratio Portfolio*

---

**Description**

Finds the tangency portfolio by minimising negative Sharpe ratio via L-BFGS-B optimisation with box constraints.

**Usage**

```
max_sharpe_portfolio(
  returns,
  risk_free = 0.02,
  freq = 252,
  allow_short = FALSE
)
```

**Arguments**

returns	Numeric matrix or xts of asset returns (T x N).
risk_free	Numeric. Annualised risk-free rate. Default 0.02.
freq	Integer. Periods per year. Default 252.
allow_short	Logical. Allow short selling? Default FALSE.

**Value**

A list: weights, return, risk, sharpe, method.

**Examples**

```
set.seed(1)
R <- matrix(rnorm(252 * 4, 0.0003, 0.012), 252, 4)
colnames(R) <- c("SPY", "GLD", "BTC", "BND")
ms <- max_sharpe_portfolio(R, risk_free = 0.05)
ms$sharpe
```

---

mc\_price\_simulation    *Monte Carlo Price / Return Simulation*

---

**Description**

Simulates portfolio or single-asset cumulative return paths from a historical return series. Computes the terminal return distribution, CLT convergence statistics, and k-fold forward-chain cross-validation estimates of annualised return.

**Usage**

```
mc_price_simulation(
  returns,
  n_paths = 5000,
  horizon = 252,
  freq = 252,
  seed = 42,
  use_hist = FALSE,
  n_cv_folds = 5
)
```

**Arguments**

returns	Numeric vector of historical returns.
n_paths	Integer. Number of simulation paths. Default 5000.
horizon	Integer. Steps forward. Default 252.
freq	Integer. Periods per year. Default 252.
seed	Integer. Default 42.
use_hist	Logical. Bootstrap from historical returns (block resampling) instead of Normal draws. Default FALSE.
n_cv_folds	Integer. Number of forward-chain CV folds. Default 5.

**Value**

A list with elements: paths\_matrix, terminal\_returns, mu\_historical, sg\_historical, perc5, perc95, prob\_profit, clt\_stats (data.frame), cv\_estimates (data.frame).

**Examples**

```
set.seed(1)
r <- rnorm(500, 0.0004, 0.012)
mc <- mc_price_simulation(r, n_paths = 1000, horizon = 252)
mc$prob_profit
```

---

mc\_return\_distribution

*Monte Carlo Return Distribution Table*

---

**Description**

Extracts a percentile table of annualised terminal returns from the output of mc\_price\_simulation().

**Usage**

```
mc_return_distribution(mc_obj)
```

**Arguments**

mc\_obj            Output from mc\_price\_simulation().

**Value**

A data.frame with columns Percentile and AnnReturn.

---

mc\_statistics

*Monte Carlo Statistics Summary*

---

**Description**

Prints a formatted console report of key statistics from mc\_price\_simulation() output, including CLT convergence and cross-validation estimates.

**Usage**

```
mc_statistics(mc_obj)
```

**Arguments**

mc\_obj            Output from mc\_price\_simulation().

**Value**

Invisibly returns mc\_obj.

---

minimize_cvar	<i>Minimise Portfolio CVaR (Multi-Restart)</i>
---------------	--

---

**Description**

Uses the Rockafellar-Uryasev linear programming reformulation of CVaR minimisation with multiple random restarts via `stats::optim`.

**Usage**

```
minimize_cvar(
  returns,
  alpha = 0.95,
  target_ret = NULL,
  allow_short = FALSE,
  freq = 252,
  n_restarts = 5
)
```

**Arguments**

returns	Numeric matrix (T x N) of asset returns.
alpha	Confidence level. Default 0.95.
target_ret	Optional numeric. Minimum target return constraint.
allow_short	Logical. Default FALSE.
freq	Integer. Periods per year. Default 252.
n_restarts	Integer. Number of random restarts. Default 5.

**Value**

A list: weights, CVaR, VaR, return, risk.

**Examples**

```
set.seed(1)
R <- matrix(rnorm(300 * 5, 0.0004, 0.011), 300, 5)
colnames(R) <- paste0("Asset", 1:5)
result <- minimize_cvar(R, alpha = 0.95)
result$CVaR
```

---

`min_variance_portfolio`*Global Minimum Variance Portfolio*

---

**Description**

Solves the unconstrained (or long-only) minimum-variance QP problem. Returns and risk are annualised.

**Usage**

```
min_variance_portfolio(returns, freq = 252, allow_short = FALSE)
```

**Arguments**

<code>returns</code>	Numeric matrix or xts of asset returns (T x N).
<code>freq</code>	Integer. Periods per year. Default 252.
<code>allow_short</code>	Logical. Allow short selling? Default FALSE.

**Value**

A list: weights, return, risk, sharpe, method.

**Examples**

```
set.seed(7)
R <- matrix(rnorm(252 * 6, 0.0002, 0.01), 252, 6)
colnames(R) <- paste0("A", 1:6)
mv <- min_variance_portfolio(R)
mv$risk
```

---

`money_flow_knn`*Money Flow Index + kNN signal*

---

**Description**

Money Flow Index + kNN signal

**Usage**

```
money_flow_knn(high, low, close, volume, k = 5, horizon = 1)
```

**Arguments**

high	High prices.
low	Low prices.
close	Close prices.
volume	Volumes.
k	Number of neighbors.
horizon	Forecast horizon for label.

