ETF Portfolio Optimization

Release v1

Maxime Solere

Aug 29, 2025

CONTENTS:

Add your content using `reStructuredText` syntax. See the reStructuredText documentation for details.

CHAPTER

ONE

CODE

1.1 backtest module

Rolling backtest utilities for an optimized portfolio.

This module defines *Backtest*, which:

- Walks forward through monthly dates and re-optimizes the portfolio at each step using in-sample data up to that date (via `Portfolio` and *Opti* in *static* mode with an in-sample cutoff).
- Optionally smooths the resulting weight paths.
- Computes out-of-sample (test) returns from the held-out period.
- Produces diagnostic plots (equity curve, weights stack, and performance attribution) as Dash-ready images while saving PNGs to disk.

Notes

- The train/test split is controlled by *Backtest.ratio_train_test* (default `17/20` i.e., 85% train, 15% test).
- Re-optimization loops over months from the cutoff to the end of the data, constructing a fresh `Portfolio` with `static=True` and `backtest=<current_date>` at each step.

class backtest.**Backtest**(*opti*)

Bases: `object`

Rolling re-optimization backtest.

1.1.1 Class Attributes

ratio_train_test
[float] Fraction of the sample used for training/in-sample (default: `17/20`).

param opti
A fully-initialized optimizer instance whose portfolio defines the configuration (universe, risk, currency, shorting, etc.) and whose optimized constituents (`Opti.optimum`) seed the initial `to_consider` set for plots.

type opti
Opti

opti
The reference optimizer object passed in.

Type
Opti

`portfolio`

Convenience alias to `opti.portfolio`.

Type
Portfolio

`to_consider`

The keys of `opti.optimum`; used to focus attribution plots.

Type
dict_keys[str]

`w_opt`

Time-indexed weights per ticker for the walk-forward re-optimizations.

Type
pandas.DataFrame | None

`returns`

Out-of-sample (test) portfolio returns.

Type
pandas.Series | None

`n`

Number of rows (time points) in the underlying NAV table.

Type
int | None

`cutoff`

Index position separating train/test based on *ratio_train_test*.

Type
int | None

`index`

Copy of the underlying `DatetimeIndex` for iteration.

Type
list[pandas.Timestamp] | None

`returns_decomp`

Per-asset contributions to test-period returns (weights × returns).

Type
pandas.DataFrame | None

`get_returns()`

Compute out-of-sample (test) returns and decomposition.

- Select the test-period rows from the full return matrix using `data.Data.get_test_data_backtest()` with the cutoff timestamp.
- Multiply by time-aligned weights to obtain per-asset contributions.
- Sum across columns to obtain the total test return series.

Side Effects

Sets *returns_decomp* and *returns*.

returns
None.

rtype
None

parse_data()

Build rolling optimal weights by re-optimizing through time.

Steps

1. Determine the train/test split using *ratio_train_test*.
2. For each test date `t` (from cutoff to end):
   - Create a new `Portfolio` with `static=True` and `backtest=index[t]` so that all data are truncated to in-sample up to that date.
   - Run *Opti* on that portfolio and store the optimal weight vector into *w_opt* at timestamp `index[t]`.

Side Effects

Sets *n*, *cutoff*, *index*, and fills *w_opt*.

returns
None.

rtype
None

plot_backtest()

Plot the backtest equity curve vs. benchmark and risk-free leg.

The title includes annualized performance (p.a.) and maximum drawdown over the test window.

Returns
Dash image component with the figure embedded.

Return type
dash.html.Img

plot_perf_attrib()

Plot cumulative performance attribution for selected tickers.

Uses the per-asset return contributions in *returns_decomp* and accumulates them through time, plotted in percent.

Returns
Dash image component with the figure embedded.

Return type
dash.html.Img

plot_weights()

Plot a stacked weight history for the most material tickers.

Heuristic

Start with the *current* optimized constituents (*to_consider*). Greedily add other tickers by descending average weight until the cumulative mean weight of the plotted set reaches at least 90%.

returns
Dash image component with the figure embedded.

rtype
dash.html.Img

`ratio_train_test = 0.85`

`smoothen_weights()`

Apply simple exponential smoothing (2/3 previous + 1/3 current).

This can reduce churn in the weights before computing test returns.

Side Effects

Overwrites *w_opt* with the smoothed series.

returns
None.

rtype
None

1.2 dashboard module

Dash application to build, optimize, backtest, and rebalance an ETF portfolio.

This module wires together the core components:

- `portfolio.Portfolio` — data preparation, universe pruning, objective.
- *opti.Opti* — portfolio optimization and performance/diagnostic plots.
- *rebalancer.Rebalancer* — converts optimal weights to a rebalance plan.
- *backtest.Backtest* — rolling walk-forward re-optimization backtest.
- *exposure.Exposure* — exposure breakdown charts.

The *Dashboard* class builds a small UI with: - Risk, currency, and shorting controls, - Cash and current holdings inputs, - Buttons to create an optimal portfolio, show exposures, rebalance, and run a backtest, - A "crypto Sharpe" helper that shows tangency-portfolio weights for a small crypto universe.

Notes

- The app uses multiple Dash callbacks. Each callback resets its triggering button's `n_clicks` to `0` after use to allow retriggering.
- Images are returned as Dash-ready components via *opti.Opti.save_fig_as_dash_img()*.

`class dashboard.Dashboard(static=False)`

Bases: `Dash`

Minimal Dash UI for ETF portfolio workflows.

Parameters
static (*bool, optional*) – If `True`, downstream components load cached CSVs instead of downloading fresh data (default: `False`).

`static`

Passed to `portfolio.Portfolio` / `data.Data`.

Type
bool

`layout_functions`

Functions that return UI chunks used to compose the main layout.

Type
list[callable]

`main_div`

Flat list of components used as children for the top-level `html.Div`.

Type
list | None

`risk`

Risk level from the UI.

Type
int | None

`currency`

Selected base currency from the UI.

Type
str | None

`allow_short`

Checklist values; empty list means no shorting (long-only).

Type
list[str] | None

`cash_sgd`

Cash input (denominated in the selected currency).

Type
float | None

`holdings`

Mapping of user-provided holdings (ticker -> value in base currency).

Type
dict[str, float] | None

`portfolio`

Portfolio object created after clicking "Create Portfolio".

Type
Portfolio | None

`opti`

Optimizer object tied to *`portfolio`*.

Type
Opti | None

`backtest`

Backtest object created after clicking “Launch Backtest”.

Type
Backtest | None

`rebalancer`

Rebalancer object created after clicking “Rebalance”.

Type
Rebalancer | None

`exposure`

Exposure object created after clicking “Display exposure”.

Type
Exposure | None

`static button_create_backtest()`

Section to run and display a rolling backtest.

Returns
Header, button, and loading wrapper for backtest graphs.

Return type
list[dash.development.base_component.Component]

`static button_create_portfolio()`

Section to create and display an optimal portfolio.

Returns
Header, button, and a loading wrapper for result graphs.

Return type
list[dash.development.base_component.Component]

`static button_crypto()`

Section to show crypto tangency-portfolio weights.

Returns
Header, button, and table container.

Return type
list[dash.development.base_component.Component]

`static button_display_exposure()`

Section to render exposure breakdown charts.

Returns
Header, button, and graph container.

Return type
list[dash.development.base_component.Component]

`static button_holdings()`

Holdings input section with an “Add Holding” button.

Returns
Header, button, and container div.

Return type
list[dash.development.base_component.Component]

static button_rebalance()

Section to build and display a rebalance table.

Returns
Header, button, and result container.

Return type
list[dash.development.base_component.Component]

callbacks()

Register all Dash callbacks (inputs, buttons, and renderers).

Each nested function has its own docstring describing inputs/outputs.

Returns
None.

Return type
None

get_layout()

Compose the static layout sections from *layout_functions*.