**Value**

List with MFI series and kNN prediction.

---

monte_carlo_option	<i>Monte Carlo Option Pricing</i>
--------------------	-----------------------------------

---

**Description**

Prices European options via GBM simulation.

**Usage**

```
monte_carlo_option(
  S,
  K,
  time_to_expiry,
  r,
  sigma,
  n_sim = 1e+05,
  n_steps = 252,
  type = "call",
  seed = NULL
)
```

**Arguments**

S	Stock price.
K	Strike.
time_to_expiry	Time to expiry.
r	Risk-free rate.
sigma	Volatility.
n_sim	Integer. Number of simulations. Default 100000.
n_steps	Integer. Number of time steps. Default 252.
type	"call" or "put".
seed	Random seed for reproducibility. Default NULL.

**Value**

A list: price, std\_error, conf\_interval, paths (sample).

---

mvo\_efficient\_frontier

*Efficient Frontier - Lightweight Scan (Raw Scale)*

---

**Description**

Traces the efficient frontier by solving a target-return QP at each point. Works on raw (non-annualised) return scale. Prefer `compute_efficient_frontier()` for annualised, production-grade output.

**Usage**

```
mvo_efficient_frontier(
  returns,
  n = 50,
  rf = 0,
  allow_short = FALSE,
  target_returns = NULL
)
```

**Arguments**

returns	Matrix or data.frame of asset returns (T x N).
n	Integer. Number of frontier points. Default 50.
rf	Numeric. Risk-free rate for Sharpe. Default 0.
allow_short	Logical. Default FALSE.
target_returns	Optional numeric vector of specific target returns.

**Value**

A list: frontier (data.frame), weights (list), mu, Sigma.

**Examples**

```
set.seed(5)
R <- matrix(rnorm(300 * 4, 0.0003, 0.01), 300, 4)
colnames(R) <- c("A", "B", "C", "D")
ef <- mvo_efficient_frontier(R, n = 30)
head(ef$frontier)
```

---

mvo_max_sharpe	<i>Maximum Sharpe Portfolio - Frontier Scan</i>
----------------	---

---

**Description**

Identifies the max-Sharpe portfolio by scanning the efficient frontier computed via `mvo_efficient_frontier()`. A simpler alternative to `max_sharpe_portfolio()` when a frontier object is already available.

**Usage**

```
mvo_max_sharpe(returns, rf = 0, allow_short = FALSE, n = 50)
```

**Arguments**

returns	Matrix or data.frame of asset returns (T x N).
rf	Numeric. Risk-free rate. Default 0.
allow_short	Logical. Default FALSE.
n	Integer. Number of frontier points. Default 50.

**Value**

A list: weights, mean (raw-scale), vol (raw-scale), sharpe.

**Examples**

```
set.seed(9)
R <- matrix(rnorm(300 * 4, 0.0003, 0.01), 300, 4)
colnames(R) <- c("A", "B", "C", "D")
mvo_max_sharpe(R)
```

---

mvo_min_variance	<i>Minimum Variance Portfolio - Lightweight (Raw Scale)</i>
------------------	---

---

**Description**

A simpler GMV solver that works on raw (non-annualised) return scale. Uses quadprog directly with a ridge-regularised covariance matrix. Prefer `min_variance_portfolio()` for annualised results.

**Usage**

```
mvo_min_variance(returns, allow_short = FALSE)
```

**Arguments**

returns            Matrix or data.frame of asset returns (T x N).  
allow\_short       Logical. Allow short selling? Default FALSE.

**Value**

A list: weights, mean (raw-scale), vol (raw-scale).

**Examples**

```
set.seed(3)
R <- matrix(rnorm(300 * 4, 0.0003, 0.01), 300, 4)
colnames(R) <- c("A", "B", "C", "D")
mvo_min_variance(R)
```

---

mvo\_summary

*MVO Summary*

---

**Description**

Prints a formatted summary of the max-Sharpe and min-variance portfolios derived from an efficient frontier object.

**Usage**

```
mvo_summary(ef_obj, risk_free = 0.02)
```

**Arguments**

ef\_obj            Output of compute\_efficient\_frontier().  
risk\_free        Numeric. Risk-free rate. Default 0.02.

**Value**

Invisibly returns a list with max\_sharpe and min\_variance rows.

---

optimize\_quotes\_gd      *Optimize Quoting Parameters via Gradient Descent*

---

### Description

Learns optimal market-making quoting parameters by minimising the mean squared error between a linear spread model and a target spread vector. The model is:  $spread = a + b \cdot vol + c \cdot |inventory| + d \cdot imbalance$ .

### Usage

```
optimize_quotes_gd(data, target_spread, learning_rate = 0.01, epochs = 1000)
```

### Arguments

data	A data.frame of historical book data containing columns volatility and imbalance (output of extract_features()).
target_spread	Numeric vector of ideal/observed spreads (length equal to nrow(data)).
learning_rate	Numeric. Gradient descent step size (alpha). Default 0.01.
epochs	Integer. Number of gradient descent iterations. Default 1000.

### Value

A named numeric vector of four optimised parameters: a (intercept), b (volatility), c (inventory), d (imbalance).

### Examples

```
set.seed(2)
book <- extract_features(simulate_orderbook(300))
target <- book$spread + rnorm(300, 0, 0.001)
optimize_quotes_gd(book, target, learning_rate = 0.005, epochs = 500)
```

---

option\_greeks      *Option Greeks (Black-Scholes Analytical)*

---

### Description

Option Greeks (Black-Scholes Analytical)

### Usage

```
option_greeks(S, K, time_to_expiry, r, sigma, type = "call", q = 0)
```

**Arguments**

S	Stock price.
K	Strike.
time_to_expiry	Time to expiry.
r	Risk-free rate.
sigma	Volatility.
type	"call" or "put".
q	Dividend yield. Default 0.

**Value**

Named numeric vector: Delta, Gamma, Theta, Vega, Rho.

**Examples**

```
option_greeks(100, 100, 1, 0.05, 0.20, "call")
```

---

option\_price\_simulation

*Option Price Simulation: CLT Demonstration (Large N)*

---

**Description**

Simulates option prices over a large grid of (mean, sd) parameters and demonstrates the Central Limit Theorem convergence.

**Usage**

```
option_price_simulation(  
  S = 100,  
  K = 100,  
  time_to_expiry = 1,  
  r = 0.05,  
  sigma_grid = c(0.1, 0.2, 0.3, 0.4),  
  n_sim_vec = c(100, 500, 1000, 5000, 10000),  
  type = "call",  
  n_rep = 200,  
  seed = 123  
)
```



---

performance\_summary    *Performance summary using PerformanceAnalytics*

---

**Description**

Performance summary using PerformanceAnalytics

**Usage**

```
performance_summary(returns)
```

**Arguments**

returns            Matrix/data.frame/xts of returns.

**Value**

PerformanceAnalytics table of annualized returns.

---

plot\_asset\_clusters    *Plot Asset Clusters*

---

**Description**

Plot Asset Clusters

**Usage**

```
plot_asset_clusters(cl_obj, type = "heatmap", title = "Asset Clustering")
```

**Arguments**

cl\_obj            Output from asset\_clustering().  
type            Character. "heatmap", "dendrogram", or "network". Default "heatmap".  
title            Character.

**Value**

A ggplot2 object or base R plot.

---

plot\_binomial\_tree      *Plot Binomial Tree (Small Trees)*

---

### Description

Visualises the stock price nodes for a binomial tree. Only recommended for  $n \leq 15$ .

### Usage

```
plot_binomial_tree(bt_obj, show = "stock", title = "Binomial Tree")
```

### Arguments

bt_obj	Output from binomial_tree_option() or american_option_binomial().
show	Character. "stock" or "option". Default "stock".
title	Character.

### Value

A ggplot2 object.

---

plot\_correlation\_heatmap  
*Plot Correlation Heatmap*

---

### Description

Plot Correlation Heatmap

### Usage

```
plot_correlation_heatmap(  
  cor_obj,  
  title = "Cross-Asset Correlation",  
  show_values = TRUE,  
  low_col = "#2166ac",  
  high_col = "#d6604d"  
)
```

### Arguments

cor_obj	Output from asset_correlation_matrix(), or a correlation matrix directly.
title	Character. Plot title.
show_values	Logical. Show numeric values in cells. Default TRUE.
low_col	Colour for low correlation.
high_col	Colour for high correlation.

**Value**

A ggplot2 object.

---

`plot_cvar_frontier`      *Plot CVaR Frontier*

---

**Description**

Plot CVaR Frontier

**Usage**

```
plot_cvar_frontier(cvar_obj, title = "CVaR-Efficient Frontier")
```

**Arguments**

<code>cvar_obj</code>	Output from <code>cvar_frontier()</code> .
<code>title</code>	Character. Plot title.

**Value**

A ggplot2 object.

---

`plot_efficient_frontier`  
*Plot the Efficient Frontier*

---

**Description**

Visualises the efficient frontier with optional overlays for max-Sharpe and min-variance portfolios, and individual asset positions.

**Usage**

```
plot_efficient_frontier(  
  ef_obj,  
  highlight_portfolios = TRUE,  
  risk_free = 0.02,  
  show_assets = TRUE,  
  title = "Efficient Frontier"  
)
```

**Arguments**

ef_obj	Output from compute_efficient_frontier().
highlight_portfolios	Logical. Overlay max-Sharpe and min-var portfolios. Default TRUE.
risk_free	Numeric. Risk-free rate. Default 0.02.
show_assets	Logical. Show individual assets. Default TRUE.
title	Character. Plot title.

**Value**

A ggplot2 object.