Returns
None (sets layout).

Return type
None

static input_cash()

Cash input whose label reflects the selected currency.

Returns
Label and numeric input for cash.

Return type
list[dash.development.base_component.Component]

static radio_currency()

Dropdown for base currency selection.

Returns
H4 label and currency dropdown.

Return type
list[dash.development.base_component.Component]

static radio_risk()

Numeric input for risk level.

Returns
A label and numeric input for risk.

Return type
list[dash.development.base_component.Component]

static radio_short()

Checklist to allow/disallow shorting.

Returns
Checklist component; empty value implies long-only.

Return type
list[dash.development.base_component.Component]

`static text_title()`

Create the app title.

Returns
Title component list.

Return type
list[dash.html.H1]

1.3 data module

Data acquisition and preprocessing utilities for multi-asset portfolio work.

This module centers around `Data`, which downloads (or loads cached) time series for:

- Foreign exchange (FX) rates to convert assets into a chosen base currency.
- A risk-free rate proxy from `^IRX` (13-week T-bill), converted to a monthly rate.
- ETF NAV/Close series and derived simple, log, and excess returns.
- A total U.S. equity market proxy (VTI) for benchmarking.
- A simple long-only tangency portfolio over a small crypto universe.

It supports a *static* mode that reads/writes CSV caches under `data_dir_path` to avoid repeated network calls, and optional backtest truncation where series are sliced up to a specified date.

1.3.1 Dependencies

`yfinance`, `pandas`, `numpy`, `matplotlib`, and `scipy` are used for retrieval, manipulation, plotting, and optimization.

Examples

Create a dataset in EUR with cached files only:

```
>>> d = Data(currency="EUR", etf_list=["VWRA.L", "EUNA.L"], static=True)
```

Create a dataset in USD, download fresh data, and trim in-sample up to January 2020:

```
>>> d = Data(currency="USD", etf_list=["VT", "BND"], static=False, backtest="2020-01-01")
```

1.4 exposure module

Exposure breakdown plots for optimized portfolios.

This module defines *`Exposure`*, which produces pie charts showing the portfolio's composition across:

- Trading currencies (via ETF native currency and FX pseudo-tickers),
- Asset class,
- Equity sector,
- Bond type,
- Geography.

The underlying exposures are sourced from `opti.portfolio.data.exposure` and optimal weights from *Opti*. Figures are returned as Dash-ready `html.Img` elements using *opti.Opti.save_fig_as_dash_img()*.

class exposure.**Exposure**(*opti*)

Bases: `object`

Build exposure pie charts from an optimized portfolio.

Parameters

opti (Opti) – Optimizer instance providing:

- `optimum`: mapping `{ticker: weight}` of optimized weights.
- `portfolio.data.exposure`: pandas DataFrame with categorical exposure columns (e.g., `Asset Class`, `Stock Sector`, `Bond Type`, `Geography`).
- `portfolio.data.etf_currency`: mapping ticker → native trading currency.
- `portfolio.currency` and `Data.possible_currencies` for FX pseudo-tickers.

Attribute opti
Reference to the optimizer.

Attribute optimum
Optimized weight mapping used to aggregate exposures.

Attribute exposure_df
Table of categorical exposures (indexed by ticker).

plot_category()

Plot exposure by high-level asset class.

Returns
Dash image component, or `None` if there is no exposure.

Return type
dash.html.Img | None

plot_currency()

Plot exposure by trading currency (including FX pseudo-tickers).

1.4.1 Logic

- If a key in `optimum` is itself a currency code in `data.Data.possible_currencies`, treat it directly as currency exposure (FX pseudo-ticker).
- Otherwise, look up the ETF's native trading currency in `portfolio.data.etf_currency` and attribute the weight accordingly.

returns
Dash image component for the currency pie chart.

rtype
dash.html.Img

plot_geo()

Plot exposure by geography.

Returns
Dash image component, or `None` if there is no exposure.

Return type
dash.html.Img | None

`plot_other_exposure`(*name*)

Generic pie chart for an exposure category column.