---

plot_embedding	<i>Plot 2D Embedding (t-SNE or UMAP)</i>
----------------	--

---

**Description**

Plot 2D Embedding (t-SNE or UMAP)

**Usage**

```
plot_embedding(
  embed_obj,
  labels = NULL,
  title = "2D Embedding",
  method = "t-SNE"
)
```

**Arguments**

embed_obj	Output from portfolio_tsne() or portfolio_umap().
labels	Optional factor / character vector for colouring. Default NULL.
title	Character.
method	Character. Label for subtitle. Default "t-SNE".

**Value**

A ggplot2 object.

---

plot\_gd\_convergence     *Plot Gradient Descent Convergence*

---

**Description**

Plot Gradient Descent Convergence

**Usage**

```
plot_gd_convergence(gd_obj, title = "Gradient Descent Convergence")
```

**Arguments**

gd_obj	Output from <code>gd_min_variance()</code> or <code>gd_max_sharpe()</code> .
title	Character.

**Value**

A ggplot2 object.

---

plot\_mc\_paths     *Plot Monte Carlo Paths*

---

**Description**

Plots a random sample of simulated price or cumulative-return paths. Accepts output from either `gbm_simulation()` (App 2) or `mc_price_simulation()` (App 2), making it usable across both apps.

**Usage**

```
plot_mc_paths(mc_obj, n_show = 200, title = "Monte Carlo Price Paths")
```

**Arguments**

mc_obj	Output from <code>gbm_simulation()</code> or <code>mc_price_simulation()</code> .
n_show	Integer. Maximum number of paths to display. Default 200.
title	Character. Plot title.

**Value**

A ggplot2 object.

---

plot\_option\_simulation

*Plot Monte Carlo Option Paths*

---

### Description

Plot Monte Carlo Option Paths

### Usage

```
plot_option_simulation(mc_obj, n_paths = 100, title = "Monte Carlo GBM Paths")
```

### Arguments

mc_obj	Output from monte_carlo_option().
n_paths	Integer. Number of paths to show. Default 100.
title	Character.

### Value

A ggplot2 object.

---

plot\_pca\_biplot

*Plot PCA Biplot*

---

### Description

Plot PCA Biplot

### Usage

```
plot_pca_biplot(
  pca_obj,
  pc_x = 1,
  pc_y = 2,
  colour_by = NULL,
  title = "PCA Biplot"
)
```

### Arguments

pca_obj	Output from portfolio_pca().
pc_x	Integer. PC on x-axis. Default 1.
pc_y	Integer. PC on y-axis. Default 2.
colour_by	Optional numeric vector to colour observations by. Default NULL.
title	Character.

**Value**

A ggplot2 object.

---

plot_regimes	<i>Plot Market Regimes</i>
--------------	----------------------------

---

**Description**

Plots a time series of returns coloured by detected regime.

**Usage**

```
plot_regimes(regime_obj, dates = NULL, title = "Market Regime Detection")
```

**Arguments**

regime_obj	Output from kmeans_regime() or em_regime().
dates	Optional Date/POSIXct vector. Uses index if xts.
title	Character. Plot title.

**Value**

A ggplot2 object.

---

plot_risk_contribution	<i>Plot Risk Contributions</i>
------------------------	--------------------------------

---

**Description**

Visualises portfolio weights alongside risk contributions using a bar chart (side-by-side) or a pie chart.

**Usage**

```
plot_risk_contribution(
  rp_obj,
  type = c("bar", "pie"),
  title = "Risk Contribution"
)
```

**Arguments**

rp_obj	Output from equal_risk_contribution() or risk_parity_portfolio().
type	Character. "bar" (default) or "pie".
title	Character. Plot title.

**Value**

A ggplot2 object.

---

portfolio\_asset\_clustering

*Portfolio Asset Clustering (Extended)*

---

**Description**

Clusters portfolio assets using k-means, hierarchical, or EM methods on correlation-transformed distance in PCA-reduced space. Returns rich statistics including per-cluster summary, correlation matrix, and dendrogram. For the general-purpose version see [asset\\_clustering](#).

**Usage**

```
portfolio_asset_clustering(
  returns,
  k = 3,
  method = "kmeans",
  n_pca = 5,
  seed = 42,
  freq = 252
)
```

**Arguments**

returns	Numeric matrix (T x N) of asset returns.
k	Integer. Number of clusters. Default 3.
method	Character. "kmeans", "hierarchical", or "em". Default "kmeans".
n_pca	Integer. PCA dims before clustering. Default 5.
seed	Integer. Default 42.
freq	Integer. Default 252.

**Value**

A list: labels, cluster\_stats, dendrogram, centers, cor\_matrix.

**Examples**

```
set.seed(42)
R <- matrix(rnorm(300 * 8, 0.0002, 0.01), 300, 8)
colnames(R) <- paste0("Asset", 1:8)
cl <- portfolio_asset_clustering(R, k = 3)
cl$cluster_stats
```

---

portfolio\_clustering    *Portfolio Clustering (full pipeline)*

---

**Description**

Runs the complete asset clustering pipeline including PCA reduction, k-means clustering, and regime assignment.

**Usage**

```
portfolio_clustering(returns, k = 3, freq = 252, plot_all = TRUE, seed = 42)
```

**Arguments**

returns	Numeric matrix of returns.
k	Integer. Clusters. Default 3.
freq	Integer. Default 252.
plot_all	Logical. Return all plots. Default TRUE.
seed	Integer. Default 42.