The method aggregates optimized weights by the category in `exposure_df[name]` (e.g., `'Asset Class'`, `'Stock Sector'`, `'Bond Type'`, `'Geography'`). If the total weight for that category set is zero, returns `None`.

Parameters
name (*str*) – Column name in `exposure_df` to aggregate by.

Returns
Dash image component, or `None` if there is no exposure.

Return type
dash.html.Img | None

`plot_pie_chart`(*dico*, *title*)

Render a pie chart from a category-to-weight dictionary.

Zero-weight categories are removed. The figure is converted to a Dash `html.Img` via *opti.Opti.save_fig_as_dash_img()*.

Parameters

- **dico** (*dict[str, float]*) – Mapping from category label to (non-normalized) weight.
- **title** (*str*) – Chart title.

Returns
Dash image component for embedding in a layout.

Return type
dash.html.Img

`plot_sector`()

Plot exposure by equity sector.

Returns
Dash image component, or `None` if there is no exposure.

Return type
dash.html.Img | None

`plot_type`()

Plot exposure by bond type.

Returns
Dash image component, or `None` if there is no exposure.

Return type
dash.html.Img | None

1.5 main module

Application entry point for the Dash ETF Portfolio Optimizer.

This module launches the *dashboard.Dashboard* app. By default it starts the server with cached/static data reads enabled (`static=True`).

1.5.1 Usage

Run the module as a script to start the server:

```
python run.py
```

Or import and call *main()* from another module:

```
from run import main
main(debug=True)
```

main.**main**(*debug: bool = False*) → None

Launch the Dash application.

Parameters
debug (*bool*) – If True, enable Dash/Flask debug mode (auto-reload, extra logs).

Returns
None.

Return type
None

1.6 opti module

Optimization and plotting for portfolio weights.

This module defines *Opti*, a small helper that:

- Builds bounds and constraints for a portfolio optimization (long-only or long/short with L1 weight budget).
- Minimizes a user-provided mean–variance-style objective exposed by a Portfolio instance.
- Computes in-sample cumulative performance and a few diagnostic plots (allocation pie, cumulative vs. benchmark, contribution, and drawdown), returning each plot as a Dash-ready html.Img element while also saving PNGs to disk.

Notes

- The solver is SciPy's minimize with SLSQP by default.
- For long/short, the equality constraint is sum(|w|) = 1; for long-only, it is sum(w) = 1.

class opti.**Opti**(*portfolio*)

Bases: object

Portfolio optimizer and plotting utility.

1.6.1 Class Attributes

solver_method
[str] Optimization algorithm passed to scipy.optimize.minimize() (default: "SLSQP").

graph_dir_path
[pathlib.Path] Root directory where PNG plots will be saved.

param portfolio
A portfolio object exposing: * n (universe size), * allow_short (bool), * objective(w=...) (callable for minimization), * etf_list (tickers), * color_map (ticker -> HEX), * data with returns, spy, and rf_rate, * currency (base currency code), * name (label for titles).

type portfolio
Portfolio

`optimum`

Sparse weight mapping after thresholding small weights and renormalizing.

Type
dict[str, float] | None

`optimum_all`

Full weight vector (including zeros) as a mapping.

Type
dict[str, float] | None

`w_opt`

Optimized weight vector.

Type
numpy.ndarray | None

`constraints`

Nonlinear equality constraint(s) for the optimizer.

Type
list[dict] | None

`bounds`

Per-asset bounds, long-only or long/short per portfolio settings.

Type
list[tuple[float, float]] | None

`cumulative`

In-sample cumulative performance of the optimized portfolio.

Type
pandas.Series | None

`portfolio`

Reference to the provided portfolio object.

Type
Portfolio

`w0`

Starting point for optimization (uniform weights).

Type
numpy.ndarray

`static abs_sum`(*lst*)

L1 norm (sum of absolute values).

Parameters
lst (*list[float] | numpy.ndarray | tuple[float, ...]*) – Iterable of numbers.

Returns
Sum of absolute values.

Return type
float

`get_bounds()`

Build per-asset bounds based on shorting permission.

- If shorting is allowed: `(-1, 1)`.
- If long-only: `(0, 1)`.

Returns
None.

Return type
None

`get_constraints()`

Construct the weight-budget equality constraint.

- Long-only: enforce `sum(w) = 1`.
- Long/short: enforce `sum(|w|) = 1`.

Returns
None.

Return type
None

`get_cumulative()`

Compute in-sample cumulative performance for the optimized weights.

Uses simple returns from `self.portfolio.data.returns` and the sparse weight mapping in *optimum*.

Returns
None (sets *cumulative*).

Return type
None

`graph_dir_path = PosixPath('/Users/maximesolere/PycharmProjects/ETF/graphs')`

`optimize()`

Solve the portfolio optimization problem.

Minimizes `self.portfolio.objective(w=w)` under the configured bounds and equality constraint. Post-processes the solution by: * thresholding very small absolute weights (< 1%) to zero, then * renormalizing by the L1 norm so the budget equals 1.

Side Effects

Sets *w_opt*, *optimum_all*, and *optimum*. Prints a message if SciPy reports failure.

returns
None (updates instance attributes).

rtype
None

`plot_drawdown()`

Plot the portfolio drawdown curve (area below zero).

Returns
Dash image component for embedding in a layout.

Return type
dash.html.Img

`plot_in_sample()`

Plot in-sample cumulative performance vs. market proxy and RF leg.

The title includes the annualized performance (p.a.) and maximum drawdown computed from *cumulative*.

Returns
Dash image component for embedding in a layout.

Return type
dash.html.Img

`plot_optimum()`

Plot the optimized allocation as a pie chart.

Colors are pulled from `self.portfolio.color_map`. The image is saved under `graphs/<currency>/<name>- optimal_allocation.png` and also returned as a Dash image.

Returns
Dash image component for embedding in a layout.

Return type
dash.html.Img

`plot_weighted_perf()`

Plot in-sample performance attribution by constituent.

The contribution per asset is the weighted cumulative excess over 1 (in percent). Colors follow the portfolio color map.

Returns
Dash image component for embedding in a layout.

Return type
dash.html.Img

`static save_fig_as_dash_img(`*fig*, *output_path*`)`

Convert a Matplotlib figure to a Dash `html.Img` (and save to disk).

If `output_path` is not `None`, the PNG is written to that path. The function always returns an inline base64-encoded `html.Img` element.

Parameters

- **fig** (*matplotlib.figure.Figure*) – Matplotlib figure to serialize.
- **output_path** (*str | pathlib.Path | None*) – File path for saving the PNG (or `None`).

Returns
Dash image component with the figure embedded.

Return type
dash.html.Img

`solver_method = 'SLSQP'`

1.7 portfolio module

Portfolio construction utilities with correlation clustering and a mean–variance objective.

This module defines:

- *Info* — configuration and utilities (risk scaling, colors, ticker universe).
- `Portfolio` — data wiring and feature engineering over `Data`, including de-duplication of highly correlated tickers via hierarchical clustering and a convex mean–variance-style objective you can pass to optimizers.

The workflow is:

1. Instantiate `Portfolio` with a target risk level, currency, holdings, etc.
2. It loads market data through `data.Data`.
3. It removes too-new tickers (with missing history) and prunes clusters of highly correlated names, keeping the one with the best (lowest) objective value.
4. It exposes `Portfolio.objective`, a callable that computes `weight_cov * variance - mean_excess` for a weight vector, suitable for SLSQP/L-BFGS-B minimization.

Notes

- Correlation clustering uses average linkage on the distance matrix `1 - |corr|` with threshold `1 - threshold_correlation`.
- Colors are assigned deterministically from Matplotlib's `tab20` colormap, extended with FX pseudo-tickers for non-base currencies.

class portfolio.**Info**(*risk*, *cash*, *holdings*, *currency*, *allow_short*)

Bases: `object`

Shared portfolio information and utilities (risk scaling, color maps, universe).