**Value**

A comprehensive list with clustering, plots, and statistics.

---

portfolio\_cvar    *Portfolio CVaR (Expected Shortfall)*

---

**Description**

Computes the Conditional Value-at-Risk (CVaR / Expected Shortfall) for a given weight vector using historical simulation.

**Usage**

```
portfolio_cvar(  
  returns,  
  weights,  
  alpha = 0.95,  
  method = c("historical", "parametric", "monte_carlo"),  
  n_sim = 10000  
)
```

**Arguments**

returns	Numeric matrix (T x N) of asset returns.
weights	Numeric vector of portfolio weights (length N).
alpha	Confidence level (e.g. 0.95). Default 0.95.
method	Character. "historical", "parametric", or "monte_carlo".
n_sim	Integer. Number of MC simulations (if method = "monte_carlo").

**Value**

A named numeric vector: VaR, CVaR (both as positive loss figures).

**Examples**

```
set.seed(42)
R <- matrix(rnorm(500 * 4, 0.0003, 0.012), 500, 4)
w <- c(0.25, 0.25, 0.25, 0.25)
portfolio_cvar(R, w, alpha = 0.95)
```

---

portfolio\_pca

*Portfolio PCA Analysis*

---

**Description**

Performs Principal Component Analysis on asset returns. Identifies dominant factors, factor loadings, and variance explained.

**Usage**

```
portfolio_pca(returns, scale_ = TRUE, n_comp = NULL, freq = 252)
```

**Arguments**

returns	Numeric matrix (T x N) of asset returns.
scale_	Logical. Standardise returns before PCA. Default TRUE.
n_comp	Integer. Number of components to retain. Default NULL (all).
freq	Integer. Periods per year. Default 252.

**Value**

A list: pca\_obj, loadings, scores, var\_explained, cumulative\_var.

**Examples**

```
set.seed(42)
R <- matrix(rnorm(300 * 8, 0.0002, 0.01), 300, 8)
colnames(R) <- paste0("Asset", 1:8)
pca <- portfolio_pca(R)
pca$var_explained
```

---

portfolio\_performance *Portfolio Performance Summary*

---

**Description**

Computes key performance metrics for a portfolio.

**Usage**

```
portfolio_performance(returns, weights = NULL, risk_free = 0.02, freq = 252)
```

**Arguments**

returns	Numeric vector of portfolio returns.
weights	Optional weight vector (for display).
risk_free	Numeric. Risk-free rate. Default 0.02.
freq	Integer. Periods per year. Default 252.

**Value**

A named list of performance metrics.

**Examples**

```
set.seed(1)
r <- rnorm(252, 0.0004, 0.012)
portfolio_performance(r, risk_free = 0.02)
```

---

portfolio_tsne	<i>Portfolio t-SNE Embedding</i>
----------------	----------------------------------

---

### Description

Reduces high-dimensional return data to 2D using t-SNE. Useful for visualising clusters in asset return space.

### Usage

```
portfolio_tsne(  
  returns,  
  perplexity = 30,  
  n_iter = 1000,  
  dims = 2,  
  scale_ = TRUE,  
  seed = 42,  
  use_pca_first = TRUE,  
  pca_dims = 20  
)
```

### Arguments

returns	Numeric matrix (T x N) of returns.
perplexity	Numeric. t-SNE perplexity. Default 30.
n_iter	Integer. t-SNE iterations. Default 1000.
dims	Integer. Output dimensions (2 or 3). Default 2.
scale_	Logical. Standardise inputs. Default TRUE.
seed	Integer. Default 42.
use_pca_first	Logical. Pre-reduce with PCA. Default TRUE.
pca_dims	Integer. PCA dims before t-SNE. Default 20.

### Value

A list: embedding (T x dims), perplexity, note.

### Examples

```
set.seed(42)  
R <- matrix(rnorm(300 * 10, 0.0002, 0.01), 300, 10)  
ts <- portfolio_tsne(R, perplexity = 30)  
head(ts$embedding)
```

---

portfolio_umap	<i>Portfolio UMAP Embedding</i>
----------------	---------------------------------

---

**Description**

Reduces returns matrix to 2D using UMAP.

**Usage**

```
portfolio_umap(  
  returns,  
  n_neighbors = 15,  
  min_dist = 0.1,  
  dims = 2,  
  scale_ = TRUE,  
  seed = 42  
)
```

**Arguments**

returns	Numeric matrix of returns.
n_neighbors	Integer. UMAP neighbours. Default 15.
min_dist	Numeric. UMAP min_dist. Default 0.1.
dims	Integer. Output dimensions. Default 2.
scale_	Logical. Default TRUE.
seed	Integer. Default 42.

**Value**

A list: embedding, config.

---

predict_regime_knn	<i>Predict Regime using kNN</i>
--------------------	---------------------------------

---

**Description**

Classifies a new order book snapshot into one of the regimes learned by `cluster_book_kmeans()` using k-nearest-neighbour classification.

**Usage**

```
predict_regime_knn(train_data, new_snapshot, k = 5)
```

**Arguments**

`train_data` A data.frame of historical snapshots that includes a regime column (i.e. output of `cluster_book_kmeans()`\$data).

`new_snapshot` A single-row data.frame containing the same feature columns as `train_data`.

`k` Integer. Number of nearest neighbours. Default 5.

**Value**

A factor of length 1 with the predicted regime label.

**Examples**

```
set.seed(3)
book <- extract_features(simulate_orderbook(500))
res <- cluster_book_kmeans(book, centers = 3)
pred <- predict_regime_knn(res$data, res$data[1, ], k = 5)
```

---

price\_option\_binomial *Binomial Tree Option Pricing (European / American)*

---

**Description**

Prices European or American options using the Cox-Ross-Rubinstein (CRR) binomial tree model.

**Usage**

```
price_option_binomial(
  S0,
  K,
  r,
  sigma,
  time_to_maturity = 1,
  n_steps = 100,
  type = c("call", "put"),
  american = FALSE
)
```

**Arguments**

`S0` Spot price.

`K` Strike price.

`r` Risk-free rate (annualised).

`sigma` Volatility (annualised).

`time_to_maturity` Time to maturity in years. Default 1.

n_steps	Number of binomial tree steps. Default 100.
type	Character. "call" or "put".
american	Logical. TRUE for American option. Default FALSE.

**Value**

Scalar option price.

**Examples**

```
price_option_binomial(S0 = 100, K = 100, r = 0.05, sigma = 0.2)
price_option_binomial(S0 = 100, K = 100, r = 0.05, sigma = 0.2,
                      american = TRUE, type = "put")
```

---

price_option_mc	<i>Monte Carlo European Option Pricing</i>
-----------------	--

---

**Description**

Prices a European call or put option using Monte Carlo simulation under risk-neutral GBM dynamics.

**Usage**

```
price_option_mc(
  S0,
  K,
  r,
  sigma,
  time_to_maturity = 1,
  n_sims = 10000,
  type = c("call", "put"),
  seed = NULL
)
```

**Arguments**

S0	Spot price.
K	Strike price.
r	Risk-free rate (annualised).
sigma	Volatility (annualised).
time_to_maturity	Time to maturity in years. Default 1.
n_sims	Number of simulations. Default 10000.
type	Character. "call" or "put".
seed	Integer random seed. Default NULL.

**Value**

A list with elements:

price	Discounted Monte Carlo option price.
std_error	Standard error of the price estimate.

**Examples**

```
price_option_mc(S0 = 100, K = 100, r = 0.05, sigma = 0.2, seed = 1)
```

---

regime_statistics	<i>Regime Statistics</i>
-------------------	--------------------------

---

**Description**

Computes per-regime return, volatility, Sharpe, and duration statistics.

**Usage**

```
regime_statistics(regime_obj)
```

**Arguments**

regime_obj	Output from kmeans_regime() or em_regime().
------------	---

**Value**

A data.frame with per-regime statistics.

---

risk_contribution	<i>Compute Risk Contributions</i>
-------------------	-----------------------------------

---

**Description**

Computes the marginal and percentage risk contribution of each asset to total portfolio volatility.

**Usage**

```
risk_contribution(weights, cov_matrix)
```

**Arguments**

weights	Numeric weight vector (length N).
cov_matrix	N x N covariance matrix.

**Value**

A data.frame with columns: Asset, Weight, MargRC, RiskContrib, PercRC.

**Examples**

```
Sig <- matrix(c(0.04, 0.02, 0.02, 0.09), 2, 2)
risk_contribution(c(0.5, 0.5), Sig)
```

---

risk\_parity\_portfolio *Risk Parity Portfolio (Convenience Wrapper)*

---

**Description**

Alias for equal\_risk\_contribution() with named budget support. Use this when you want to specify a custom (possibly named) risk budget.

**Usage**

```
risk_parity_portfolio(returns, budget = NULL, freq = 252, ...)
```

**Arguments**

returns	Numeric matrix (T x N) of asset returns.
budget	Named numeric vector. Risk budget per asset (sums to 1). Default is equal budget.
freq	Integer. Periods per year. Default 252.
...	Additional arguments passed to equal_risk_contribution().

**Value**

Same list as equal\_risk\_contribution().

**Examples**

```
set.seed(1)
R <- matrix(rnorm(300 * 3, 0.0003, 0.01), 300, 3)
colnames(R) <- c("Equity", "Bond", "Gold")
rp <- risk_parity_portfolio(R, budget = c(0.5, 0.3, 0.2))
rp$weights
```

---

risk\_parity\_weights     *Risk Parity Weights — Fast Iterative Solver*

---

### Description

Computes equal-risk-contribution weights directly from a covariance matrix using a simple iterative proportional scaling algorithm. This is a lightweight alternative to `equal_risk_contribution()` when only a covariance matrix is available (no return series needed).

### Usage

```
risk_parity_weights(covmat, max_iter = 1000, tol = 1e-08)
```

### Arguments

<code>covmat</code>	Square numeric covariance matrix (N x N).
<code>max_iter</code>	Integer. Maximum number of iterations. Default 1000.
<code>tol</code>	Numeric. Convergence tolerance on normalised risk contribution deviation. Default 1e-8.