1.7.1 Class Attributes

threshold_correlation
[float] Minimum absolute correlation to be considered the "same cluster". Used as `1 - threshold_correlation` on the correlation distance.

etf_list
[list[str]] Canonical ETF universe (deduplicated and sorted at import time).

name
[dict[int, str]] Human labels for discrete risk tiers (may be overridden per instance).

param risk
Discrete risk appetite (e.g., 1=low, 2=medium, 3=high). Drives *weight_cov*.

type risk
int

param cash
Available cash (used in `Portfolio.get_liquidity()`).

type cash
float

param holdings
Current positions as a mapping `{ticker: current_value}`. Optional.

type holdings
dict[str, float] | None

param currency
Base currency (one of `data.Data.possible_currencies`). Defaults to `"USD"` when `None`.

type currency
str | None

param allow_short
Whether shorting is conceptually allowed (does not alter logic here, but exposed for downstream optimizers).

type allow_short
bool

`color_map`

Mapping from ticker to HEX color for plotting (set by *get_color_map()*).

Type
dict[str, str] | None

`weight_cov`

Coefficient in the mean–variance objective (set by *get_weight_cov()*).

Type
float | None

`risk, cash, holdings, allow_short, currency`

Type
see parameters

`n`

Current universe size (set after *etf_list* finalization).

Type
int

```
etf_list = ['AGG', 'DGT', 'DIA', 'DVY', 'EEM', 'EFA', 'EPP', 'EWA', 'EWC', 'EWD',
'EWG', 'EWH', 'EWI', 'EWJ', 'EWL', 'EWM', 'EWN', 'EWP', 'EWQ', 'EWS', 'EWT', 'EWU',
'EWW', 'EWY', 'EWZ', 'EZA', 'EZU', 'FEZ', 'FXI', 'GLD', 'IBB', 'ICF', 'IDU', 'IEF',
'IEV', 'IGE', 'IGM', 'IGPT', 'IGV', 'IJH', 'IJJ', 'IJK', 'IJR', 'IJS', 'IJT',
'ILCB', 'ILCG', 'ILCV', 'ILF', 'IMCB', 'IMCG', 'IMCV', 'IOO', 'ISCB', 'ISCG',
'ISCV', 'ITOT', 'IUSG', 'IUSV', 'IVE', 'IVV', 'IVW', 'IWB', 'IWD', 'IWF', 'IWM',
'IWN', 'IWO', 'IWP', 'IWR', 'IWS', 'IWV', 'IXC', 'IXG', 'IXJ', 'IXN', 'IXP', 'IYC',
'IYE', 'IYF', 'IYG', 'IYH', 'IYJ', 'IYK', 'IYM', 'IYR', 'IYT', 'IYW', 'IYY', 'IYZ',
'LQD', 'MDY', 'OEF', 'ONEQ', 'PBW', 'PEJ', 'PEY', 'PJP', 'PSI', 'PWB', 'PWV', 'QQQ',
'RSP', 'RTH', 'RWR', 'SHY', 'SLYG', 'SLYV', 'SMH', 'SOXX', 'SPEU', 'SPTM', 'SPY',
'SPYG', 'SPYV', 'SUSA', 'TIP', 'TLT', 'VAW', 'VB', 'VBK', 'VBR', 'VCR', 'VDC',
'VDE', 'VFH', 'VGK', 'VGT', 'VHT', 'VIS', 'VNQ', 'VO', 'VOX', 'VPL', 'VPU', 'VTI',
'VTV', 'VUG', 'VV', 'VWO', 'VXF', 'XLB', 'XLE', 'XLF', 'XLG', 'XLI', 'XLK', 'XLP',
'XLU', 'XLV', 'XLY', 'XMMO', 'XMVM', 'XNTK', 'XSMO']
```

`get_color_map()`

Build a deterministic HEX color mapping for the current universe.