### Value

A list with elements:

<code>weights</code>	Named numeric vector of risk-parity weights.
<code>iterations</code>	Number of iterations until convergence.

### Examples

```
Sig <- matrix(c(0.04, 0.02, 0.02, 0.09), 2, 2)
colnames(Sig) <- rownames(Sig) <- c("A", "B")
risk_parity_weights(Sig)
```

---

rolling\_correlation     *Rolling Correlation*

---

### Description

Computes rolling pairwise correlations between two assets over time.

### Usage

```
rolling_correlation(r1, r2, window = 60, method = "pearson")
```

**Arguments**

r1	Numeric vector. Returns of asset 1.
r2	Numeric vector. Returns of asset 2.
window	Integer. Rolling window. Default 60.
method	Character. "pearson" etc. Default "pearson".

**Value**

A numeric vector of rolling correlations.

---

rolling\_cv\_forecast    *Rolling cross-validation for return forecasts*

---

**Description**

Rolling cross-validation for return forecasts

**Usage**

```
rolling_cv_forecast(
  returns,
  window = 252,
  horizon = 21,
  step = 21,
  estimator = c("mean", "median")
)
```

**Arguments**

returns	Matrix/data.frame of returns.
window	Training window length.
horizon	Forecast horizon.
step	Step size between folds.
estimator	"mean" or "median".

**Value**

Data frame with fold metrics.

---

run\_quantportr\_app      *Launch the QuantPortR Interactive Dashboard*

---

### Description

Launch the QuantPortR Interactive Dashboard

### Usage

```
run_quantportr_app(host = "127.0.0.1", port = 3838, launch.browser = TRUE)
```

### Arguments

host                    Character. Host address. Default "127.0.0.1".  
port                    Integer. Port to listen on. Default 3838.  
launch.browser        Logical. Open browser on launch. Default TRUE.

### Value

No return value. Called for the side effect of launching a Shiny application. Internally this starts a Shiny app object and blocks the current R session until the app is stopped.

---

sampling\_distribution      *Sampling Distribution Demonstration*

---

### Description

Demonstrates properties of sampling distributions (mean, variance) through simulation, verifying theoretical results.

### Usage

```
sampling_distribution(  
  pop,  
  stat = "mean",  
  n_samples = c(10, 30, 50, 100, 200),  
  n_reps = 2000,  
  seed = 42  
)
```

### Arguments

pop                    Numeric vector. Population of returns.  
stat                    Character. "mean" or "variance". Default "mean".  
n\_samples            Integer. Sample sizes to test.  
n\_reps                Integer. Repetitions. Default 2000.  
seed                    Integer. Default 42.

**Value**

A list: results, plots, theoretical\_vs\_empirical.

---

scree_plot	<i>Scree Plot</i>
------------	-------------------

---

**Description**

Scree Plot

**Usage**

```
scree_plot(pca_obj, title = "Scree Plot (PCA Variance Explained)")
```

**Arguments**

pca_obj	Output from portfolio_pca().
title	Character.

**Value**

A ggplot2 object.

---

simulate_gbm_paths	<i>Simulate GBM Price Paths</i>
--------------------	---------------------------------

---

**Description**

Simulates stock price paths under Geometric Brownian Motion using the exact log-normal increments. Returns a matrix with rows as time steps and columns as simulations (compatible with App 1 option pricing workflow).

**Usage**

```
simulate_gbm_paths(  
  S0,  
  mu,  
  sigma,  
  time_horizon = 1,  
  n_steps = 252,  
  n_sims = 1000,  
  seed = NULL  
)
```

**Arguments**

<code>S0</code>	Initial price.
<code>mu</code>	Annual drift.
<code>sigma</code>	Annual volatility.
<code>time_horizon</code>	Time horizon in years. Default 1.
<code>n_steps</code>	Number of time steps. Default 252.
<code>n_sims</code>	Number of simulation paths. Default 1000.
<code>seed</code>	Integer random seed. Default NULL.

**Value**

Numeric matrix of simulated prices (`n_steps+1` rows x `n_sims` cols).

**Examples**

```
paths <- simulate_gbm_paths(S0 = 100, mu = 0.08, sigma = 0.2, seed = 42)
dim(paths) # 253 x 1000
```

---

simulate\_orderbook     *Simulate a Basic Order Book*

---

**Description**

Generates synthetic limit order book (LOB) snapshots over time by combining a random-walk mid-price with Poisson-distributed depth and a stochastic bid-ask spread.

**Usage**

```
simulate_orderbook(n_steps = 1000, p0 = 100)
```

**Arguments**

<code>n_steps</code>	Integer. Number of time steps to simulate. Default 1000.
<code>p0</code>	Numeric. Initial mid-price. Default 100.

**Value**

A data.frame with columns `time`, `mid_price`, `bid`, `ask`, `bid_depth`, and `ask_depth`.

**Examples**

```
set.seed(42)
book <- simulate_orderbook(n_steps = 500, p0 = 100)
head(book)
```

---

unbiasedness_check	<i>Unbiasedness Check</i>
--------------------	---------------------------

---

**Description**

Tests whether a list of estimates is unbiased w.r.t. a true value.

**Usage**

```
unbiasedness_check(estimates, true_val, tol = 0.05)
```

**Arguments**

estimates	Numeric vector of estimates.
true_val	Numeric. True parameter value.
tol	Relative tolerance. Default 0.05 (5%).

**Value**

A list: bias, rel\_bias, is\_unbiased.

---

var_cvar	<i>VaR and CVaR analysis</i>
----------	------------------------------

---

**Description**

VaR and CVaR analysis

**Usage**

```
var_cvar(
  returns,
  alpha = 0.95,
  method = c("historical", "gaussian"),
  portfolio_weights = NULL
)
```

**Arguments**

returns	Matrix/data.frame of returns.
alpha	Confidence level.
method	"historical" or "gaussian".
portfolio_weights	Optional weights to compute portfolio risk.

**Value**

Data frame with VaR and CVaR.

---

var\_cvar\_analysis      *Full VaR / CVaR Portfolio Analysis*

---

**Description**

Computes VaR and CVaR using three methods and returns a comprehensive report.

**Usage**

```
var_cvar_analysis(  
  returns,  
  weights,  
  alphas = c(0.9, 0.95, 0.99),  
  n_sim = 50000,  
  freq = 252  
)
```

**Arguments**

returns	Numeric matrix (T x N) of asset returns.
weights	Numeric vector of portfolio weights.
alphas	Numeric vector of confidence levels. Default c(0.90, 0.95, 0.99).
n_sim	Integer. MC simulations. Default 50000.
freq	Integer. Periods per year. Default 252.

**Value**

A list with tables and ggplot objects.

# Index

## \* datasets

- example\_prices, 23
- american\_option\_binomial, 4
- annualize\_returns, 5
- asset\_clustering, 5, 54
- asset\_correlation, 6
- asset\_correlation\_matrix, 7
- binomial\_tree\_option, 8
- bootstrap\_returns, 8
- bs\_option\_price, 9
- calc\_returns, 10
- clt\_demonstration, 11
- clt\_pnl\_ci, 12
- clt\_sample\_means, 12
- cluster\_book\_kmeans, 13
- cluster\_summary, 13
- compute\_efficient\_frontier, 14
- consistency\_check, 15
- cross\_asset\_analysis, 15
- cross\_validate\_portfolio, 16
- cvar\_frontier, 17
- cvar\_minimize, 17
- detect\_regimes, 18
- download\_prices, 19
- em\_clustering, 20
- em\_regime, 21
- embedding\_2d, 20
- equal\_risk\_contribution, 22
- example\_prices, 23
- extract\_features, 23
- fetch\_yahoo\_prices, 24
- format\_weights, 24
- gaussian\_mixture\_em, 25
- gbm\_simulation, 25
- gd\_max\_sharpe, 27
- gd\_min\_variance, 28
- get\_example\_prices, 29
- get\_returns, 29
- gradient\_descent, 30
- gradient\_descent\_portfolio, 30
- kmeans\_regime, 31
- knn\_classify, 32
- knn\_money\_flow, 33
- knn\_predict, 34
- market\_regime\_kmeans, 34
- max\_sharpe\_portfolio, 35
- mc\_price\_simulation, 36
- mc\_return\_distribution, 37
- mc\_statistics, 37
- min\_variance\_portfolio, 39
- minimize\_cvar, 38
- money\_flow\_knn, 39
- monte\_carlo\_option, 40
- mvo\_efficient\_frontier, 41
- mvo\_max\_sharpe, 42
- mvo\_min\_variance, 42
- mvo\_summary, 43
- optimize\_quotes\_gd, 44
- option\_greeks, 44
- option\_price\_simulation, 45
- option\_price\_summary, 46
- performance\_summary, 47
- plot\_asset\_clusters, 47
- plot\_binomial\_tree, 48
- plot\_correlation\_heatmap, 48
- plot\_cvar\_frontier, 49
- plot\_efficient\_frontier, 49
- plot\_embedding, 50
- plot\_gd\_convergence, 51
- plot\_mc\_paths, 51

plot\_option\_simulation, 52  
plot\_pca\_biplot, 52  
plot\_regimes, 53  
plot\_risk\_contribution, 53  
portfolio\_asset\_clustering, 54  
portfolio\_clustering, 55  
portfolio\_cvar, 55  
portfolio\_pca, 56  
portfolio\_performance, 57  
portfolio\_tsne, 58  
portfolio\_umap, 59  
predict\_regime\_knn, 59  
price\_option\_binomial, 60  
price\_option\_mc, 61

regime\_statistics, 62  
risk\_contribution, 62  
risk\_parity\_portfolio, 63  
risk\_parity\_weights, 64  
rolling\_correlation, 64  
rolling\_cv\_forecast, 65  
run\_quantportr\_app, 66

sampling\_distribution, 66  
scree\_plot, 67  
simulate\_gbm\_paths, 67  
simulate\_orderbook, 68

unbiasedness\_check, 69

var\_cvar, 69  
var\_cvar\_analysis, 70