Colors are drawn from Matplotlib's `tab20` colormap. FX pseudo-tickers for all non-base currencies are appended so that currency series can be plotted alongside ETFs.

Returns
None.

Return type
None

get_weight_cov()

Derive the risk-aversion coefficient used in the objective.

The coefficient is computed from the discrete `risk` as:

```
weight_cov = 52 * exp(-0.3259 * risk) - 2
```

Larger `risk` implies a smaller penalty on variance.

Returns
None.

Return type
None

name = {1: 'Low risk', 2: 'Medium risk', 3: 'High risk'}

threshold_correlation = 0.95

1.8 rebalancer module

Rebalancing utilities to translate optimal weights into actionable trades.

This module exposes *Rebalancer*, which takes an optimized portfolio (*Opti*) and computes:

- Target currency amounts per ticker from optimal weights and total liquidity.
- Dollar (or base-currency) buy/sell differences versus current holdings.
- A tidy pandas DataFrame summarizing the rebalance plan with human-readable before/after allocations.

1.8.1 Workflow

1. Construct *Rebalancer* with an *Opti* instance.
2. It extracts the original holdings, computes target amounts and differences, resolves long names, and assembles `rebalance_df`.

class rebalancer.**Rebalancer**(*opti*)

Bases: `object`

Build a rebalance plan from an optimized portfolio.

Parameters
opti (Opti) – An optimizer instance exposing: * `optimum_all` — mapping `{ticker: weight}` over the current universe. * `portfolio.holdings` — current position values (same currency as liquidity). * `portfolio.liquidity` — cash + current holdings total value. * `portfolio.data.etf_full_names` — pandas Series mapping ticker -> long name.

opti

Reference to the optimizer/portfolio wrapper.

Type
Opti

`goal`

Target currency amounts per ticker (weight × liquidity).

Type
dict[str, float] | None

`difference`

Rounded currency deltas to trade (positive = buy, negative = sell). Only non-zero entries are kept.

Type
dict[str, float] | None

`rebalance_df`

Summary table with columns: `['Ticker', 'ETF', 'Buy/Sell', 'Before', 'After']`.

Type
pandas.DataFrame | None

`full_names`

Mapping from ticker to long display name for tickers in *difference*.

Type
dict[str, str] | None

`original`

Baseline allocation per ticker. For tickers present in current holdings, values are formatted percentage strings like `'12%'`; for new tickers not currently held, the value is `0`.

Type
dict[str, str | float] | None

`get_df()`

Assemble the rebalance summary DataFrame.

The resulting table includes: * `Ticker` — symbol, * `ETF` — long name, * `Buy/Sell` — currency amount to trade (rounded int), * `Before` — current allocation as a percentage string or `0`, * `After` — target allocation as a percentage string.

Rows with an empty target (NaN) **and** `Before == 0` are dropped. The table is sorted by `Buy/Sell` descending.

1.8.2 Side Effects

Sets *rebalance_df*.

returns
None.

rtype
None

`get_difference()`

Compute target amounts and buy/sell differences versus current holdings.

- Target amount per ticker: `goal[t] = weight[t] * liquidity`.
- Difference per ticker: `goal[t] - current_value[t]` (rounded to int).
- Removes zero (after rounding) entries.

1.8.3 Side Effects

Sets *goal* and *difference*.

returns
None.

rtype
None

`get_full_names()`

Resolve long ETF names for tickers that require trades.

1.8.4 Side Effects

Sets *full_names* using `portfolio.data.etf_full_names` for the tickers present in *difference*.

returns
None.

rtype
None

`get_original()`

Build the baseline (current) allocation dictionary.

Converts current holdings `{ticker: value}` into percentage strings relative to the total, e.g., `'8%'`. For tickers that appear in the optimized universe but are not currently held, inserts `0`.

1.8.5 Side Effects

Sets *original*.

returns
None.

rtype
None

PYTHON MODULE INDEX

INDEX

N

O

P

R

S

T

W